Table_of_Contents.txt lines 1 to 60

Table_of_Contents.txt lines 61 to 120

Table_of_Contents.txt lines 121 to 180

README lines 1 to 60

```
This is revision 2.0 of Jerasure.  This is pretty much Jerasure 1.2 without the
original Galois Field backend.  Version 2.0 links directly to GF-Complete, which
is more flexible than the original, and *much* faster, because it leverages SIMD
instructions.

Authors: James S. Plank (University of Tennessee)
         Kevin M. Greenan (Box)

------------------------------------------------------------

External Documentation:

The programmer's manual and tutorial is provided in two places:

1.) A copy is hosted on BitBucket at https://bitbucket.org/jimplank/jerasure/downloads/Jerasure-Manual.pdf
2.) A copy is also available at http://web.eecs.utk.edu/~plank/plank/papers/UT-EECS-14-721.html

See https://bitbucket.org/jimplank/gf-complete for GF-Complete.

NOTE: You must have GF-Complete installed (or compiled) in order to use Jerasure 2.0.

There are two directories of source code:

The src directory contains the jerasure code.
The Examples directory contains the example programs.

------------------------------------------------------------

If you do not have Autoconf 2.65 or later installed, you can simply build
from the tarball distribution:

http://www.kaymgee.com/Kevin_Greenan/Software_files/jerasure.tar.gz

Installing if you are allowed to install GF-Complete on your machine:
(You can skip the autoreconf step if you're using a tarball distribution.)

1.) Install GF-Complete
2.) autoreconf --force --install (*skip* if you are building from tarball)
3.) ./configure
4.) make
5.) sudo make install

This will install the library into your machine's lib directory,
the headers into include, and the example programs into bin.

The configuration process assumes shared objects are searched for in
/usr/local/lib. If this is not the case on your system, you can specify a
search path at configuration time. For example:
  ./configure LD_LIBRARY_PATH=/usr/local/lib

------------------------------------------------------------

Installing if you can compile GF-Complete, but you cannot install it:

1.) Install GF-Complete.  Let's suppose the full path to GF-Complete is
    in the environment variable  GFP
2A.) On Linux, set the environment variable LD_LIBRARY_PATH so that it
     includes $GFP/src/.libs
2B.) On a mac, set the environment variable DYLD_LIBRARY_PATH so that it
     includes $GFP/src/.libs
```

```
2.) ./configure LDFLAGS=-L$GFP/src/.libs/ CPPFLAGS=-I$GFP/include
3.) make

The examples will be in the directory Examples.  The include files will
be in the directory include, and the library will be called libJerasure.a
in the directory src/.libs.

------------------------------------------------------------

As long as GF-Complete is installed, Jerasure 2.0 can be used just as previous
versions.  There is no need to define custom Galois Fields.  Jerasure will
determine the default field to use, if one is not specified.

If you would like to explore a using a different Galois Field implementation,
please see the manual.

------------------------------------------------------------

Testing GF-Complete

If the GF-Complete tools are installed in /usr/local/bin

  make check

If the GF-Complete tools are installed elsewhere

  make GF_COMPLETE_DIR=$(pwd)/../gf-complete/tools check

To run some tests with valgrind

  make VALGRIND='valgrind --tool=memcheck --quiet' \
       GF_COMPLETE_DIR=$(pwd)/../gf-complete/tools \
       check
```

License.txt lines 1 to 37

Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure Coding Techniques

Revision 2.0: Galois Field backend now links to GF-Complete

src/cauchy.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "galois.h"
#include "jerasure.h"
#include "cauchy.h"

static int PPs[33] = { -1, -1, -1, -1, -1, -1, -1, -1,
                       -1, -1, -1, -1, -1, -1, -1, -1,
                       -1, -1, -1, -1, -1, -1, -1, -1,
                       -1, -1, -1, -1, -1, -1, -1, -1, -1 };
static int NOs[33];
static int ONEs[33][33];
```

src/cauchy.c lines 61 to 120

```
static int *cbest_0;
static int *cbest_1;
static int cbest_2[3];
static int cbest_3[7];
static int cbest_4[15];
static int cbest_5[31];
static int cbest_6[63];
static int cbest_7[127];
static int cbest_8[255];
static int cbest_9[511];
static int cbest_10[1023];
static int cbest_11[1023];
static int *cbest_12, *cbest_13, *cbest_14, *cbest_15, *cbest_16, *cbest_17, *cbest_18, *cbest_19, *cbest_20,
           *cbest_21, *cbest_22, *cbest_23, *cbest_24, *cbest_25, *cbest_26, *cbest_27, *cbest_28, *cbest_29, *cbest_30,
           *cbest_31, *cbest_32;

static int cbest_max_k[33] = { -1, -1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 1023, -1,
     -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
     -1, -1, -1, -1 };

static int cbest_init = 0;

static int *cbest_all[33];


#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

int cauchy_n_ones(int n, int w)
{
  int no;
  int cno;
  int nones;
  int i, j;
  int highbit;

  highbit = (1 << (w-1));

  if (PPs[w] == -1) {
    nones = 0;
    PPs[w] = galois_single_multiply(highbit, 2, w);
    for (i = 0; i < w; i++) {
      if (PPs[w] & (1 << i)) {
        ONEs[w][nones] = (1 << i);
        nones++;
      }
    }
    NOs[w] = nones;
  }

  no = 0;
  for (i = 0; i < w; i++) if (n & (1 << i)) no++;
  cno = no;
  for (i = 1; i < w; i++) {
    if (n & highbit) {
      n ^= highbit;
      n <<= 1;
      n ^= PPs[w];
      cno--;
      for (j = 0; j < NOs[w]; j++) {
```

```
        cno += (n & ONEs[w][j]) ? 1 : -1;
      }
    } else {
      n <<= 1;
    }
    no += cno;
  }
  return no;
}

int *cauchy_original_coding_matrix(int k, int m, int w)
{
  int *matrix;
  int i, j, index;

  if (w < 31 && (k+m) > (1 << w)) return NULL;
  matrix = talloc(int, k*m);
  if (matrix == NULL) return NULL;
  index = 0;
  for (i = 0; i < m; i++) {
    for (j = 0; j < k; j++) {
      matrix[index] = galois_single_divide(1, (i ^ (m+j)), w);
      index++;
    }
  }
  return matrix;
}

int *cauchy_xy_coding_matrix(int k, int m, int w, int *X, int *Y)
{
  int index, i, j;
  int *matrix;

  matrix = talloc(int, k*m);
  if (matrix == NULL) { return NULL; }
  index = 0;
  for (i = 0; i < m; i++) {
    for (j = 0; j < k; j++) {
      matrix[index] = galois_single_divide(1, (X[i] ^ Y[j]), w);
      index++;
    }
  }
  return matrix;
}

void cauchy_improve_coding_matrix(int k, int m, int w, int *matrix)
{
  int index, i, j, x;
  int tmp;
  int bno, tno, bno_index;

  for (j = 0; j < k; j++) {
    if (matrix[j] != 1) {
      tmp = galois_single_divide(1, matrix[j], w);
      index = j;
      for (i = 0; i < m; i++) {
        matrix[index] = galois_single_multiply(matrix[index], tmp, w);
        index += k;
      }
    }
```

```
  }
  for (i = 1; i < m; i++) {
    bno = 0;
    index = i*k;
    for (j = 0; j < k; j++) bno += cauchy_n_ones(matrix[index+j], w);
    bno_index = -1;
    for (j = 0; j < k; j++) {
      if (matrix[index+j] != 1) {
        tmp = galois_single_divide(1, matrix[index+j], w);
        tno = 0;
        for (x = 0; x < k; x++) {
          tno += cauchy_n_ones(galois_single_multiply(matrix[index+x], tmp, w), w);
        }
        if (tno < bno) {
          bno = tno;
          bno_index = j;
        }
      }
    }
    if (bno_index != -1) {
      tmp = galois_single_divide(1, matrix[index+bno_index], w);
      for (j = 0; j < k; j++) {
        matrix[index+j] = galois_single_multiply(matrix[index+j], tmp, w);
      }
    }
  }
}

int *cauchy_good_general_coding_matrix(int k, int m, int w)
{
  int *matrix, i;

  if (m == 2 && k <= cbest_max_k[w]) {
    matrix = talloc(int, k*m);
    if (matrix == NULL) return NULL;
    if (!cbest_init) {
      cbest_init = 1;
      cbest_all[0] = cbest_0; cbest_all[1] = cbest_1; cbest_all[2] = cbest_2; cbest_all[3] = cbest_3; cbest_all[4] =
      cbest_4; cbest_all[5] = cbest_5; cbest_all[6] = cbest_6; cbest_all[7] = cbest_7; cbest_all[8] = cbest_8;
      cbest_all[9] = cbest_9; cbest_all[10] = cbest_10; cbest_all[11] = cbest_11; cbest_all[12] = cbest_12;
      cbest_all[13] = cbest_13; cbest_all[14] = cbest_14; cbest_all[15] = cbest_15; cbest_all[16] = cbest_16;
      cbest_all[17] = cbest_17; cbest_all[18] = cbest_18; cbest_all[19] = cbest_19; cbest_all[20] = cbest_20;
      cbest_all[21] = cbest_21; cbest_all[22] = cbest_22; cbest_all[23] = cbest_23; cbest_all[24] = cbest_24;
      cbest_all[25] = cbest_25; cbest_all[26] = cbest_26; cbest_all[27] = cbest_27; cbest_all[28] = cbest_28;
      cbest_all[29] = cbest_29; cbest_all[30] = cbest_30; cbest_all[31] = cbest_31; cbest_all[32] = (int *) cbest_32;
    }
    for (i = 0; i < k; i++) {
      matrix[i] = 1;
      matrix[i+k] = cbest_all[w][i];
    }
    return matrix;
  } else {
    matrix = cauchy_original_coding_matrix(k, m, w);
    if (matrix == NULL) return NULL;
    cauchy_improve_coding_matrix(k, m, w, matrix);
    return matrix;
  }
}

static int cbest_2[3] = { 1, 2, 3 };
```

src/cauchy.c lines 241 to 300

```
static int cbest_3[7] = { 1, 2, 5, 4, 7, 3, 6 };

static int cbest_4[15] = { 1, 2, 9, 4, 8, 13, 3, 6, 12, 5, 11, 15, 10, 14, 7 };

static int cbest_5[31] = { 1, 2, 18, 4, 9, 8, 22, 16, 3, 11, 19, 5, 10, 6, 20, 27, 13, 23, 26, 12,
    17, 25, 24, 31, 30, 7, 15, 21, 29, 14, 28 };

static int cbest_6[63] = { 1, 2, 33, 4, 8, 49, 16, 32, 57, 3, 6, 12, 24, 48, 5, 35, 9, 37, 10, 17,
    41, 51, 56, 61, 18, 28, 53, 14, 20, 34, 7, 13, 25, 36, 59, 26, 39, 40, 45, 50, 60, 52, 63,
    11, 30, 55, 19, 22, 29, 43, 58, 15, 21, 38, 44, 47, 62, 27, 54, 42, 31, 23, 46 };

static int cbest_7[127] = { 1, 2, 68, 4, 34, 8, 17, 16, 76, 32, 38, 3, 64, 69, 5, 19, 35, 70, 6, 9,
    18, 102, 10, 36, 85, 12, 21, 42, 51, 72, 77, 84, 20, 25, 33, 50, 78, 98, 24, 39, 49, 100, 110
   , 48, 65, 93, 40, 66, 71, 92, 7, 46, 55, 87, 96, 103, 106, 11, 23, 37, 54, 81, 86, 108, 13,
    22, 27, 43, 53, 73, 80, 14, 26, 52, 74, 79, 99, 119, 44, 95, 101, 104, 111, 118, 29, 59, 89,
    94, 117, 28, 41, 58, 67, 88, 115, 116, 47, 57, 83, 97, 107, 114, 127, 56, 82, 109, 113, 126,
    112, 125, 15, 63, 75, 123, 124, 31, 45, 62, 91, 105, 122, 30, 61, 90, 121, 60, 120 };

static int cbest_8[255] = { 1, 2, 142, 4, 71, 8, 70, 173, 3, 35, 143, 16, 17, 67, 134, 140, 172, 6, 34
   , 69, 201, 216, 5, 33, 86, 12, 65, 138, 158, 159, 175, 10, 32, 43, 66, 108, 130, 193, 234, 9,
    24, 25, 50, 68, 79, 100, 132, 174, 200, 217, 20, 21, 42, 48, 87, 169, 41, 54, 64, 84, 96, 117
   , 154, 155, 165, 226, 77, 82, 135, 136, 141, 168, 192, 218, 238, 7, 18, 19, 39, 40, 78, 113,
    116, 128, 164, 180, 195, 205, 220, 232, 14, 26, 27, 58, 109, 156, 157, 203, 235, 13, 28, 29, 38
   , 51, 56, 75, 85, 90, 101, 110, 112, 139, 171, 11, 37, 49, 52, 76, 83, 102, 119, 131, 150, 151
   , 167, 182, 184, 188, 197, 219, 224, 45, 55, 80, 94, 97, 133, 170, 194, 204, 221, 227, 236, 36,
    47, 73, 92, 98, 104, 118, 152, 153, 166, 202, 207, 239, 251, 22, 23, 44, 74, 91, 148, 149, 161
   , 181, 190, 233, 46, 59, 88, 137, 146, 147, 163, 196, 208, 212, 222, 250, 57, 81, 95, 106, 111,
    129, 160, 176, 199, 243, 249, 15, 53, 72, 93, 103, 115, 125, 162, 183, 185, 189, 206, 225, 255,
    186, 210, 230, 237, 242, 248, 30, 31, 62, 89, 99, 105, 114, 121, 124, 178, 209, 213, 223, 228,
    241, 254, 60, 191, 198, 247, 120, 240, 107, 127, 144, 145, 177, 211, 214, 246, 245, 123, 126,
    187, 231, 253, 63, 179, 229, 244, 61, 122, 215, 252 };

static int cbest_9[511] = { 1, 2, 264, 4, 132, 8, 66, 16, 33, 32, 280, 64, 140, 128, 3, 70, 265, 5,
    133, 256, 266, 6, 9, 35, 67, 134, 268, 396, 10, 17, 34, 330, 12, 18, 68, 198, 297, 20, 37, 74
   , 136, 148, 165, 281, 296, 24, 36, 41, 65, 82, 99, 164, 272, 282, 388, 40, 49, 98, 141, 194,
    284, 328, 412, 48, 97, 129, 142, 196, 346, 71, 72, 96, 130, 313, 392, 80, 206, 257, 267, 312,
    334, 7, 135, 156, 173, 192, 258, 269, 397, 404, 11, 78, 144, 161, 172, 260, 270, 299, 331, 344,
    398, 13, 19, 39, 69, 86, 103, 160, 167, 199, 202, 298, 322, 384, 14, 21, 38, 43, 75, 102, 137,
    149, 166, 204, 289, 332, 408, 462, 22, 25, 42, 51, 83, 101, 138, 150, 273, 283, 288, 301, 350,
    389, 429, 26, 50, 76, 100, 195, 274, 285, 300, 329, 363, 390, 413, 428, 28, 45, 84, 143, 197,
    200, 214, 231, 276, 286, 315, 320, 347, 362, 414, 458, 44, 53, 73, 90, 107, 131, 152, 169, 181,
    230, 314, 338, 361, 393, 400, 454, 460, 52, 57, 81, 106, 115, 168, 175, 180, 207, 229, 305, 335
   , 348, 360, 394, 421, 478, 56, 105, 114, 157, 163, 174, 193, 210, 227, 228, 259, 304, 317, 326,
    405, 420, 445, 79, 104, 113, 145, 158, 162, 212, 226, 261, 271, 316, 345, 379, 399, 406, 444,
    450, 456, 87, 88, 112, 146, 203, 225, 262, 291, 323, 336, 378, 385, 425, 452, 474, 15, 205, 222
   , 224, 239, 290, 303, 333, 367, 377, 386, 409, 424, 431, 463, 470, 476, 23, 139, 151, 189, 208,
    238, 302, 324, 351, 366, 376, 410, 430, 437, 27, 47, 77, 94, 111, 177, 188, 237, 275, 293, 342,
    365, 391, 436, 448, 29, 46, 55, 85, 110, 119, 171, 176, 183, 201, 215, 218, 235, 236, 277, 287,
    292, 321, 355, 364, 415, 417, 459, 466, 472, 30, 54, 59, 91, 109, 118, 153, 170, 182, 220, 234,
    278, 307, 339, 354, 401, 416, 423, 441, 455, 461, 468, 495, 58, 108, 117, 154, 233, 306, 319,
    349, 353, 383, 395, 402, 422, 440, 447, 479, 494, 92, 116, 211, 232, 318, 327, 340, 352, 382,
    446, 493, 61, 159, 213, 216, 247, 309, 381, 407, 427, 451, 457, 464, 491, 492, 60, 89, 123, 147
   , 185, 246, 263, 308, 337, 371, 380, 426, 433, 453, 475, 487, 490, 122, 184, 191, 223, 245, 370,
    387, 432, 439, 471, 477, 486, 489, 511, 121, 179, 190, 209, 243, 244, 295, 325, 359, 369, 411,
    438, 485, 488, 510, 95, 120, 178, 242, 294, 343, 358, 368, 419, 449, 483, 484, 509, 219, 241,
    357, 418, 443, 467, 473, 482, 507, 508, 31, 221, 240, 255, 279, 356, 442, 469, 481, 503, 506,
    155, 254, 403, 480, 502, 505, 63, 93, 127, 253, 311, 341, 375, 501, 504, 62, 126, 187, 217, 251
   , 252, 310, 374, 435, 465, 499, 500, 125, 186, 250, 373, 434, 498, 124, 249, 372, 497, 248, 496
    };
```

```
static int cbest_10[1023] = { 1, 2, 516, 4, 258, 8, 129, 16, 32, 580, 64, 128, 290, 145, 256, 3, 512,
    517, 5, 259, 518, 588, 6, 9, 18, 36, 72, 144, 774, 10, 17, 131, 262, 288, 524, 645, 12, 33,
    133, 266, 294, 387, 532, 576, 581, 20, 34, 65, 137, 274, 548, 582, 24, 66, 291, 838, 40, 68,
    130, 147, 161, 322, 644, 709, 806, 48, 132, 193, 257, 386, 596, 80, 136, 298, 419, 612, 661, 772
   , 96, 149, 260, 272, 306, 403, 513, 146, 153, 160, 264, 292, 385, 514, 519, 544, 584, 589, 708,
    870, 7, 19, 37, 73, 192, 354, 590, 770, 775, 11, 38, 74, 177, 263, 289, 418, 520, 525, 534, 641
   , 660, 725, 802, 836, 846, 13, 22, 76, 148, 209, 267, 295, 320, 330, 402, 526, 528, 533, 577,
    647, 717, 804, 14, 21, 26, 35, 44, 135, 152, 165, 201, 275, 304, 384, 401, 435, 549, 578, 583,
    604, 608, 782, 903, 25, 52, 67, 88, 139, 270, 296, 391, 417, 550, 620, 653, 790, 834, 839, 41,
    50, 69, 104, 141, 176, 278, 302, 323, 395, 423, 540, 598, 640, 705, 724, 807, 866, 28, 42, 49,
    70, 82, 100, 163, 208, 282, 310, 556, 592, 597, 646, 663, 677, 711, 716, 868, 878, 81, 134, 151
   , 164, 195, 200, 299, 326, 352, 362, 400, 434, 564, 613, 657, 768, 773, 902, 967, 97, 138, 155,
    169, 197, 261, 273, 307, 358, 390, 416, 433, 451, 614, 652, 733, 800, 814, 844, 854, 935, 56, 84
   , 98, 140, 181, 217, 265, 293, 328, 338, 394, 422, 515, 545, 585, 704, 788, 822, 871, 919, 162,
    179, 276, 355, 407, 427, 546, 586, 591, 616, 662, 669, 676, 710, 727, 741, 771, 780, 901, 39, 75
   , 150, 157, 194, 211, 225, 268, 280, 308, 314, 389, 411, 439, 521, 530, 535, 628, 656, 721, 803,
    832, 837, 842, 847, 966, 23, 77, 112, 154, 168, 196, 300, 321, 331, 393, 421, 432, 450, 522, 527
   , 529, 552, 606, 643, 673, 693, 713, 732, 805, 864, 874, 934, 999, 15, 27, 45, 54, 78, 90, 108,
    180, 216, 305, 483, 560, 579, 600, 605, 609, 719, 778, 783, 852, 876, 886, 899, 918, 983, 46, 53
   , 89, 167, 178, 185, 203, 213, 271, 297, 324, 334, 336, 360, 370, 406, 426, 467, 542, 551, 610,
    621, 649, 668, 726, 740, 786, 791, 810, 820, 835, 900, 917, 931, 951, 965, 975, 30, 51, 105, 156
   , 205, 210, 224, 279, 303, 356, 366, 388, 405, 410, 438, 449, 459, 536, 541, 594, 599, 622, 655,
    720, 812, 818, 862, 867, 933, 29, 43, 71, 83, 92, 101, 106, 143, 173, 283, 311, 312, 346, 392,
    409, 420, 437, 443, 557, 566, 593, 642, 659, 672, 692, 707, 712, 737, 757, 869, 879, 911, 998,
    60, 102, 241, 327, 353, 363, 399, 425, 482, 558, 565, 624, 679, 718, 735, 749, 769, 798, 898,
    963, 982, 58, 86, 166, 183, 184, 202, 212, 219, 233, 286, 359, 431, 466, 615, 636, 648, 689, 729
   , 801, 815, 840, 845, 850, 855, 884, 916, 930, 950, 964, 974, 981, 995, 1015, 57, 85, 99, 120,
    171, 199, 204, 229, 318, 329, 339, 368, 404, 448, 458, 465, 499, 654, 671, 685, 784, 789, 823,
    872, 882, 915, 932, 949, 997, 1007, 116, 142, 159, 172, 277, 408, 436, 442, 455, 481, 491, 547,
    572, 587, 617, 630, 658, 665, 706, 723, 736, 756, 776, 781, 816, 860, 894, 897, 910, 947, 991,
    114, 221, 240, 269, 281, 309, 315, 332, 342, 344, 378, 398, 424, 441, 475, 487, 531, 618, 629,
    678, 695, 734, 743, 748, 808, 833, 843, 929, 943, 962, 973, 113, 182, 189, 218, 227, 232, 301,
    364, 374, 430, 457, 523, 553, 562, 602, 607, 688, 728, 753, 796, 830, 865, 875, 927, 980, 994,
    1014, 55, 79, 91, 109, 170, 187, 198, 215, 228, 284, 415, 464, 498, 554, 561, 601, 670, 675, 684
   , 715, 745, 765, 779, 848, 853, 877, 887, 909, 914, 948, 979, 996, 1006, 1013, 47, 110, 158, 249
   , 316, 325, 335, 337, 361, 371, 397, 447, 454, 480, 490, 497, 538, 543, 611, 632, 664, 722, 787,
    811, 821, 880, 896, 913, 946, 961, 971, 990, 1011, 31, 94, 220, 245, 357, 367, 429, 440, 474,
    486, 537, 595, 623, 651, 681, 694, 701, 742, 759, 813, 819, 858, 863, 892, 928, 942, 945, 972,
    989, 993, 1003, 1023, 62, 93, 107, 188, 207, 226, 237, 243, 313, 340, 347, 376, 456, 471, 473,
    507, 567, 568, 626, 752, 890, 907, 926, 1005, 61, 103, 124, 175, 186, 214, 372, 414, 453, 463,
    489, 503, 559, 625, 638, 674, 691, 714, 731, 739, 744, 764, 794, 799, 828, 908, 925, 939, 959,
    978, 1012, 59, 87, 122, 248, 287, 350, 396, 413, 446, 485, 495, 496, 637, 751, 826, 841, 851,
    885, 912, 941, 960, 970, 977, 1010, 118, 121, 235, 244, 319, 369, 382, 428, 445, 574, 650, 667,
    680, 700, 758, 761, 785, 873, 883, 944, 988, 992, 1002, 1009, 1022, 117, 206, 223, 231, 236, 242
   , 470, 472, 506, 573, 631, 687, 777, 817, 856, 861, 895, 906, 987, 1004, 1021, 115, 174, 191, 333
   , 343, 345, 379, 452, 462, 469, 488, 502, 505, 619, 690, 697, 730, 738, 755, 809, 888, 924, 938,
    958, 969, 1019, 253, 365, 375, 412, 484, 494, 501, 563, 603, 750, 767, 792, 797, 831, 923, 940,
    957, 976, 1001, 234, 251, 285, 348, 444, 479, 555, 634, 666, 760, 824, 849, 905, 955, 1008, 111,
    222, 230, 247, 317, 380, 461, 511, 539, 633, 686, 703, 747, 881, 937, 986, 1020, 95, 190, 468,
    493, 504, 570, 696, 754, 859, 893, 968, 985, 1018, 63, 126, 252, 341, 377, 500, 569, 627, 683,
    766, 891, 922, 956, 1000, 1017, 125, 239, 250, 373, 478, 639, 795, 829, 904, 921, 954, 123, 246,
    351, 460, 477, 510, 702, 746, 763, 827, 936, 953, 119, 383, 492, 509, 575, 984, 682, 699, 857,
    1016, 238, 255, 889, 920, 476, 762, 793, 952, 349, 508, 635, 825, 381, 698, 254, 571, 127 };

static int cbest_11[1023] = { 1,
    2, 1026, 4, 513, 8, 16, 1282, 32, 64, 641, 128, 256, 512, 1346, 1024, 3, 673, 1027, 5, 10, 20, 40, 80, 160, 320,
    640, 6, 9, 515, 1030, 1280, 1539, 17, 517, 1034, 1283, 12, 18, 33, 521, 1042, 1362, 34, 65, 529, 1058, 1286, 1795,
    24, 36, 66, 129, 545, 643, 1090, 1290, 1667, 68, 130, 257, 577, 645, 672, 1154, 1298, 1344, 48, 72, 132, 258, 336,
    649, 681, 1314, 1347, 136, 168, 260, 514, 657, 769, 1538, 1923, 84, 96, 144, 264, 516, 1025, 1350, 1410, 1859, 42,
    272, 520, 705, 1032, 1354, 11, 21, 41, 81, 161, 192, 288, 321, 528, 675, 1028, 1537, 1699, 1794, 7, 22, 82, 162,
```

```
    322, 544, 642, 677, 897, 1031, 1046, 1066, 1106, 1186, 1281, 1366, 1378, 1666, 14, 44, 164, 324, 384, 523, 533,
    553, 576, 593, 644, 833, 1035, 1040, 1288, 1360, 1987, 13, 19, 28, 88, 328, 519, 648, 680, 689, 1043, 1056, 1284,
    1363, 1474, 1543, 1793, 1955, 26, 35, 56, 176, 656, 768, 1038, 1059, 1088, 1287, 1302, 1322, 1442, 1547, 1665,
    1922, 25, 37, 52, 67, 112, 340, 352, 525, 531, 737, 1091, 1152, 1291, 1296, 1555, 1858, 1875, 38, 69, 74, 104, 131,
    224, 547, 651, 661, 683, 704, 721, 961, 1050, 1062, 1155, 1299, 1312, 1345, 1370, 1571, 1799, 49, 70, 73, 133, 138,
    148, 170, 208, 259, 337, 448, 537, 549, 579, 647, 674, 929, 1094, 1294, 1315, 1352, 1536, 1603, 1671, 1698, 1803,
    1921, 50, 134, 137, 169, 261, 266, 276, 296, 338, 416, 581, 676, 896, 1074, 1098, 1158, 1348, 1394, 1408, 1675,
    1707, 1811, 1857, 2019, 76, 85, 97, 145, 262, 265, 522, 532, 552, 561, 585, 592, 653, 659, 685, 771, 832, 849,
    1064, 1162, 1194, 1306, 1318, 1351, 1386, 1411, 1506, 1683, 1827, 1986, 2003, 43, 86, 98, 140, 146, 172, 273, 344,
    518, 688, 773, 1033, 1110, 1122, 1170, 1355, 1490, 1542, 1697, 1792, 1927, 1954, 100, 193, 268, 274, 289, 597, 609,
    665, 697, 707, 777, 1029, 1044, 1104, 1184, 1330, 1364, 1376, 1414, 1546, 1664, 1731, 1863, 1931, 1963, 23, 46, 83,
    92, 152, 163, 184, 194, 290, 323, 368, 524, 530, 555, 693, 709, 736, 753, 785, 993, 1036, 1047, 1067, 1107, 1187,
    1218, 1320, 1358, 1367, 1379, 1418, 1450, 1545, 1554, 1867, 1874, 1939, 1985, 15, 30, 45, 60, 90, 120, 165, 180,
    196, 240, 280, 292, 325, 330, 360, 385, 480, 546, 650, 660, 679, 682, 713, 720, 745, 801, 899, 960, 977, 1041,
    1289, 1361, 1426, 1472, 1541, 1570, 1703, 1798, 1953, 29, 58, 89, 116, 166, 200, 232, 326, 329, 386, 464, 535, 536,
    548, 578, 595, 646, 835, 901, 928, 1048, 1057, 1070, 1190, 1285, 1300, 1368, 1382, 1440, 1475, 1559, 1579, 1602,
    1619, 1670, 1802, 1879, 1891, 1920, 27, 57, 177, 304, 388, 527, 557, 580, 691, 725, 837, 905, 937, 1039, 1054,
    1089, 1114, 1292, 1303, 1323, 1374, 1443, 1553, 1674, 1706, 1715, 1801, 1810, 1856, 1873, 1991, 2018, 2035, 53,
    106, 113, 178, 212, 332, 341, 353, 392, 424, 541, 560, 584, 601, 652, 658, 684, 770, 841, 848, 913, 1060, 1082,
    1096, 1153, 1202, 1297, 1402, 1478, 1522, 1569, 1673, 1682, 1705, 1797, 1826, 1959, 1995, 2002, 2027, 39, 54, 75,
    105, 114, 225, 342, 354, 400, 539, 569, 739, 772, 1051, 1063, 1078, 1092, 1138, 1160, 1192, 1304, 1313, 1326, 1371,
    1384, 1398, 1446, 1482, 1514, 1551, 1601, 1669, 1696, 1763, 1815, 1835, 1926, 71, 139, 149, 171, 209, 226, 298,
    356, 449, 565, 596, 608, 625, 663, 664, 696, 706, 723, 741, 776, 853, 865, 963, 1072, 1095, 1130, 1156, 1250, 1295,
    1310, 1353, 1392, 1687, 1730, 1747, 1809, 1862, 1930, 1962, 1971, 2007, 2017, 51, 78, 108, 135, 150, 210, 228, 267,
    277, 297, 339, 348, 417, 450, 551, 554, 587, 617, 655, 687, 692, 708, 752, 784, 931, 965, 992, 1009, 1075, 1099,
    1159, 1174, 1234, 1316, 1338, 1349, 1395, 1409, 1458, 1494, 1504, 1544, 1563, 1575, 1681, 1825, 1866, 1883, 1929,
    1938, 1961, 1984, 2001, 77, 142, 174, 263, 278, 346, 376, 418, 452, 496, 583, 669, 678, 701, 712, 729, 744, 761,
    800, 898, 933, 969, 976, 1001, 1065, 1108, 1120, 1163, 1168, 1195, 1307, 1319, 1334, 1356, 1387, 1416, 1448, 1488,
    1507, 1540, 1607, 1702, 1807, 1865, 1925, 1952, 87, 99, 141, 147, 156, 173, 188, 216, 248, 270, 300, 345, 372, 420,
    456, 488, 534, 563, 594, 667, 699, 757, 779, 789, 809, 834, 851, 900, 1102, 1111, 1123, 1171, 1328, 1412, 1491,
    1558, 1578, 1587, 1611, 1618, 1679, 1711, 1729, 1861, 1878, 1890, 1907, 1943, 2023, 94, 101, 124, 154, 186, 244,
    269, 275, 284, 526, 556, 589, 690, 724, 775, 836, 904, 936, 945, 981, 1045, 1068, 1105, 1166, 1185, 1198, 1216,
    1331, 1365, 1377, 1390, 1415, 1430, 1510, 1552, 1577, 1714, 1800, 1819, 1831, 1872, 1899, 1937, 1990, 2034, 47, 62,
    93, 102, 122, 153, 185, 195, 282, 291, 312, 362, 369, 432, 468, 540, 599, 600, 611, 715, 747, 840, 857, 912, 1037,
    1052, 1112, 1126, 1219, 1321, 1359, 1372, 1419, 1424, 1451, 1568, 1623, 1635, 1672, 1691, 1701, 1704, 1723, 1796,
    1958, 1994, 2011, 2026, 2043, 31, 61, 91, 121, 181, 197, 202, 234, 241, 281, 293, 308, 331, 361, 370, 481, 538,
    568, 613, 695, 711, 738, 755, 781, 787, 995, 1080, 1118, 1178, 1188, 1210, 1380, 1400, 1427, 1473, 1498, 1530,
    1550, 1557, 1600, 1617, 1668, 1719, 1735, 1762, 1779, 1814, 1834, 1843, 1877, 1889, 1935, 1967, 1993, 2025, 2039,
    59, 117, 167, 182, 198, 201, 233, 242, 294, 327, 387, 465, 482, 559, 564, 605, 624, 662, 722, 740, 803, 852, 864,
    881, 907, 917, 939, 962, 979, 997, 1049, 1071, 1086, 1146, 1191, 1206, 1222, 1266, 1301, 1324, 1369, 1383, 1406,
    1422, 1441, 1454, 1480, 1512, 1526, 1549, 1686, 1713, 1739, 1746, 1771, 1808, 1833, 1871, 1970, 1989, 2006, 2016,
    2033, 118, 305, 334, 364, 389, 394, 404, 426, 466, 484, 543, 550, 573, 586, 603, 616, 633, 654, 686, 717, 749, 793,
    805, 843, 873, 903, 930, 964, 1008, 1055, 1115, 1128, 1142, 1200, 1226, 1258, 1293, 1308, 1375, 1476, 1520, 1562,
    1574, 1680, 1824 };
```

src/cauchy_best_r6.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "galois.h"
#include "jerasure.h"
#include "cauchy.h"

static int PPs[33] = { -1, -1, -1, -1, -1, -1, -1, -1,
                       -1, -1, -1, -1, -1, -1, -1, -1,
                       -1, -1, -1, -1, -1, -1, -1, -1,
                       -1, -1, -1, -1, -1, -1, -1, -1, -1 };
static int NOs[33];
static int ONEs[33][33];
```

src/cauchy_best_r6.c lines 61 to 120

```c
static int *cbest_0;
static int *cbest_1;
static int cbest_2[3];
static int cbest_3[7];
static int cbest_4[15];
static int cbest_5[31];
static int cbest_6[63];
static int cbest_7[127];
static int cbest_8[255];
static int cbest_9[511];
static int cbest_10[1023];
static int cbest_11[1023];
static int cbest_12[1023], cbest_13[1023], cbest_14[1023], cbest_15[1023],
           cbest_16[1023], cbest_17[1023], cbest_18[1023],
           cbest_19[1023], cbest_20[1023], cbest_21[1023], cbest_22[1023],
           cbest_23[1023], cbest_24[1023], cbest_25[1023],
           cbest_26[1023], cbest_27[1023], cbest_28[1023], cbest_29[1023],
           cbest_30[1023], cbest_31[1023];
static unsigned int cbest_32[1023];

static int cbest_max_k[33] = { -1, -1, 3, 7, 15, 31, 63, 127, 255, 511, 1023, 1023,
     1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023, 1023,
     1023, 1023, 1023, 1023 };

static int cbest_init = 0;

static int *cbest_all[33];


#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

int cauchy_n_ones(int n, int w)
{
  int no;
  int cno;
  int nones;
  int i, j;
  int highbit;

  highbit = (1 << (w-1));

  if (PPs[w] == -1) {
    nones = 0;
    PPs[w] = galois_single_multiply(highbit, 2, w);
    for (i = 0; i < w; i++) {
      if (PPs[w] & (1 << i)) {
        ONEs[w][nones] = (1 << i);
        nones++;
      }
    }
    NOs[w] = nones;
  }

  no = 0;
  for (i = 0; i < w; i++) if (n & (1 << i)) no++;
  cno = no;
  for (i = 1; i < w; i++) {
    if (n & highbit) {
      n ^= highbit;
```

```
      n <<= 1;
      n ^= PPs[w];
      cno--;
      for (j = 0; j < NOs[w]; j++) {
        cno += (n & ONEs[w][j]) ? 1 : -1;
      }
    } else {
      n <<= 1;
    }
    no += cno;
  }
  return no;
}

int *cauchy_original_coding_matrix(int k, int m, int w)
{
  int *matrix;
  int i, j, index;

  if (w < 31 && (k+m) > (1 << w)) return NULL;
  matrix = talloc(int, k*m);
  if (matrix == NULL) return NULL;
  index = 0;
  for (i = 0; i < m; i++) {
    for (j = 0; j < k; j++) {
      matrix[index] = galois_single_divide(1, (i ^ (m+j)), w);
      index++;
    }
  }
  return matrix;
}

int *cauchy_xy_coding_matrix(int k, int m, int w, int *X, int *Y)
{
  int index, i, j;
  int *matrix;

  matrix = talloc(int, k*m);
  if (matrix == NULL) { return NULL; }
  index = 0;
  for (i = 0; i < m; i++) {
    for (j = 0; j < k; j++) {
      matrix[index] = galois_single_divide(1, (X[i] ^ Y[j]), w);
      index++;
    }
  }
  return matrix;
}

void cauchy_improve_coding_matrix(int k, int m, int w, int *matrix)
{
  int index, i, j, x;
  int tmp;
  int bno, tno, bno_index;

  for (j = 0; j < k; j++) {
    if (matrix[j] != 1) {
      tmp = galois_single_divide(1, matrix[j], w);
      index = j;
      for (i = 0; i < m; i++) {
```

```
        matrix[index] = galois_single_multiply(matrix[index], tmp, w);
        index += k;
      }
    }
  }
  for (i = 1; i < m; i++) {
    bno = 0;
    index = i*k;
    for (j = 0; j < k; j++) bno += cauchy_n_ones(matrix[index+j], w);
    bno_index = -1;
    for (j = 0; j < k; j++) {
      if (matrix[index+j] != 1) {
        tmp = galois_single_divide(1, matrix[index+j], w);
        tno = 0;
        for (x = 0; x < k; x++) {
          tno += cauchy_n_ones(galois_single_multiply(matrix[index+x], tmp, w), w);
        }
        if (tno < bno) {
          bno = tno;
          bno_index = j;
        }
      }
    }
    if (bno_index != -1) {
      tmp = galois_single_divide(1, matrix[index+bno_index], w);
      for (j = 0; j < k; j++) {
        matrix[index+j] = galois_single_multiply(matrix[index+j], tmp, w);
      }
    }
  }
}

int *cauchy_good_general_coding_matrix(int k, int m, int w)
{
  int *matrix, i;

  if (m == 2 && k <= cbest_max_k[w]) {
    matrix = talloc(int, k*m);
    if (matrix == NULL) return NULL;
    if (!cbest_init) {
      cbest_init = 1;
      cbest_all[0] = cbest_0; cbest_all[1] = cbest_1; cbest_all[2] = cbest_2; cbest_all[3] = cbest_3; cbest_all[4] =
      cbest_4; cbest_all[5] = cbest_5; cbest_all[6] = cbest_6; cbest_all[7] = cbest_7; cbest_all[8] = cbest_8;
      cbest_all[9] = cbest_9; cbest_all[10] = cbest_10; cbest_all[11] = cbest_11; cbest_all[12] = cbest_12;
      cbest_all[13] = cbest_13; cbest_all[14] = cbest_14; cbest_all[15] = cbest_15; cbest_all[16] = cbest_16;
      cbest_all[17] = cbest_17; cbest_all[18] = cbest_18; cbest_all[19] = cbest_19; cbest_all[20] = cbest_20;
      cbest_all[21] = cbest_21; cbest_all[22] = cbest_22; cbest_all[23] = cbest_23; cbest_all[24] = cbest_24;
      cbest_all[25] = cbest_25; cbest_all[26] = cbest_26; cbest_all[27] = cbest_27; cbest_all[28] = cbest_28;
      cbest_all[29] = cbest_29; cbest_all[30] = cbest_30; cbest_all[31] = cbest_31; cbest_all[32] = (int *) cbest_32;
    }
    for (i = 0; i < k; i++) {
      matrix[i] = 1;
      matrix[i+k] = cbest_all[w][i];
    }
    return matrix;
  } else {
    matrix = cauchy_original_coding_matrix(k, m, w);
    if (matrix == NULL) return NULL;
    cauchy_improve_coding_matrix(k, m, w, matrix);
    return matrix;
```

src/cauchy_best_r6.c lines 241 to 300

```
  }
}

static int cbest_2[3] = { 1, 2, 3 };
static int cbest_3[7] = { 1, 2, 5, 4, 7, 3, 6 };

static int cbest_4[15] = { 1, 2, 9, 4, 8, 13, 3, 6, 12, 5, 11, 15, 10, 14, 7 };

static int cbest_5[31] = { 1, 2, 18, 4, 9, 8, 22, 16, 3, 11, 19, 5, 10, 6, 20, 27, 13, 23, 26, 12,
    17, 25, 24, 31, 30, 7, 15, 21, 29, 14, 28 };

static int cbest_6[63] = { 1, 2, 33, 4, 8, 49, 16, 32, 57, 3, 6, 12, 24, 48, 5, 35, 9, 37, 10, 17,
    41, 51, 56, 61, 18, 28, 53, 14, 20, 34, 7, 13, 25, 36, 59, 26, 39, 40, 45, 50, 60, 52, 63,
    11, 30, 55, 19, 22, 29, 43, 58, 15, 21, 38, 44, 47, 62, 27, 54, 42, 31, 23, 46 };

static int cbest_7[127] = { 1, 2, 68, 4, 34, 8, 17, 16, 76, 32, 38, 3, 64, 69, 5, 19, 35, 70, 6, 9,
    18, 102, 10, 36, 85, 12, 21, 42, 51, 72, 77, 84, 20, 25, 33, 50, 78, 98, 24, 39, 49, 100, 110
   , 48, 65, 93, 40, 66, 71, 92, 7, 46, 55, 87, 96, 103, 106, 11, 23, 37, 54, 81, 86, 108, 13,
    22, 27, 43, 53, 73, 80, 14, 26, 52, 74, 79, 99, 119, 44, 95, 101, 104, 111, 118, 29, 59, 89,
    94, 117, 28, 41, 58, 67, 88, 115, 116, 47, 57, 83, 97, 107, 114, 127, 56, 82, 109, 113, 126,
    112, 125, 15, 63, 75, 123, 124, 31, 45, 62, 91, 105, 122, 30, 61, 90, 121, 60, 120 };

static int cbest_8[255] = { 1, 2, 142, 4, 71, 8, 70, 173, 3, 35, 143, 16, 17, 67, 134, 140, 172, 6, 34
   , 69, 201, 216, 5, 33, 86, 12, 65, 138, 158, 159, 175, 10, 32, 43, 66, 108, 130, 193, 234, 9,
    24, 25, 50, 68, 79, 100, 132, 174, 200, 217, 20, 21, 42, 48, 87, 169, 41, 54, 64, 84, 96, 117
   , 154, 155, 165, 226, 77, 82, 135, 136, 141, 168, 192, 218, 238, 7, 18, 19, 39, 40, 78, 113,
    116, 128, 164, 180, 195, 205, 220, 232, 14, 26, 27, 58, 109, 156, 157, 203, 235, 13, 28, 29, 38
   , 51, 56, 75, 85, 90, 101, 110, 112, 139, 171, 11, 37, 49, 52, 76, 83, 102, 119, 131, 150, 151
   , 167, 182, 184, 188, 197, 219, 224, 45, 55, 80, 94, 97, 133, 170, 194, 204, 221, 227, 236, 36,
    47, 73, 92, 98, 104, 118, 152, 153, 166, 202, 207, 239, 251, 22, 23, 44, 74, 91, 148, 149, 161
   , 181, 190, 233, 46, 59, 88, 137, 146, 147, 163, 196, 208, 212, 222, 250, 57, 81, 95, 106, 111,
    129, 160, 176, 199, 243, 249, 15, 53, 72, 93, 103, 115, 125, 162, 183, 185, 189, 206, 225, 255,
    186, 210, 230, 237, 242, 248, 30, 31, 62, 89, 99, 105, 114, 121, 124, 178, 209, 213, 223, 228,
    241, 254, 60, 191, 198, 247, 120, 240, 107, 127, 144, 145, 177, 211, 214, 246, 245, 123, 126,
    187, 231, 253, 63, 179, 229, 244, 61, 122, 215, 252 };

static int cbest_9[511] = { 1, 2, 264, 4, 132, 8, 66, 16, 33, 32, 280, 64, 140, 128, 3, 70, 265, 5,
    133, 256, 266, 6, 9, 35, 67, 134, 268, 396, 10, 17, 34, 330, 12, 18, 68, 198, 297, 20, 37, 74
   , 136, 148, 165, 281, 296, 24, 36, 41, 65, 82, 99, 164, 272, 282, 388, 40, 49, 98, 141, 194,
    284, 328, 412, 48, 97, 129, 142, 196, 346, 71, 72, 96, 130, 313, 392, 80, 206, 257, 267, 312,
    334, 7, 135, 156, 173, 192, 258, 269, 397, 404, 11, 78, 144, 161, 172, 260, 270, 299, 331, 344,
    398, 13, 19, 39, 69, 86, 103, 160, 167, 199, 202, 298, 322, 384, 14, 21, 38, 43, 75, 102, 137,
    149, 166, 204, 289, 332, 408, 462, 22, 25, 42, 51, 83, 101, 138, 150, 273, 283, 288, 301, 350,
    389, 429, 26, 50, 76, 100, 195, 274, 285, 300, 329, 363, 390, 413, 428, 28, 45, 84, 143, 197,
    200, 214, 231, 276, 286, 315, 320, 347, 362, 414, 458, 44, 53, 73, 90, 107, 131, 152, 169, 181,
    230, 314, 338, 361, 393, 400, 454, 460, 52, 57, 81, 106, 115, 168, 175, 180, 207, 229, 305, 335
   , 348, 360, 394, 421, 478, 56, 105, 114, 157, 163, 174, 193, 210, 227, 228, 259, 304, 317, 326,
    405, 420, 445, 79, 104, 113, 145, 158, 162, 212, 226, 261, 271, 316, 345, 379, 399, 406, 444,
    450, 456, 87, 88, 112, 146, 203, 225, 262, 291, 323, 336, 378, 385, 425, 452, 474, 15, 205, 222
   , 224, 239, 290, 303, 333, 367, 377, 386, 409, 424, 431, 463, 470, 476, 23, 139, 151, 189, 208,
    238, 302, 324, 351, 366, 376, 410, 430, 437, 27, 47, 77, 94, 111, 177, 188, 237, 275, 293, 342,
    365, 391, 436, 448, 29, 46, 55, 85, 110, 119, 171, 176, 183, 201, 215, 218, 235, 236, 277, 287,
    292, 321, 355, 364, 415, 417, 459, 466, 472, 30, 54, 59, 91, 109, 118, 153, 170, 182, 220, 234,
    278, 307, 339, 354, 401, 416, 423, 441, 455, 461, 468, 495, 58, 108, 117, 154, 233, 306, 319,
    349, 353, 383, 395, 402, 422, 440, 447, 479, 494, 92, 116, 211, 232, 318, 327, 340, 352, 382,
    446, 493, 61, 159, 213, 216, 247, 309, 381, 407, 427, 451, 457, 464, 491, 492, 60, 89, 123, 147
   , 185, 246, 263, 308, 337, 371, 380, 426, 433, 453, 475, 487, 490, 122, 184, 191, 223, 245, 370,
    387, 432, 439, 471, 477, 486, 489, 511, 121, 179, 190, 209, 243, 244, 295, 325, 359, 369, 411,
    438, 485, 488, 510, 95, 120, 178, 242, 294, 343, 358, 368, 419, 449, 483, 484, 509, 219, 241,
    357, 418, 443, 467, 473, 482, 507, 508, 31, 221, 240, 255, 279, 356, 442, 469, 481, 503, 506,
```

src/cauchy_best_r6.c lines 301 to 360

```
    155, 254, 403, 480, 502, 505, 63, 93, 127, 253, 311, 341, 375, 501, 504, 62, 126, 187, 217, 251
   , 252, 310, 374, 435, 465, 499, 500, 125, 186, 250, 373, 434, 498, 124, 249, 372, 497, 248, 496
    };

static int cbest_10[1023] = { 1, 2, 516, 4, 258, 8, 129, 16, 32, 580, 64, 128, 290, 145, 256, 3, 512,
    517, 5, 259, 518, 588, 6, 9, 18, 36, 72, 144, 774, 10, 17, 131, 262, 288, 524, 645, 12, 33,
    133, 266, 294, 387, 532, 576, 581, 20, 34, 65, 137, 274, 548, 582, 24, 66, 291, 838, 40, 68,
    130, 147, 161, 322, 644, 709, 806, 48, 132, 193, 257, 386, 596, 80, 136, 298, 419, 612, 661, 772
   , 96, 149, 260, 272, 306, 403, 513, 146, 153, 160, 264, 292, 385, 514, 519, 544, 584, 589, 708,
    870, 7, 19, 37, 73, 192, 354, 590, 770, 775, 11, 38, 74, 177, 263, 289, 418, 520, 525, 534, 641
   , 660, 725, 802, 836, 846, 13, 22, 76, 148, 209, 267, 295, 320, 330, 402, 526, 528, 533, 577,
    647, 717, 804, 14, 21, 26, 35, 44, 135, 152, 165, 201, 275, 304, 384, 401, 435, 549, 578, 583,
    604, 608, 782, 903, 25, 52, 67, 88, 139, 270, 296, 391, 417, 550, 620, 653, 790, 834, 839, 41,
    50, 69, 104, 141, 176, 278, 302, 323, 395, 423, 540, 598, 640, 705, 724, 807, 866, 28, 42, 49,
    70, 82, 100, 163, 208, 282, 310, 556, 592, 597, 646, 663, 677, 711, 716, 868, 878, 81, 134, 151
   , 164, 195, 200, 299, 326, 352, 362, 400, 434, 564, 613, 657, 768, 773, 902, 967, 97, 138, 155,
    169, 197, 261, 273, 307, 358, 390, 416, 433, 451, 614, 652, 733, 800, 814, 844, 854, 935, 56, 84
   , 98, 140, 181, 217, 265, 293, 328, 338, 394, 422, 515, 545, 585, 704, 788, 822, 871, 919, 162,
    179, 276, 355, 407, 427, 546, 586, 591, 616, 662, 669, 676, 710, 727, 741, 771, 780, 901, 39, 75
   , 150, 157, 194, 211, 225, 268, 280, 308, 314, 389, 411, 439, 521, 530, 535, 628, 656, 721, 803,
    832, 837, 842, 847, 966, 23, 77, 112, 154, 168, 196, 300, 321, 331, 393, 421, 432, 450, 522, 527
   , 529, 552, 606, 643, 673, 693, 713, 732, 805, 864, 874, 934, 999, 15, 27, 45, 54, 78, 90, 108,
    180, 216, 305, 483, 560, 579, 600, 605, 609, 719, 778, 783, 852, 876, 886, 899, 918, 983, 46, 53
   , 89, 167, 178, 185, 203, 213, 271, 297, 324, 334, 336, 360, 370, 406, 426, 467, 542, 551, 610,
    621, 649, 668, 726, 740, 786, 791, 810, 820, 835, 900, 917, 931, 951, 965, 975, 30, 51, 105, 156
   , 205, 210, 224, 279, 303, 356, 366, 388, 405, 410, 438, 449, 459, 536, 541, 594, 599, 622, 655,
    720, 812, 818, 862, 867, 933, 29, 43, 71, 83, 92, 101, 106, 143, 173, 283, 311, 312, 346, 392,
    409, 420, 437, 443, 557, 566, 593, 642, 659, 672, 692, 707, 712, 737, 757, 869, 879, 911, 998,
    60, 102, 241, 327, 353, 363, 399, 425, 482, 558, 565, 624, 679, 718, 735, 749, 769, 798, 898,
    963, 982, 58, 86, 166, 183, 184, 202, 212, 219, 233, 286, 359, 431, 466, 615, 636, 648, 689, 729
   , 801, 815, 840, 845, 850, 855, 884, 916, 930, 950, 964, 974, 981, 995, 1015, 57, 85, 99, 120,
    171, 199, 204, 229, 318, 329, 339, 368, 404, 448, 458, 465, 499, 654, 671, 685, 784, 789, 823,
    872, 882, 915, 932, 949, 997, 1007, 116, 142, 159, 172, 277, 408, 436, 442, 455, 481, 491, 547,
    572, 587, 617, 630, 658, 665, 706, 723, 736, 756, 776, 781, 816, 860, 894, 897, 910, 947, 991,
    114, 221, 240, 269, 281, 309, 315, 332, 342, 344, 378, 398, 424, 441, 475, 487, 531, 618, 629,
    678, 695, 734, 743, 748, 808, 833, 843, 929, 943, 962, 973, 113, 182, 189, 218, 227, 232, 301,
    364, 374, 430, 457, 523, 553, 562, 602, 607, 688, 728, 753, 796, 830, 865, 875, 927, 980, 994,
    1014, 55, 79, 91, 109, 170, 187, 198, 215, 228, 284, 415, 464, 498, 554, 561, 601, 670, 675, 684
   , 715, 745, 765, 779, 848, 853, 877, 887, 909, 914, 948, 979, 996, 1006, 1013, 47, 110, 158, 249
   , 316, 325, 335, 337, 361, 371, 397, 447, 454, 480, 490, 497, 538, 543, 611, 632, 664, 722, 787,
    811, 821, 880, 896, 913, 946, 961, 971, 990, 1011, 31, 94, 220, 245, 357, 367, 429, 440, 474,
    486, 537, 595, 623, 651, 681, 694, 701, 742, 759, 813, 819, 858, 863, 892, 928, 942, 945, 972,
    989, 993, 1003, 1023, 62, 93, 107, 188, 207, 226, 237, 243, 313, 340, 347, 376, 456, 471, 473,
    507, 567, 568, 626, 752, 890, 907, 926, 1005, 61, 103, 124, 175, 186, 214, 372, 414, 453, 463,
    489, 503, 559, 625, 638, 674, 691, 714, 731, 739, 744, 764, 794, 799, 828, 908, 925, 939, 959,
    978, 1012, 59, 87, 122, 248, 287, 350, 396, 413, 446, 485, 495, 496, 637, 751, 826, 841, 851,
    885, 912, 941, 960, 970, 977, 1010, 118, 121, 235, 244, 319, 369, 382, 428, 445, 574, 650, 667,
    680, 700, 758, 761, 785, 873, 883, 944, 988, 992, 1002, 1009, 1022, 117, 206, 223, 231, 236, 242
   , 470, 472, 506, 573, 631, 687, 777, 817, 856, 861, 895, 906, 987, 1004, 1021, 115, 174, 191, 333
   , 343, 345, 379, 452, 462, 469, 488, 502, 505, 619, 690, 697, 730, 738, 755, 809, 888, 924, 938,
    958, 969, 1019, 253, 365, 375, 412, 484, 494, 501, 563, 603, 750, 767, 792, 797, 831, 923, 940,
    957, 976, 1001, 234, 251, 285, 348, 444, 479, 555, 634, 666, 760, 824, 849, 905, 955, 1008, 111,
    222, 230, 247, 317, 380, 461, 511, 539, 633, 686, 703, 747, 881, 937, 986, 1020, 95, 190, 468,
    493, 504, 570, 696, 754, 859, 893, 968, 985, 1018, 63, 126, 252, 341, 377, 500, 569, 627, 683,
    766, 891, 922, 956, 1000, 1017, 125, 239, 250, 373, 478, 639, 795, 829, 904, 921, 954, 123, 246,
    351, 460, 477, 510, 702, 746, 763, 827, 936, 953, 119, 383, 492, 509, 575, 984, 682, 699, 857,
    1016, 238, 255, 889, 920, 476, 762, 793, 952, 349, 508, 635, 825, 381, 698, 254, 571, 127 };

static int cbest_11[1023] = { 1,
    2, 1026, 4, 513, 8, 16, 1282, 32, 64, 641, 128, 256, 512, 1346, 1024, 3, 673, 1027, 5, 10, 20, 40, 80, 160, 320,
```

```
    640, 6, 9, 515, 1030, 1280, 1539, 17, 517, 1034, 1283, 12, 18, 33, 521, 1042, 1362, 34, 65, 529, 1058, 1286, 1795,
    24, 36, 66, 129, 545, 643, 1090, 1290, 1667, 68, 130, 257, 577, 645, 672, 1154, 1298, 1344, 48, 72, 132, 258, 336,
    649, 681, 1314, 1347, 136, 168, 260, 514, 657, 769, 1538, 1923, 84, 96, 144, 264, 516, 1025, 1350, 1410, 1859, 42,
    272, 520, 705, 1032, 1354, 11, 21, 41, 81, 161, 192, 288, 321, 528, 675, 1028, 1537, 1699, 1794, 7, 22, 82, 162,
    322, 544, 642, 677, 897, 1031, 1046, 1066, 1106, 1186, 1281, 1366, 1378, 1666, 14, 44, 164, 324, 384, 523, 533,
    553, 576, 593, 644, 833, 1035, 1040, 1288, 1360, 1987, 13, 19, 28, 88, 328, 519, 648, 680, 689, 1043, 1056, 1284,
    1363, 1474, 1543, 1793, 1955, 26, 35, 56, 176, 656, 768, 1038, 1059, 1088, 1287, 1302, 1322, 1442, 1547, 1665,
    1922, 25, 37, 52, 67, 112, 340, 352, 525, 531, 737, 1091, 1152, 1291, 1296, 1555, 1858, 1875, 38, 69, 74, 104, 131,
    224, 547, 651, 661, 683, 704, 721, 961, 1050, 1062, 1155, 1299, 1312, 1345, 1370, 1571, 1799, 49, 70, 73, 133, 138,
    148, 170, 208, 259, 337, 448, 537, 549, 579, 647, 674, 929, 1094, 1294, 1315, 1352, 1536, 1603, 1671, 1698, 1803,
    1921, 50, 134, 137, 169, 261, 266, 276, 296, 338, 416, 581, 676, 896, 1074, 1098, 1158, 1348, 1394, 1408, 1675,
    1707, 1811, 1857, 2019, 76, 85, 97, 145, 262, 265, 522, 532, 552, 561, 585, 592, 653, 659, 685, 771, 832, 849,
    1064, 1162, 1194, 1306, 1318, 1351, 1386, 1411, 1506, 1683, 1827, 1986, 2003, 43, 86, 98, 140, 146, 172, 273, 344,
    518, 688, 773, 1033, 1110, 1122, 1170, 1355, 1490, 1542, 1697, 1792, 1927, 1954, 100, 193, 268, 274, 289, 597, 609,
    665, 697, 707, 777, 1029, 1044, 1104, 1184, 1330, 1364, 1376, 1414, 1546, 1664, 1731, 1863, 1931, 1963, 23, 46, 83,
    92, 152, 163, 184, 194, 290, 323, 368, 524, 530, 555, 693, 709, 736, 753, 785, 993, 1036, 1047, 1067, 1107, 1187,
    1218, 1320, 1358, 1367, 1379, 1418, 1450, 1545, 1554, 1867, 1874, 1939, 1985, 15, 30, 45, 60, 90, 120, 165, 180,
    196, 240, 280, 292, 325, 330, 360, 385, 480, 546, 650, 660, 679, 682, 713, 720, 745, 801, 899, 960, 977, 1041,
    1289, 1361, 1426, 1472, 1541, 1570, 1703, 1798, 1953, 29, 58, 89, 116, 166, 200, 232, 326, 329, 386, 464, 535, 536,
    548, 578, 595, 646, 835, 901, 928, 1048, 1057, 1070, 1190, 1285, 1300, 1368, 1382, 1440, 1475, 1559, 1579, 1602,
    1619, 1670, 1802, 1879, 1891, 1920, 27, 57, 177, 304, 388, 527, 557, 580, 691, 725, 837, 905, 937, 1039, 1054,
    1089, 1114, 1292, 1303, 1323, 1374, 1443, 1553, 1674, 1706, 1715, 1801, 1810, 1856, 1873, 1991, 2018, 2035, 53,
    106, 113, 178, 212, 332, 341, 353, 392, 424, 541, 560, 584, 601, 652, 658, 684, 770, 841, 848, 913, 1060, 1082,
    1096, 1153, 1202, 1297, 1402, 1478, 1522, 1569, 1673, 1682, 1705, 1797, 1826, 1959, 1995, 2002, 2027, 39, 54, 75,
    105, 114, 225, 342, 354, 400, 539, 569, 739, 772, 1051, 1063, 1078, 1092, 1138, 1160, 1192, 1304, 1313, 1326, 1371,
    1384, 1398, 1446, 1482, 1514, 1551, 1601, 1669, 1696, 1763, 1815, 1835, 1926, 71, 139, 149, 171, 209, 226, 298,
    356, 449, 565, 596, 608, 625, 663, 664, 696, 706, 723, 741, 776, 853, 865, 963, 1072, 1095, 1130, 1156, 1250, 1295,
    1310, 1353, 1392, 1687, 1730, 1747, 1809, 1862, 1930, 1962, 1971, 2007, 2017, 51, 78, 108, 135, 150, 210, 228, 267,
    277, 297, 339, 348, 417, 450, 551, 554, 587, 617, 655, 687, 692, 708, 752, 784, 931, 965, 992, 1009, 1075, 1099,
    1159, 1174, 1234, 1316, 1338, 1349, 1395, 1409, 1458, 1494, 1504, 1544, 1563, 1575, 1681, 1825, 1866, 1883, 1929,
    1938, 1961, 1984, 2001, 77, 142, 174, 263, 278, 346, 376, 418, 452, 496, 583, 669, 678, 701, 712, 729, 744, 761,
    800, 898, 933, 969, 976, 1001, 1065, 1108, 1120, 1163, 1168, 1195, 1307, 1319, 1334, 1356, 1387, 1416, 1448, 1488,
    1507, 1540, 1607, 1702, 1807, 1865, 1925, 1952, 87, 99, 141, 147, 156, 173, 188, 216, 248, 270, 300, 345, 372, 420,
    456, 488, 534, 563, 594, 667, 699, 757, 779, 789, 809, 834, 851, 900, 1102, 1111, 1123, 1171, 1328, 1412, 1491,
    1558, 1578, 1587, 1611, 1618, 1679, 1711, 1729, 1861, 1878, 1890, 1907, 1943, 2023, 94, 101, 124, 154, 186, 244,
    269, 275, 284, 526, 556, 589, 690, 724, 775, 836, 904, 936, 945, 981, 1045, 1068, 1105, 1166, 1185, 1198, 1216,
    1331, 1365, 1377, 1390, 1415, 1430, 1510, 1552, 1577, 1714, 1800, 1819, 1831, 1872, 1899, 1937, 1990, 2034, 47, 62,
    93, 102, 122, 153, 185, 195, 282, 291, 312, 362, 369, 432, 468, 540, 599, 600, 611, 715, 747, 840, 857, 912, 1037,
    1052, 1112, 1126, 1219, 1321, 1359, 1372, 1419, 1424, 1451, 1568, 1623, 1635, 1672, 1691, 1701, 1704, 1723, 1796,
    1958, 1994, 2011, 2026, 2043, 31, 61, 91, 121, 181, 197, 202, 234, 241, 281, 293, 308, 331, 361, 370, 481, 538,
    568, 613, 695, 711, 738, 755, 781, 787, 995, 1080, 1118, 1178, 1188, 1210, 1380, 1400, 1427, 1473, 1498, 1530,
    1550, 1557, 1600, 1617, 1668, 1719, 1735, 1762, 1779, 1814, 1834, 1843, 1877, 1889, 1935, 1967, 1993, 2025, 2039,
    59, 117, 167, 182, 198, 201, 233, 242, 294, 327, 387, 465, 482, 559, 564, 605, 624, 662, 722, 740, 803, 852, 864,
    881, 907, 917, 939, 962, 979, 997, 1049, 1071, 1086, 1146, 1191, 1206, 1222, 1266, 1301, 1324, 1369, 1383, 1406,
    1422, 1441, 1454, 1480, 1512, 1526, 1549, 1686, 1713, 1739, 1746, 1771, 1808, 1833, 1871, 1970, 1989, 2006, 2016,
    2033, 118, 305, 334, 364, 389, 394, 404, 426, 466, 484, 543, 550, 573, 586, 603, 616, 633, 654, 686, 717, 749, 793,
    805, 843, 873, 903, 930, 964, 1008, 1055, 1115, 1128, 1142, 1200, 1226, 1258, 1293, 1308, 1375, 1476, 1520, 1562,
    1574, 1680, 1824 };

static int cbest_12[1023] = {
    1, 2, 2089, 4, 8, 3133, 16, 2088, 1044, 3, 32, 522, 261, 3639, 5, 64, 65, 2057, 2091, 6, 130, 3132, 260,
    2219, 9, 2093, 3117, 10, 128, 257, 2081, 3890, 12, 1566, 2217, 3129, 17, 514, 520, 3135, 18, 3196, 3637, 20,
    256, 1028, 1045, 3638, 24, 33, 523, 783, 2105, 3197, 3647, 34, 1040, 1945, 2056, 2090, 2595, 3125, 3891, 36,
    1046, 2348, 40, 1598, 2218, 3635, 7, 48, 66, 67, 131, 263, 269, 512, 538, 782, 1076, 1174, 1819, 2049, 2059,
    2092, 2152, 3113, 3116, 3384, 3607, 3888, 68, 69, 277, 554, 1108, 1109, 1558, 1564, 1944, 2061, 2080, 2478,
    3894, 72, 73, 134, 265, 293, 391, 526, 586, 587, 779, 2153, 2211, 2216, 2563, 3128, 4016, 11, 14, 80, 81,
    129, 132, 138, 530, 1172, 1692, 2073, 2095, 2223, 2316, 3101, 3109, 3119, 3134, 3645, 3874, 13, 96, 97, 146,
    259, 262, 268, 289, 799, 1052, 1060, 1158, 1567, 1818, 2083, 2221, 2344, 2349, 2476, 2980, 3045, 3192, 3368,
    3636, 144, 162, 276, 389, 515, 521, 578, 579, 781, 972, 1024, 1156, 2085, 2120, 2235, 2282, 2312, 3131, 3623,
    19, 22, 28, 160, 194, 195, 264, 292, 390, 778, 846, 1238, 1239, 1562, 1596, 1684, 1937, 1947, 2104, 2233,
```

src/cauchy_best_r6.c lines 421 to 480

```
    2593, 3193, 3385, 3633, 3643, 3646, 3889, 21, 26, 324, 518, 536, 909, 1029, 1297, 2008, 2121, 2209, 2283,
    2479, 2594, 2854, 3085, 3121, 3124, 3188, 3198, 3605, 3895, 25, 136, 258, 288, 528, 552, 798, 842, 1030,
    1140, 1141, 1490, 1556, 2470, 2603, 2626, 2721, 3892, 3898, 4017, 35, 38, 44, 56, 320, 388, 423, 486, 524,
    570, 584, 585, 618, 619, 648, 777, 780, 1041, 1594, 1817, 2097, 2107, 2317, 2561, 3105, 3164, 3189, 3199,
    3263, 3634, 3875, 37, 42, 52, 285, 309, 421, 1036, 1042, 1047, 1056, 1072, 1170, 1313, 1427, 1823, 1936,
    1946, 2048, 2053, 2058, 2065, 2109, 2280, 2345, 2477, 2579, 2627, 2981, 3041, 3044, 3112, 3127, 3369, 3388,
    3603, 3606, 3631, 3765, 3872, 41, 192, 193, 273, 399, 775, 797, 908, 968, 1104, 1105, 1168, 1236, 1237, 1281,
    1296, 1554, 1582, 1599, 1949, 2051, 2060, 2313, 2332, 2340, 2474, 3115, 3165, 3336, 3360, 3641, 4018, 49, 70,
    71, 76, 77, 88, 89, 112, 113, 387, 454, 484, 513, 516, 539, 640, 744, 745, 1004, 1077, 1175, 1234, 1235,
    1550, 1668, 1680, 1694, 1803, 2063, 2112, 2210, 2215, 2227, 2281, 2472, 2562, 2784, 3261, 3449, 3547, 3591,
    3621, 84, 85, 104, 105, 142, 154, 178, 226, 227, 242, 243, 267, 271, 385, 422, 534, 555, 771, 776, 791, 834,
    840, 844, 973, 1068, 1078, 1166, 1425, 1488, 1559, 1560, 1565, 1688, 1816, 2072, 2077, 2094, 2136, 2144,
    2154, 2222, 2298, 2336, 2350, 2468, 2543, 2611, 2624, 2658, 2729, 2850, 2855, 2982, 3081, 3097, 3100, 3108,
    3118, 3184, 3247, 3326, 3376, 3644, 3702, 4024, 82, 83, 135, 164, 170, 210, 211, 279, 284, 308, 328, 356,
    372, 417, 420, 527, 542, 576, 577, 616, 617, 656, 795, 847, 1048, 1110, 1111, 1312, 1329, 1392, 1426, 1590,
    1592, 1822, 2082, 2113, 2128, 2156, 2187, 2220, 2364, 2466, 2471, 2785, 2976, 3077, 3111, 3180, 3245, 3882,
    3893, 3899, 3902, 4081, 15, 30, 50, 60, 98, 99, 120, 121, 133, 139, 166, 272, 291, 295, 325, 340, 398, 419,
    531, 558, 582, 583, 712, 713, 774, 796, 838, 1064, 1084, 1173, 1182, 1232, 1233, 1280, 1522, 1542, 1676,
    1693, 1811, 1948, 2009, 2084, 2137, 2145, 2155, 2185, 2213, 2225, 2234, 2286, 2299, 2599, 2601, 2625, 2659,
    2745, 2870, 2988, 3047, 3130, 3185, 3194, 3327, 3352, 3386, 3545, 3601, 3619, 3622, 3629, 3703, 3733, 3773,
    3896, 3955, 140, 147, 152, 176, 186, 198, 199, 224, 225, 281, 305, 332, 386, 397, 532, 546, 590, 591, 680,
    696, 843, 901, 911, 974, 1026, 1032, 1053, 1061, 1116, 1117, 1159, 1164, 1491, 1552, 1802, 1941, 2069, 2087,
    2122, 2129, 2157, 2168, 2232, 2296, 2314, 2318, 2446, 2567, 2592, 2737, 2852, 3040, 3073, 3093, 3103, 3156,
    3181, 3389, 3627, 3632, 3642, 3858, 3873, 4020, 74, 75, 145, 163, 168, 208, 209, 266, 270, 321, 384, 395,
    452, 487, 502, 664, 770, 790, 905, 1025, 1054, 1148, 1149, 1152, 1157, 1301, 1360, 1424, 1578, 1686, 1821,
    1851, 1929, 1939, 2012, 2208, 2287, 2328, 2333, 2341, 2346, 2381, 2397, 2444, 2464, 2475, 2723, 2752, 3084,
    3120, 3195, 3337, 3361, 3370, 3417, 3514, 3579, 3604, 3615, 3780, 3878, 3984, 4019, 23, 29, 46, 92, 93, 161,
    240, 241, 278, 348, 416, 450, 562, 568, 610, 611, 650, 769, 773, 789, 794, 1154, 1222, 1223, 1328, 1376,
    1494, 1520, 1563, 1597, 1685, 1951, 2010, 2075, 2123, 2169, 2284, 2297, 2473, 2511, 2541, 2602, 2619, 2666,
    2720, 3107, 3123, 3157, 3294, 3324, 3372, 3448, 3453, 3546, 3587, 3670, 3710, 27, 196, 197, 290, 294, 418,
    482, 519, 537, 540, 574, 688, 787, 793, 969, 1062, 1142, 1143, 1435, 1548, 1588, 1662, 1663, 1801, 1810,
    1992, 2055, 2096, 2101, 2106, 2337, 2351, 2469, 2539, 2542, 2560, 2571, 2577, 2634, 2753, 2851, 2919, 2983,
    3013, 3043, 3104, 3259, 3262, 3322, 3377, 3380, 3441, 3625, 3717, 3767, 3781, 3880, 3900, 3939, 4025, 4049,
    137, 174, 280, 304, 322, 326, 344, 352, 393, 396, 455, 480, 485, 529, 553, 556, 760, 761, 832, 900, 910, 964,
    970, 1005, 1031, 1124, 1125, 1180, 1289, 1317, 1333, 1540, 1557, 1574, 1580, 1586, 1690, 1724, 1921, 1940,
    2052, 2064, 2099, 2108, 2250, 2266, 2285, 2304, 2365, 2467, 2578, 2583, 2609, 2656, 2667, 2848, 2977, 2996,
    3126, 3166, 3176, 3215, 3239, 3257, 3295, 3325, 3512, 3602, 3617, 3630, 3671, 3700, 3711, 3761, 3764, 3842,
    3883, 3903, 4080, 39, 45, 57, 148, 172, 184, 287, 297, 336, 394, 448, 525, 544, 571, 588, 589, 649, 746, 747,
    835, 841, 845, 904, 996, 1006, 1038, 1074, 1080, 1092, 1093, 1178, 1220, 1221, 1300, 1489, 1492, 1538, 1595,
    1670, 1682, 1756, 1757, 1795, 1815, 1820, 1835, 1850, 1928, 1938, 2050, 2111, 2179, 2356, 2440, 2509, 2565,
    2587, 2635, 2871, 2915, 2978, 2984, 2989, 3046, 3053, 3114, 3148, 3172, 3207, 3292, 3323, 3340, 3353, 3356,
    3364, 3387, 3401, 3445, 3544, 3589, 3640, 3729, 3757, 3769, 3796, 3897, 3954, 3959, 251, 373, 581, 1057,
    1107, 1137, 1299, 1630, 1882, 2214, 2308, 2315, 2319, 2597, 2853, 3167, 3260, 3482, 3590, 3862 };

static int cbest_13[1023] = {
    1, 2, 4109, 4, 6155, 8, 7176, 16, 3588, 4108, 32, 2054, 3, 1027, 1794, 6, 64, 4620, 6154, 5, 897, 4111, 6153,
    7177, 12, 2310, 3077, 4105, 128, 9, 24, 513, 1155, 10, 1026, 4101, 7689, 18, 48, 256, 896, 3589, 4365, 6159,
    7178, 17, 448, 4557, 4621, 4684, 5647, 36, 96, 224, 1025, 1792, 20, 112, 512, 769, 1538, 1795, 2052, 3076,
    3590, 6147, 6283, 7180, 33, 56, 72, 192, 2055, 2342, 3584, 4125, 6411, 7945, 28, 34, 577, 1154, 6922, 7240,
    14, 144, 384, 2050, 4110, 4493, 5134, 5903, 6152, 7168, 7304, 7, 40, 65, 1171, 3620, 4104, 4141, 5004, 288,
    768, 1153, 2308, 2567, 3652, 4097, 4107, 4397, 4685, 6157, 6171, 6379, 7688, 13, 68, 1024, 1810, 2311, 3461,
    4676, 7179, 66, 129, 576, 1826, 3079, 3844, 4100, 4173, 4622, 5646, 6347, 8073, 25, 80, 899, 905, 1031, 1091,
    1798, 2182, 2306, 2338, 2502, 3073, 3596, 4364, 4616, 5390, 5645, 6158, 6187, 7050, 7192, 11, 26, 913, 1922,
    4117, 4556, 4652, 5262, 6151, 6299, 6443, 6923, 7181, 19, 49, 130, 136, 257, 1169, 1283, 1536, 1793, 2062,
    2326, 2566, 2823, 4103, 4237, 6145, 7182, 7232, 7272, 7691, 7944, 50, 449, 515, 901, 961, 1035, 1163, 1170,
    1539, 1802, 2048, 2340, 2695, 3586, 3591, 3604, 3616, 4553, 4612, 4680, 4748, 5005, 6146, 6219, 6282, 7208,
    7241, 7288, 22, 37, 52, 97, 160, 225, 585, 1152, 1251, 1808, 2374, 2631, 3141, 3585, 4124, 4549, 4876, 5135,
    5902, 6281, 6410, 7169, 7172, 7305, 7320, 7560, 21, 38, 98, 113, 132, 226, 258, 452, 545, 898, 904, 1187,
    2053, 2070, 2278, 2438, 3205, 3460, 3525, 3636, 3972, 4127, 4137, 4157, 4367, 4677, 5132, 5839, 5901, 6163,
    6169, 6409, 6666, 7693, 57, 73, 100, 114, 193, 228, 272, 450, 456, 517, 912, 1029, 1043, 1159, 1219, 1796,
```

```
    2343, 3078, 3333, 4113, 4121, 4361, 4381, 4393, 4492, 4559, 4623, 4686, 5518, 6275, 6920, 7721, 8137, 29, 35,
    44, 74, 104, 146, 641, 1030, 1090, 1730, 1824, 2246, 2318, 2822, 3072, 3085, 3592, 3621, 3660, 3780, 4099,
    4140, 4149, 4165, 4205, 4333, 4617, 4700, 5454, 5643, 6149, 6377, 7051, 7170, 7244, 7753, 7947, 15, 30, 42,
    60, 76, 120, 145, 196, 240, 260, 292, 320, 385, 480, 514, 771, 900, 960, 1123, 1411, 1818, 1986, 2051, 2086,
    2304, 2694, 2951, 3622, 3648, 3653, 4096, 4106, 4221, 4396, 4429, 4589, 4636, 4653, 4716, 5422, 5679, 6156,
    6170, 6378, 6603, 7184, 7242, 7681, 8072, 41, 58, 194, 200, 521, 581, 584, 785, 865, 1059, 1139, 1168, 1282,
    1347, 1811, 2058, 2198, 2350, 2565, 2630, 3644, 4301, 4357, 4485, 4732, 5391, 5644, 5711, 6191, 6203, 6287,
    6297, 7193, 7296, 7306, 70, 88, 148, 208, 289, 544, 579, 801, 1034, 1089, 1099, 1162, 1315, 1570, 1827, 1830,
    1890, 2309, 2324, 2336, 2358, 2759, 3075, 3093, 3845, 4133, 4143, 4172, 4613, 4648, 4668, 4681, 4749, 4812,
    4996, 5263, 5278, 5388, 5639, 5775, 6167, 6175, 6185, 6315, 6346, 6383, 6415, 6427, 6447, 7194, 7949, 7961,
    69, 84, 116, 152, 264, 290, 386, 392, 516, 593, 773, 833, 909, 993, 1033, 1161, 1175, 1250, 1542, 1602, 1799,
    1800, 1920, 2118, 2498, 2727, 3140, 3597, 3654, 4253, 4389, 4399, 4495, 4672, 4780, 4877, 5260, 6186, 6195,
    6211, 6251, 6345, 6441, 6794, 6926, 6938, 7018, 7183, 7233, 7256, 7273, 7624, 7690, 7977, 67, 82, 400, 529,
    609, 640, 929, 1041, 1157, 1179, 1186, 1249, 1281, 1666, 1858, 1923, 2060, 2180, 2334, 2406, 2500, 2563,
    2711, 3173, 3204, 3463, 3524, 3598, 3628, 3716, 4036, 4116, 4169, 4489, 4509, 4608, 4618, 4660, 4678, 4764,
    5006, 5130, 5899, 5919, 6150, 6267, 6298, 6351, 6442, 6475, 6667, 6730, 6954, 7048, 7174, 7209, 7212, 7224,
    7289, 7308, 7400, 7432, 81, 140, 176, 232, 296, 416, 625, 770, 915, 945, 1028, 1042, 1158, 1185, 1218, 1537,
    1814, 2068, 2183, 2307, 2322, 2339, 2390, 2503, 2950, 3109, 3189, 3332, 3397, 3457, 3606, 3612, 3846, 4102,
    4236, 4525, 4687, 5000, 5020, 5582, 5641, 5871, 5935, 6144, 6339, 6403, 6539, 6921, 6986, 7173, 7188, 7196,
    7321, 7336, 7561, 7705, 7937, 8065, 8075, 27, 54, 108, 168, 216, 304, 388, 432, 520, 561, 580, 705, 777, 784,
    864, 907, 1039, 1167, 1203, 1546, 1803, 1806, 2314, 2366, 2382, 2564, 2639, 2821, 3081, 3084, 3149, 3221,
    3365, 3469, 3587, 3600, 3605, 3617, 3668, 3840, 4373, 4413, 4541, 4552, 4701, 5133, 5198, 5838, 5900, 6179,
    6218, 6303, 6395, 6914, 7042, 7216, 7234, 7274, 7322, 7368, 7692, 8169, 131, 137, 164, 464, 578, 673, 800,
    903, 1095, 1122, 1195, 1410, 1475, 1762, 1809, 1822, 1834, 2056, 2063, 2078, 2082, 2178, 2276, 2327, 2330,
    2370, 2599, 2627, 2693, 2758, 3137, 3477, 3594, 3618, 3684, 3812, 4119, 4175, 4548, 4555, 4573, 4637, 4644,
    4654, 4682, 4692, 4717, 5068, 5126, 5254, 5446, 5486, 5519, 5663, 5837, 5895, 6031, 6161, 6217, 6279, 6280,
    6291, 6435, 7210, 7236, 7276, 7290, 7695, 7720, 7725, 7737, 7745, 8136, 51, 138, 280, 352, 549, 592, 657,
    772, 832, 908, 917, 992, 1051, 1058, 1121, 1138, 1191, 1217, 1346, 1816, 1842, 2018, 2049, 2066, 2190, 2274,
    2341, 2346, 2372, 2510, 2629, 2726, 3493, 3521, 3608, 3632, 3637, 3700, 3860, 3973, 4126, 4136, 4156, 4189,
    4229, 4366, 4405, 4421, 4461, 4628, 4696, 4733, 5276, 5420, 5455, 5470, 5516, 5642, 5677, 6162, 6168, 6223,
    6273, 6363, 6408, 7171, 7245, 7264, 7752, 7946, 8077, 8089, 23, 53, 134, 161, 336, 454, 496, 528, 608, 921,
    928, 1057, 1088, 1098, 1107, 1137, 1173, 1314, 1379, 1540, 1554, 1728, 1797, 1930, 2102, 2196, 2316, 2348,
    2375, 2436, 2562, 2710, 2819, 3074, 3092, 3509, 3638, 3876, 4112, 4120, 4360, 4380, 4392, 4445, 4545, 4558,
    4565, 4605, 4614, 4649, 4669, 4740, 4813, 4997, 5386, 5406, 5418, 5423, 5452, 5675, 5678, 5695, 5703, 6274,
    6285, 6371, 6664, 7054, 7066, 7185, 7200, 7243, 7246, 7280, 7324, 7680, 7685, 7729, 7881, 39, 78, 133, 156,
    162, 227, 274, 312, 328, 453, 519, 569, 624, 914, 965, 1032, 1037, 1075, 1097, 1165, 1313, 1363, 1409, 1825,
    1828, 1938, 2074, 2084, 2214, 2242, 2279, 2439, 2534, 2691, 3143, 3593, 3650, 3656, 3661, 3781, 4123, 4148,
    4153, 4239, 4353, 4363, 4481, 4551, 4673, 4781, 5128, 5258, 5358, 5637, 5710, 5807, 5835, 5897, 5917, 6173,
    6355, 6413, 6419, 6425, 6459, 6795, 6858, 6927, 7019, 7297, 7300, 7307, 7312, 7562, 7723, 7953 };

static int cbest_14[1023] = {
    1, 2, 8737, 4, 8, 13105, 16, 8736, 32, 4368, 3, 2184, 17, 1092, 15289, 34, 546, 5, 64, 68, 273, 8739, 8745,
    6, 136, 13104, 272, 9, 8741, 128, 8873, 12, 6552, 13109, 10, 544, 13107, 256, 18, 24, 3276, 8753, 33, 1088,
    4369, 15288, 16381, 2185, 8705, 13089, 13173, 20, 512, 1093, 2186, 4372, 8195, 8738, 8744, 13113, 35, 66,
    547, 1094, 1638, 2188, 4370, 4376, 8752, 8865, 12832, 15291, 36, 48, 65, 69, 257, 2176, 6416, 8707, 10921, 7,
    19, 25, 70, 137, 3208, 8740, 9008, 9829, 13088, 13169, 15281, 96, 132, 138, 277, 554, 1024, 1108, 1604, 6544,
    8749, 8872, 50, 140, 275, 281, 562, 802, 819, 3272, 4504, 7644, 8747, 8801, 8805, 12833, 13073, 13108, 21,
    100, 129, 192, 276, 401, 1636, 4352, 4436, 8743, 12563, 13106, 15257, 15293, 13, 38, 40, 72, 130, 200, 264,
    274, 280, 514, 550, 818, 2216, 2252, 6553, 8193, 8875, 9009, 15259, 11, 14, 49, 384, 400, 545, 1124, 2048,
    2218, 6554, 8761, 8877, 13075, 13111, 15273, 409, 1100, 1126, 14197, 15153, 15839, 42, 528, 768, 1109, 4097,
    4432, 8937, 9144, 15016, 16377, 16380, 51, 76, 98, 102, 204, 408, 552, 563, 803, 1090, 2200, 2248, 3277,
    8203, 8704, 10785, 13172, 144, 260, 548, 560, 800, 817, 1028, 1089, 1536, 3822, 4096, 4353, 8194, 8760,
    13112, 13117, 13241, 15272, 16365, 28, 258, 265, 1606, 1634, 4380, 6556, 8864, 8869, 8993, 9136, 10913,
    12561, 12836, 15290, 26, 80, 84, 196, 385, 513, 816, 1056, 2180, 2187, 2190, 3072, 3268, 3278, 4373, 4400,
    4496, 6145, 7508, 8706, 8755, 8941, 9016, 9145, 9287, 10920, 12013, 12290, 13175, 15017, 22, 67, 152, 1095,
    1639, 2189, 3212, 3292, 4371, 4374, 4377, 4568, 4572, 6418, 6536, 6584, 8225, 8929, 9825, 9828, 12849, 13093,
    13141, 13168, 13233, 14129, 14747, 15280, 16373, 16383, 37, 529, 570, 769, 1104, 1600, 1632, 2177, 5460,
    6144, 6417, 6586, 7636, 8466, 8713, 8748, 8867, 13091, 13097, 13171, 15837, 16313, 16364, 71, 134, 168, 285,
    392, 520, 530, 770, 1026, 1058, 1096, 2056, 2178, 2201, 2284, 3209, 4233, 4360, 4402, 4508, 4914, 6424, 7640,
    8709, 8721, 8746, 8757, 8800, 8804, 8992, 9121, 9137, 10923, 11809, 12837, 13072, 13115, 13137, 14193, 15566,
```

src/cauchy_best_r6.c lines 541 to 600

```
    97, 133, 139, 142, 145, 261, 288, 305, 337, 555, 561, 785, 801, 1025, 1120, 1140, 1605, 1646, 2112, 2116,
    2730, 3210, 3264, 3754, 4378, 6420, 6545, 8201, 8211, 8451, 8742, 8754, 8803, 8809, 8841, 8889, 9001, 9017,
    9831, 10925, 12291, 12562, 12834, 12840, 15161, 15256, 15283, 15292, 29, 56, 141, 259, 268, 279, 284, 674,
    1110, 1142, 1538, 1570, 1911, 2280, 2457, 3273, 3820, 4232, 4505, 6281, 6546, 7628, 7645, 8192, 8715, 8751,
    8769, 8874, 9012, 9285, 9317, 12848, 12960, 13057, 13077, 13092, 14880, 15258, 15265, 15285, 15295, 27, 52,
    74, 81, 85, 101, 160, 193, 304, 336, 516, 558, 784, 1060, 1348, 1365, 1540, 1637, 2052, 2254, 2286, 3140,
    3200, 3274, 3818, 4225, 4437, 4506, 4560, 6480, 6528, 6548, 7440, 8190, 8199, 8467, 8723, 8807, 8876, 9010,
    9129, 9837, 11945, 12307, 12547, 13074, 13090, 13096, 13110, 13143, 13655, 14745, 14777, 15241, 15353, 15357,
    15831, 23, 39, 41, 44, 73, 131, 153, 194, 201, 278, 290, 307, 403, 515, 536, 551, 610, 614, 806, 823, 1116,
    1122, 1228, 1612, 1910, 2050, 2217, 2220, 2253, 2456, 2696, 3224, 3240, 3293, 3720, 4105, 4438, 6280, 6537,
    8331, 8720, 8756, 8773, 8879, 8881, 9120, 9761, 9845, 10849, 10853, 12288, 12593, 13157, 13297, 13617, 14196,
    15008, 15152, 15243, 15567, 15835, 15838, 16245, 15, 58, 112, 170, 202, 283, 341, 405, 571, 682, 810, 1040,
    1105, 1125, 1364, 1568, 1620, 1860, 1877, 2049, 2192, 2219, 2250, 3076, 4101, 4112, 4224, 4356, 4500, 5392,
    6448, 6555, 7504, 7646, 8210, 8227, 8450, 8711, 8813, 8888, 8936, 9000, 10889, 12009, 12567, 12835, 12841,
    12968, 13059, 13177, 13181, 13305, 13309, 14888, 15155, 16108, 16376, 104, 306, 386, 393, 402, 531, 568, 771,
    822, 930, 1030, 1101, 1127, 1602, 1644, 2120, 2728, 3080, 3288, 3752, 3814, 4104, 4354, 4361, 4434, 4440,
    4444, 6153, 7576, 8202, 8219, 8464, 8602, 8843, 8933, 9013, 9797, 10784, 10789, 10891, 11877, 12571, 12896,
    12900, 12961, 13056, 13237, 14097, 14181, 14474, 14881, 15249, 15264, 15275, 16357, 16369, 43, 116, 148, 224,
    266, 282, 289, 340, 388, 404, 411, 465, 522, 576, 580, 608, 672, 1072, 1102, 1141, 1220, 1876, 1909, 2208,
    2222, 2240, 2244, 2282, 3136, 4100, 4433, 4468, 4564, 4912, 5456, 6273, 6400, 6484, 7444, 7620, 7632, 8323,
    8722, 8765, 9011, 9128, 9799, 9965, 10787, 10989, 12306, 12546, 13079, 13116, 13139, 13240, 13245, 14199,
    15033, 15240, 15251, 15261, 15564, 15869, 16317, 16361, 54, 57, 77, 99, 103, 162, 205, 232, 269, 518, 553,
    556, 566, 787, 1062, 1091, 1111, 1143, 1542, 2060, 2210, 2232, 2249, 3788, 3816, 6152, 6450, 7629, 7783,
    8182, 8197, 8209, 8241, 8449, 8481, 8587, 8763, 8868, 8880, 8939, 9146, 9295, 9844, 10912, 10917, 11471,
    11813, 12005, 12289, 12560, 13153, 13156, 13301, 13653, 13685, 15009, 15018, 15157, 15242, 15562, 16379, 46,
    53, 78, 208, 320, 410, 448, 464, 549, 612, 772, 804, 821, 938, 955, 1029, 1349, 1537, 1642, 1908, 2202, 2234,
    3242, 3284, 3722, 3823, 4240, 4464, 6160, 6272, 6529, 6558, 6576, 7098, 7237, 8233, 8329, 8771, 8775, 8871,
    8905, 8940, 9020, 9283, 9286, 10793, 12012, 12294, 12844, 12857, 12969, 13058, 13174, 13587, 14099, 14521,
    14739, 14761, 14889, 15020, 15321, 15430, 15833, 16109, 16349, 82, 88, 146, 262, 291, 537, 578, 680, 776,
    786, 807, 1032, 1074, 1117, 1160, 1574, 1607, 1635, 1654, 1894, 1907, 2124, 2182, 2204, 2440, 3084, 3225,
    3308, 3810, 4099, 4120, 4301, 4381, 4488, 4643, 6147, 6557, 6578, 6616, 6620, 7782, 8188, 8207, 8218, 8224,
    8465, 8603, 8833, 8928, 9249, 9824, 11873, 12305, 12337, 12545, 12565, 12577, 12853, 12897, 12901, 12977,
    13119, 13140, 13232, 13243, 14128, 14133, 14180, 14475, 14746, 15121, 15217, 15263, 15274, 15771, 15829,
    16177, 16356, 16372, 16382, 30, 59, 113, 197, 206, 469, 811, 820, 827, 896, 928, 954, 1041, 1057, 1136, 1216,
    1344, 1361, 1572, 1621, 1861, 2080, 2144, 2181, 2193, 2321, 3073, 3204, 3214, 3269, 3279, 4113, 4357, 4384,
    4404, 4420, 4497, 4922, 7236, 7492, 7509, 8579, 8600, 8712, 8729, 8866, 9138, 9319, 9827, 9833, 10915, 10985,
    11267, 12595, 12631, 12838, 13101, 13170, 13382, 13619, 13651, 14131, 14472, 14504, 15032, 15325, 15565,
    15827, 16125, 16312 };

static int cbest_15[1023] = {
    1, 2, 16385, 4, 8, 24577, 16, 32, 64, 28673, 128, 256, 512, 1024, 30721, 2048, 4096, 8192, 16384, 31745, 3,
    6, 12, 24, 48, 96, 192, 384, 768, 1536, 3072, 6144, 12288, 24576, 5, 16387, 9, 16389, 10, 17, 16393, 24579,
    28672, 18, 33, 14336, 16401, 24581, 20, 34, 65, 7168, 16417, 24585, 32257, 36, 66, 129, 3584, 16449, 24593,
    28675, 40, 68, 130, 257, 1792, 16513, 24609, 28677, 30720, 72, 132, 258, 513, 896, 16641, 24641, 28681, 80,
    136, 260, 448, 514, 1025, 15360, 16897, 24705, 28689, 144, 224, 264, 516, 1026, 2049, 17409, 24833, 28705,
    30723, 112, 160, 272, 520, 1028, 2050, 4097, 7680, 18433, 25089, 28737, 30725, 56, 288, 528, 1032, 2052,
    4098, 8193, 20481, 25601, 28801, 30729, 31744, 32513, 28, 320, 544, 1040, 2056, 3840, 4100, 8194, 26625,
    28929, 30737, 14, 576, 1056, 2064, 4104, 8196, 16386, 29185, 30753, 7, 13, 25, 49, 97, 193, 385, 640, 769,
    1088, 1537, 1920, 2080, 3073, 4112, 6145, 8200, 12289, 15872, 16388, 29697, 30785, 31747, 26, 50, 98, 194,
    386, 770, 1152, 1538, 2112, 3074, 4128, 6146, 8208, 12290, 16391, 16392, 16397, 16409, 16433, 16481, 16577,
    16769, 17153, 17921, 19457, 22529, 24578, 30849, 31749, 52, 100, 196, 388, 772, 960, 1280, 1540, 2176, 3076,
    4160, 6148, 8224, 12292, 16400, 24580, 30977, 31753, 11, 104, 200, 392, 776, 1544, 2304, 3080, 4224, 6152,
    7936, 8256, 12296, 16416, 24583, 24584, 24589, 24601, 24625, 24673, 24769, 24961, 25345, 26113, 27649, 31233,
    31761, 32256, 19, 22, 208, 400, 480, 784, 1552, 2560, 3088, 4352, 6160, 8320, 12304, 14337, 16395, 16448,
    24592, 28674, 31777, 21, 35, 38, 44, 416, 800, 1568, 3104, 4608, 6176, 7169, 8448, 12320, 14338, 16403,
    16512, 24608, 28676, 31809, 32641, 37, 67, 70, 76, 88, 240, 832, 1600, 3136, 3585, 3968, 5120, 6208, 7170,
    8704, 12352, 14340, 16405, 16419, 16640, 23553, 24587, 24640, 28679, 28680, 28685, 28697, 28721, 28769,
    28865, 29057, 29441, 30209, 31873, 32259, 41, 69, 131, 134, 140, 152, 176, 1664, 1793, 3200, 3586, 6272,
    7172, 9216, 12416, 14344, 16128, 16421, 16451, 16896, 19969, 24595, 24704, 28688, 32001, 32261, 42, 73, 120,
    133, 259, 262, 268, 280, 304, 352, 897, 1794, 3328, 3588, 6400, 7176, 10240, 12544, 14352, 16425, 16453,
```

```
    16515, 17408, 18177, 24597, 24611, 24832, 28704, 30722, 32265, 74, 81, 137, 261, 449, 515, 518, 524, 536,
    560, 608, 704, 898, 1796, 1984, 3592, 6656, 7184, 12800, 14368, 15361, 16457, 16517, 16643, 17281, 18432,
    24613, 24643, 25088, 28161, 28683, 28736, 30724, 32273, 60, 82, 138, 145, 225, 265, 450, 517, 900, 1027,
    1030, 1036, 1048, 1072, 1120, 1216, 1408, 1800, 3600, 7200, 13312, 14400, 15362, 16465, 16521, 16645, 16833,
    16899, 20480, 24617, 24645, 24707, 25600, 26369, 28691, 28800, 30727, 30728, 30733, 30745, 30769, 30817,
    30913, 31105, 31489, 32289, 32512, 84, 113, 146, 161, 226, 266, 273, 452, 521, 904, 1029, 1808, 2051, 2054,
    2060, 2072, 2096, 2144, 2240, 2432, 2816, 3616, 7232, 7681, 8064, 14464, 15364, 16529, 16609, 16649, 16901,
    17411, 24649, 24709, 24835, 25473, 26624, 28693, 28707, 28928, 30736, 32321, 30, 57, 114, 148, 162, 228, 274,
    289, 456, 522, 529, 912, 992, 1033, 1824, 2053, 3648, 4099, 4102, 4108, 4120, 4144, 4192, 4288, 4480, 4864,
    5632, 7296, 7682, 14592, 15368, 16497, 16545, 16657, 16905, 17413, 18435, 24065, 24657, 24713, 24837, 25025,
    25091, 28709, 28739, 29184, 30752, 32385, 29, 58, 116, 164, 232, 276, 290, 321, 464, 530, 545, 928, 1034,
    1041, 1856, 2057, 3712, 3841, 4101, 7424, 7684, 8195, 8198, 8204, 8216, 8240, 8288, 8384, 8576, 8960, 9728,
    11264, 14848, 15376, 16441, 16673, 16913, 17417, 18437, 20483, 24721, 24801, 24841, 25093, 25603, 28713,
    28741, 28803, 29696, 30465, 30731, 30784, 31746, 32515, 15, 168, 292, 322, 532, 546, 577, 1042, 1057, 2058,
    2065, 3842, 4105, 7688, 8197, 15392, 16390, 16396, 16408, 16413, 16432, 16480, 16576, 16705, 16768, 16929,
    17152, 17425, 17920, 18441, 19456, 20225, 20485, 22528, 24689, 24737, 24849, 25097, 25605, 26627, 28745,
    28805, 28931, 29569, 30739, 30848, 31748, 32517, 32705, 296, 324, 496, 548, 578, 641, 1044, 1058, 1089, 1921,
    2066, 2081, 3844, 4032, 4106, 4113, 7696, 8201, 15424, 15873, 16256, 16399, 16961, 17441, 18449, 20489,
    24633, 24865, 25105, 25609, 26629, 28753, 28809, 28933, 29121, 29187, 30741, 30755, 30976, 31751, 31752,
    31757, 31769, 31793, 31841, 31937, 32129, 32521, 27, 51, 54, 99, 102, 108, 195, 198, 204, 216, 328, 387, 390,
    396, 408, 432, 552, 580, 642, 771, 774, 780, 792, 816, 864, 1060, 1090, 1153, 1539, 1542, 1548, 1560, 1584,
    1632, 1728, 1922, 2068, 2082, 2113, 3075, 3078, 3084, 3096, 3120, 3168, 3264, 3456, 3848, 4114, 4129, 6147,
    6150, 6156, 6168, 6192, 6240, 6336, 6528, 6912, 7712, 8202, 8209, 12291, 12294, 12300, 12312, 12336, 12384,
    12480, 12672, 13056, 13824, 15488, 15874, 17025, 17473, 18305, 18465, 20497, 24582, 24588, 24600, 24605,
    24624, 24672, 24768, 24897, 24960, 25121, 25344, 25617, 26112, 26633, 27648, 28417, 28817, 28897, 28937,
    29189, 29699, 30757, 30787, 31232, 31760, 32529, 53, 101, 197, 336, 389, 584, 644, 773, 961, 1064, 1092,
    1154, 1281, 1541, 1924, 2084, 2114, 2177, 3077, 3856, 4116, 4130, 4161, 6149, 7744, 8210, 8225, 12293, 15616,
    15876, 16394, 16411, 16435, 16483, 16579, 16771, 17155, 17537, 17923, 18497, 19459, 20513, 22531, 24591,
    25153, 25633, 26641, 28785, 28833, 28945, 29193, 29701, 30761, 30789, 30851, 31776, 32545, 105, 201, 248,
    393, 592, 648, 777, 962, 1096, 1156, 1282, 1545, 1928, 2088, 2116, 2178, 2305, 3081, 3872, 4132, 4162, 4225,
    6153, 7808, 7937, 8212, 8226, 8257, 12297, 15880, 16402, 16437, 16485, 16581, 16773, 17157, 17345, 17665,
    17925, 18561, 19461, 20545, 22533, 25217, 25665, 26497, 26657, 28729, 28961, 29201, 29705, 30793, 30853,
    30979, 31617, 31755, 31808, 32577, 32640, 23, 46, 92, 106, 184, 202, 209, 368, 394, 401, 481, 656, 736, 778,
    785, 964, 1104, 1160, 1284, 1472, 1546, 1553, 1936, 2016, 2120, 2180, 2306, 2561, 2944, 3082, 3089, 3904,
    4136, 4164, 4226, 4353, 5888, 6154, 6161, 7938, 8228, 8258, 8321, 11776, 12298, 12305, 15888, 16404, 16418,
    16489, 16585, 16777, 17161, 17929, 18689, 19465, 20609, 22537, 23552, 24321, 24586, 24603, 24627, 24675,
    24771, 24963, 25347, 25729, 26115, 26689, 27651, 28678, 28684, 28696, 28701, 28720, 28768, 28864, 28993,
    29056, 29217, 29440, 29713, 30208, 30801, 30857, 30981, 31169, 31235, 31763, 31872, 32258, 45, 78, 156, 312,
    624, 672, 801, 1554, 2562, 14384, 16593, 16785, 17937, 24677, 31237, 31765 };

static int cbest_16[1023] = {
    1, 2, 34821, 4, 8, 52231, 16, 60934, 34820, 32, 17410, 3, 8705, 17, 34, 5, 68, 34823, 34829, 6, 136, 30467,
    34817, 39173, 52230, 64, 272, 52229, 9, 544, 52227, 10, 1088, 12, 2176, 26115, 4352, 50311, 60935, 18, 8704,
    45956, 60932, 20, 128, 24, 34837, 47876, 17408, 30466, 33, 36, 17411, 52239, 59974, 40, 22978, 48, 256, 8707,
    17414, 34822, 34828, 43524, 60930, 65286, 19, 35, 8709, 17418, 23938, 26114, 34816, 34836, 34853, 39172,
    52247, 21, 69, 15233, 21762, 30465, 34825, 52225, 52228, 7, 25, 38, 72, 137, 13057, 19586, 34831, 34881,
    52226, 42, 65, 80, 273, 10881, 26113, 29987, 34819, 34855, 39175, 50, 66, 70, 76, 545, 9793, 11489, 45957,
    60933, 60942, 11, 14, 84, 96, 138, 1089, 11969, 34957, 37253, 50310, 52261, 13, 152, 274, 1090, 2177, 34885,
    34949, 35093, 39169, 50309, 52246, 52263, 49, 144, 257, 546, 4353, 8713, 32643, 32901, 34845, 35365, 38341,
    47877, 60950, 22, 28, 140, 512, 2180, 4354, 25155, 30471, 35909, 40261, 45716, 52291, 52295, 100, 129, 160,
    276, 2178, 8706, 17426, 26123, 39717, 49351, 52235, 59975, 44, 98, 130, 168, 548, 4360, 8708, 8721, 34833,
    51271, 52237, 60928, 81, 132, 192, 304, 514, 1092, 15232, 30464, 30475, 34839, 34861, 39181, 43525, 45958,
    47872, 50307, 52238, 52367, 53703, 60931, 60951, 60966, 65287, 51, 88, 102, 204, 280, 408, 816, 1632, 3264,
    6528, 8720, 13056, 17409, 17412, 26119, 45952, 47396, 52243, 52503, 59972, 60964, 26, 37, 85, 170, 196, 288,
    340, 552, 680, 1360, 2720, 4356, 5440, 10880, 17416, 22858, 22976, 23936, 26112, 29986, 34844, 40565, 40805,
    52245, 52775, 54663, 41, 56, 153, 162, 200, 306, 612, 1096, 1224, 2448, 4896, 7616, 8737, 9792, 11488, 17422,
    17440, 21760, 22979, 30483, 34849, 34852, 39189, 47044, 47878, 49895, 50855, 59494, 60454, 65284, 145, 320,
    336, 1028, 2184, 8711, 11968, 17415, 17442, 34824, 34889, 34893, 43520, 50583, 52224, 60940, 60994, 23, 29,
    258, 392, 560, 608, 17419, 23698, 23939, 30482, 34827, 34830, 34832, 34863, 34880, 43526, 50319, 52259,
    56583, 60943, 64806, 46, 58, 74, 176, 260, 324, 384, 1024, 1104, 3808, 8225, 8712, 14993, 17427, 17474,
```

src/cauchy_best_r6.c lines 661 to 720

```
    19584, 21763, 32642, 34818, 34838, 34854, 39174, 45964, 52257, 57574, 59970, 39, 73, 82, 92, 264, 290, 2192,
    5744, 5984, 8769, 11429, 19587, 25154, 26131, 29985, 30227, 30470, 34869, 35077, 38933, 39205, 41604, 43532,
    52242, 52359, 58054, 60948, 61070, 63366, 65294, 43, 52, 112, 116, 400, 2056, 4368, 8739, 12577, 16450,
    17478, 20802, 21766, 26122, 29747, 30497, 34841, 34851, 34956, 37252, 42212, 50327, 52244, 52255, 52260,
    60998, 61206, 65282, 67, 71, 77, 89, 178, 184, 576, 672, 1904, 8773, 10401, 10883, 11425, 11849, 13061,
    15235, 15241, 17546, 18626, 22850, 22982, 26121, 26130, 30499, 34884, 34948, 34965, 35092, 39168, 39188,
    42692, 45700, 45717, 45972, 47884, 50191, 50308, 52233, 52262, 59014, 60938, 15, 27, 78, 97, 139, 148, 164,
    289, 352, 356, 1120, 1216, 2992, 8717, 8841, 9313, 13059, 13065, 17538, 17682, 19594, 23522, 30469, 30474,
    32900, 34857, 34883, 35364, 37013, 37249, 38340, 39171, 44612, 45836, 52293, 59906, 60246, 57, 86, 232, 275,
    640, 784, 1091, 2872, 8715, 8977, 9797, 17424, 17434, 17954, 23942, 26118, 26147, 29953, 32641, 34897, 34901,
    34945, 34981, 35908, 37255, 38221, 39233, 40260, 49349, 51127, 52290, 52294, 59766, 59982, 60110, 60929,
    60949, 154, 224, 321, 368, 547, 648, 712, 952, 4112, 4384, 9795, 11491, 15237, 16321, 17430, 22986, 25153,
    26117, 26145, 28787, 29027, 29955, 32403, 32897, 34871, 34887, 34951, 38337, 39207, 39716, 40533, 45596,
    45959, 47873, 47892, 49350, 51007, 51269, 52234, 52241, 52271, 52289, 52303, 60946, 60967, 104, 118, 141,
    156, 259, 328, 393, 513, 516, 561, 580, 704, 768, 800, 1496, 2181, 2208, 4355, 8725, 8736, 11393, 11457,
    11971, 19590, 21106, 21346, 22306, 22918, 29507, 30473, 30535, 34868, 34917, 34959, 35009, 35013, 35076,
    40021, 45701, 45953, 45973, 47397, 49231, 50305, 50341, 51270, 52236, 52269, 52279, 52311, 52487, 59973,
    60965, 30, 59, 101, 142, 161, 177, 186, 261, 277, 325, 385, 464, 520, 578, 786, 1105, 1344, 1424, 2179, 2182,
    4362, 8710, 8723, 10885, 17420, 19170, 20130, 23946, 24675, 25635, 29991, 30603, 31683, 32647, 34835, 34840,
    34860, 34909, 34953, 34973, 35095, 35101, 35911, 39180, 39309, 39477, 40517, 40549, 40737, 50306, 50326,
    50343, 52254, 52277, 52366, 53702, 59910, 59990, 60962, 61062, 45, 54, 83, 93, 99, 114, 131, 134, 146, 169,
    172, 194, 236, 265, 278, 291, 358, 476, 528, 549, 650, 736, 1094, 1122, 1436, 1572, 2048, 2193, 2240, 2432,
    2856, 4361, 8724, 8741, 11493, 17450, 17482, 19858, 21770, 22786, 22914, 22994, 30531, 32903, 32909, 34847,
    34913, 34964, 35089, 35367, 35373, 38343, 39183, 39237, 39719, 40535, 45988, 47045, 47879, 49827, 51267,
    52327, 52365, 52502, 53583, 53701, 54661, 59426, 59495, 60455, 60958, 65285, 53, 113, 117, 133, 193, 234,
    305, 308, 372, 448, 468, 515, 550, 582, 642, 716, 748, 936, 1093, 1152, 1300, 1408, 1428, 2210, 2848, 3144,
    4369, 8224, 8722, 11153, 11459, 11841, 11973, 14992, 17446, 17448, 22798, 25163, 26127, 30481, 34877, 34983,
    35333, 35361, 35917, 39177, 39685, 40263, 40564, 40804, 43521, 43540, 45236, 45476, 45580, 45718, 47636,
    47893, 49893, 50183, 50371, 50567, 50991, 52251, 52299, 52325, 52363, 52501, 52774, 53575, 54662, 60452,
    60941, 60947, 60995, 65302, 103, 166, 179, 185, 205, 281, 296, 312, 354, 370, 409, 472, 714, 740, 817, 928,
    1164, 1432, 1480, 1568, 1633, 1872, 2244, 2600, 3265, 4358, 4364, 4386, 5712, 6288, 6529, 8768, 8833, 9729,
    10065, 10673, 10889, 11397, 11428, 11761, 17413, 17444, 22790, 22856, 23682, 23954, 26183, 29713, 29984,
    29995, 30123, 30226, 34848, 34859, 34896, 34900, 35905, 40257, 43527, 45116, 45572, 45712, 45828, 46016,
    47156, 47392, 47874, 49894, 50854, 50983, 52307, 52499, 52773, 60960, 60974, 60992, 64807, 90, 149, 171, 197,
    208, 238, 282, 341, 353, 357, 518, 553, 681, 744, 770, 1098, 1156, 1280, 1296, 1361, 1472, 1634, 2188, 2328,
    2721, 2864, 3266, 3744, 4357, 4420, 5200, 5441, 6530, 7496, 8729, 8738, 9585, 9801, 9929, 10553, 11395,
    11497, 12576, 14977, 15249, 17417, 21778, 22794, 22859, 22977, 23696, 23818, 23937, 26129, 26251, 29715,
    29746, 30055, 30479, 30480, 30496, 33989, 34870, 34888, 34892, 34905, 34915, 34919, 34989, 35017, 38213,
    39191, 39713, 40741, 45954, 45965, 45990, 46924, 47908, 49347, 50317, 50582, 50735, 50853, 52240, 52267,
    52771, 54423, 56581, 57506, 57510, 57575, 59492, 59971, 59991, 60102, 60936, 60982, 198, 374, 1097, 1225,
    7488, 15113, 17441, 17472, 26387, 29883, 30211, 34879, 35205, 50463, 50999, 52258, 61071 };


static int cbest_17[1023] = {
    1, 2, 65540, 4, 32770, 8, 16385, 16, 32, 73732, 64, 128, 36866, 256, 512, 18433, 1024, 2048, 4096, 74756,
    8192, 16384, 37378, 32768, 3, 18689, 65536, 65541, 5, 32771, 65542, 6, 9, 18, 36, 72, 144, 288, 576, 1152,
    2304, 4608, 9216, 18432, 98310, 10, 17, 16387, 32774, 36864, 65548, 81925, 12, 33, 16389, 32778, 49155,
    65556, 73728, 73733, 74884, 20, 34, 65, 16393, 32786, 65572, 73734, 24, 66, 129, 16401, 32802, 36867, 65604,
    106502, 40, 68, 130, 257, 16417, 32834, 65668, 73740, 90117, 102406, 48, 132, 258, 513, 16449, 32898, 36870,
    37442, 65796, 73748, 80, 136, 260, 514, 1025, 16513, 18435, 33026, 36874, 53251, 66052, 73764, 83973, 96,
    264, 516, 1026, 2049, 16641, 18437, 33282, 36882, 51203, 66564, 73796, 160, 272, 520, 1028, 2050, 4097,
    16897, 18441, 18688, 33794, 36898, 37376, 67588, 73860, 74752, 74757, 110598, 192, 528, 1032, 2052, 4098,
    8193, 9344, 17409, 18449, 18721, 34818, 36930, 69636, 73988, 74758, 320, 544, 1040, 2056, 4100, 4672, 8194,
    18465, 36994, 74244, 92165, 107526, 384, 1056, 2064, 2336, 4104, 8196, 16386, 18497, 20481, 37122, 37379,
    40962, 74764, 81924, 91141, 640, 1088, 1168, 2080, 4112, 8200, 16388, 18561, 24577, 32769, 49154, 55299,
    74772, 75780, 102918, 584, 768, 2112, 4128, 8208, 16392, 37382, 37890, 74788, 77828, 98308, 292, 1280, 2176,
    4160, 8224, 16400, 18945, 32772, 32784, 37386, 38914, 53763, 65537, 74820, 74900, 146, 1536, 4224, 8256,
    16416, 18691, 19457, 32776, 37394, 49153, 65538, 65543, 65568, 84229, 90116, 111622, 7, 19, 37, 73, 145, 289,
    577, 1153, 2305, 2560, 4352, 4609, 8320, 9217, 16448, 18693, 37410, 45058, 51459, 75012, 98306, 98311,
    111110, 11, 38, 74, 290, 578, 1154, 2306, 3072, 4610, 8448, 9218, 16512, 18434, 18697, 22529, 32775, 32800,
```

src/cauchy_best_r6.c lines 721 to 780

```
    36865, 53250, 65544, 65549, 65558, 65612, 65684, 65828, 66116, 66692, 67844, 70148, 75268, 81921, 83972,
    93189, 106500, 13, 22, 76, 148, 580, 1156, 2308, 4612, 5120, 8704, 9220, 16640, 18436, 18705, 26625, 32779,
    32806, 32832, 32842, 32914, 33058, 33346, 33922, 35074, 37506, 41986, 51202, 65550, 65552, 65557, 73729,
    74880, 74885, 81927, 102404, 14, 21, 26, 35, 44, 152, 296, 1160, 2312, 4616, 6144, 9224, 16391, 16403, 16421,
    16457, 16529, 16673, 16896, 16961, 17537, 18440, 20993, 25601, 32787, 32896, 36868, 36880, 37440, 37450,
    37634, 65573, 73730, 73735, 73760, 74886, 76804, 92421, 98318, 114695, 25, 52, 67, 88, 304, 592, 2320, 4624,
    9232, 10240, 16395, 17408, 18448, 18720, 18753, 32782, 32803, 33024, 36872, 49159, 53249, 55811, 65574,
    65600, 65605, 78852, 81933, 98326, 106498, 106503, 107654, 41, 50, 69, 104, 131, 176, 608, 1184, 4640, 9248,
    12288, 16397, 18464, 18817, 32790, 32835, 33280, 38402, 49163, 51201, 55555, 65564, 65606, 65664, 65669,
    73736, 73741, 73750, 73804, 73876, 74020, 74308, 74892, 76036, 78340, 81941, 90113, 91269, 92164, 98342,
    102402, 102407, 28, 42, 49, 70, 82, 100, 133, 208, 259, 352, 1216, 2368, 9280, 9360, 16405, 16419, 18496,
    20480, 32794, 32899, 33792, 36871, 36896, 37443, 39426, 49171, 65580, 65670, 65792, 65797, 73742, 73744,
    73749, 81957, 90119, 91140, 98374, 110596, 81, 134, 137, 164, 200, 261, 416, 515, 704, 2432, 4736, 16409,
    16451, 18560, 19201, 24576, 32810, 32838, 33027, 34816, 36875, 36902, 36928, 36938, 37010, 37154, 38018,
    39170, 46082, 49187, 55298, 65588, 65798, 66048, 66053, 73765, 74916, 81989, 83969, 98438, 102982, 106510,
    112134, 122887, 97, 138, 262, 265, 274, 328, 400, 517, 832, 1027, 1408, 4680, 4864, 9472, 16425, 16453,
    16515, 18725, 19713, 32818, 32902, 33283, 36883, 36992, 37446, 45570, 49219, 65620, 65676, 66054, 66560,
    66565, 73766, 73792, 73797, 74948, 82053, 83975, 90125, 98566, 102414, 106518, 107524, 118791, 56, 84, 98,
    140, 161, 266, 273, 518, 521, 530, 548, 656, 800, 1029, 1664, 2051, 2816, 9728, 16433, 16517, 16643, 18439,
    18451, 18469, 18505, 18577, 18944, 19009, 19585, 23041, 27649, 32850, 32906, 33030, 33795, 36878, 36899,
    37120, 37377, 40960, 49283, 53255, 53762, 53827, 65636, 65804, 66566, 67584, 67589, 73756, 73798, 73856,
    73861, 74753, 82181, 90133, 98822, 102422, 106534, 110594, 110599, 111750, 116743, 162, 193, 268, 522, 529,
    1030, 1033, 1042, 1060, 1096, 1312, 1600, 2053, 2340, 3328, 4099, 5632, 9345, 16465, 16521, 16645, 16899,
    18443, 18690, 19456, 22785, 32866, 33034, 33286, 34819, 36886, 36931, 37458, 49152, 49411, 51207, 53259,
    55297, 65700, 65812, 66060, 67590, 69632, 69637, 73772, 73862, 73984, 73989, 74754, 74759, 74784, 75140,
    82437, 83981, 84228, 90149, 93445, 99334, 102438, 102916, 106566, 194, 276, 321, 524, 545, 1034, 1041, 2054,
    2057, 2066, 2084, 2120, 2192, 2624, 3200, 4101, 4673, 6656, 8195, 9346, 11264, 16481, 16649, 16901, 17411,
    18445, 18692, 18723, 26881, 32930, 33042, 33290, 33798, 36890, 36995, 37380, 37392, 37474, 37888, 49667,
    51211, 51458, 53267, 65732, 66068, 66572, 69638, 73780, 73990, 74240, 74245, 75396, 82949, 83989, 84261,
    90181, 92161, 93317, 100358, 102470, 106630, 107522, 107527, 111174, 112, 168, 196, 280, 322, 385, 532, 546,
    1036, 1057, 1170, 2058, 2065, 2337, 4102, 4105, 4114, 4132, 4168, 4240, 4384, 4674, 5248, 6400, 8197, 9348,
    13312, 16545, 16657, 16905, 17413, 18453, 18467, 18696, 22528, 32962, 33298, 33802, 34822, 36906, 36934,
    37123, 37384, 38912, 40963, 42114, 50179, 51219, 51491, 53283, 53761, 65860, 66084, 66580, 67596, 70212,
    73812, 73868, 74246, 74760, 74765, 74774, 74828, 75044, 75332, 77060, 79364, 81920, 84005, 90245, 91137,
    92167, 93188, 102534, 106758, 110606, 126983, 324, 386, 536, 641, 1044, 1058, 1089, 1169, 2060, 2081, 2338,
    4106, 4113, 4676, 8198, 8201, 8210, 8228, 8264, 8336, 8480, 8768, 9352, 10496, 12800, 16577, 16913, 17417,
    18457, 18499, 18704, 18729, 20483, 25729, 26624, 33090, 33314, 33810, 34826, 36914, 36998, 37570, 51235,
    53315, 56067, 65924, 66596, 67604, 67876, 69644, 73828, 73996, 74766, 74768, 74773, 75776, 75781, 76932,
    81926, 84037, 86021, 90373, 91143, 102662, 102914, 102919, 107014, 110614, 111620, 124935, 388, 552, 585,
    642, 769, 1048, 1090, 2068, 2082, 2113, 4108, 4129, 8202, 8209, 16390, 16402, 16420, 16456, 16528, 16672,
    16705, 16929, 16960, 17425, 17536, 18473, 18501, 18563, 18737, 20485, 20992, 21057, 24579, 25600, 32912,
    33154, 33826, 34834, 35106, 36946, 37002, 37126, 37383, 37408, 37698, 37891, 40966, 45056, 51267, 51457,
    53379, 65824, 66180, 66628, 66708, 67620, 69652, 73892, 74004, 74252, 74789, 74902, 75782, 77824, 77829,
    78980, 84101, 90629, 92173, 92420, 98304, 98309, 107534, 110630, 111108, 114694, 120839, 123911, 392, 644,
    770, 1281, 2072, 2114, 2177, 4116, 4130, 4161, 8204, 8225, 16769, 17441, 18565, 24581, 33410, 33938, 34850,
    36962, 37130, 38530, 38915, 40970, 46594, 49158, 53248, 55303, 55810, 69668, 73924, 74260, 74790, 74821,
    74896, 74901, 77830, 91149, 92181, 92453, 107542, 110662 };

static int cbest_18[1023] = {
    1, 2, 131136, 4, 65568, 8, 32784, 16, 16392, 32, 8196, 64, 4098, 128, 2049, 256, 512, 132160, 1024, 2048,
    66080, 4096, 33040, 8192, 16520, 8260, 16384, 3, 4130, 32768, 131137, 5, 65569, 131138, 6, 9, 2065, 32785,
    65536, 65570, 131140, 196704, 10, 17, 16393, 32786, 65572, 131144, 163920, 12, 18, 33, 8197, 16394, 32788,
    65576, 98352, 131072, 131152, 147528, 20, 34, 65, 4099, 8198, 16396, 32792, 65584, 81960, 131168, 132168,
    139332, 24, 36, 66, 129, 258, 516, 1032, 2064, 49176, 73764, 135234, 40, 68, 130, 257, 2051, 4102, 4128,
    8204, 16408, 32816, 40980, 65632, 69666, 131264, 133185, 48, 72, 132, 513, 2053, 4106, 8212, 8256, 16424,
    24588, 32848, 36882, 65696, 66084, 67617, 131392, 132161, 80, 136, 260, 514, 1025, 2057, 4114, 8228, 16456,
    16512, 20490, 32912, 34833, 65824, 131648, 132162, 96, 144, 264, 1026, 12294, 18441, 33024, 66081, 132164,
    197728, 160, 272, 520, 1028, 2050, 2081, 4162, 8324, 10245, 16648, 33042, 33296, 66048, 66082, 66592, 133184,
    164944, 197216, 192, 288, 528, 2052, 2113, 4097, 4226, 6147, 8452, 16904, 33041, 33808, 67616, 132096,
    132176, 148552, 320, 544, 1040, 2056, 2177, 4354, 8708, 17416, 34832, 66088, 98864, 132192, 135232, 140356,
```

src/cauchy_best_r6.c lines 781 to 840

```
    164176, 384, 576, 1056, 2305, 4100, 4610, 8193, 9220, 16521, 18440, 33044, 66096, 69664, 82472, 98608,
    136258, 640, 1088, 2080, 2561, 4104, 5122, 8194, 10244, 16522, 33048, 36880, 74276, 132288, 134209, 139268,
    139328, 147656, 768, 1152, 2112, 3073, 4112, 6146, 8261, 16385, 16524, 20488, 49432, 66144, 69634, 70178,
    73760, 82088, 132416, 1280, 2176, 8200, 8262, 12292, 16386, 33072, 34817, 40976, 41236, 49304, 66208, 68129,
    132672, 147520, 1536, 2304, 4131, 4160, 4352, 8208, 16388, 16536, 24584, 32769, 33104, 37138, 66336, 73828,
    81952, 2560, 4224, 6145, 8224, 8268, 8704, 16552, 24716, 32770, 33168, 35089, 41044, 49168, 131139, 134208,
    135266, 148544, 163904, 198240, 7, 3072, 4134, 8276, 12290, 16400, 16584, 17408, 20618, 24652, 32772, 65537,
    65571, 67104, 98336, 131141, 196705, 11, 2067, 4138, 4608, 8292, 8320, 10241, 16416, 18569, 24580, 32776,
    32787, 33552, 34816, 36914, 65538, 65573, 68128, 74272, 131142, 131145, 133201, 136256, 163921, 165200,
    196672, 196706, 13, 19, 2069, 4146, 5120, 8448, 12358, 16395, 16448, 16776, 20482, 20522, 32789, 34064,
    49160, 65540, 65574, 65577, 67633, 98353, 131073, 131146, 131153, 147529, 163922, 196708, 14, 21, 35, 2073,
    6144, 8199, 8388, 10309, 12326, 16397, 17032, 18433, 32790, 32793, 32800, 35088, 37136, 40964, 65544, 65578,
    65585, 69632, 70176, 81961, 98320, 98354, 99120, 131074, 131148, 131154, 131169, 132169, 139333, 140292,
    140352, 147530, 148680, 163924, 196712, 229488, 22, 25, 37, 67, 259, 517, 1033, 4194, 8516, 9216, 16398,
    16640, 17544, 18457, 32794, 32832, 36866, 49177, 65552, 65580, 65586, 73765, 81928, 81962, 98356, 131076,
    131156, 131170, 132170, 135235, 139334, 147532, 163928, 196640, 196720, 213096, 26, 38, 41, 69, 131, 518,
    1034, 2066, 2097, 4103, 4129, 4258, 6179, 8205, 8772, 10240, 10261, 16409, 16896, 18568, 32796, 32817, 32896,
    40981, 49178, 65588, 65633, 69667, 70146, 73732, 73766, 81964, 82600, 98360, 131080, 131160, 131172, 131265,
    131394, 131652, 132172, 133200, 139264, 163856, 180312, 197736, 204900, 28, 42, 49, 70, 73, 133, 262, 1036,
    2068, 2129, 4107, 4386, 6163, 8206, 8213, 8257, 9284, 16410, 16425, 24589, 32818, 32849, 36883, 40982, 49180,
    65592, 65600, 65634, 65697, 65826, 66085, 66600, 67632, 114744, 131088, 131176, 131266, 131393, 133187,
    135238, 135264, 139340, 147464, 147544, 163952, 164952, 172116, 200802, 44, 50, 74, 81, 134, 137, 261, 266,
    515, 524, 2055, 2072, 2193, 4115, 4132, 4642, 8214, 8229, 8258, 10308, 12288, 16412, 16426, 16457, 16513,
    18432, 20491, 20616, 24590, 32820, 32850, 32913, 33026, 33280, 33300, 33816, 35073, 41232, 49560, 65636,
    65664, 65698, 65825, 66052, 66086, 67585, 67619, 69670, 73772, 74340, 81976, 82464, 106548, 131104, 131184,
    131268, 131649, 132104, 132163, 132184, 133189, 135242, 136290, 139348, 147560, 155724, 164928, 168018,
    196832, 197220, 198753, 52, 76, 82, 97, 138, 145, 265, 274, 532, 1027, 1048, 2059, 2321, 4110, 4136, 5154,
    8230, 12295, 16428, 16458, 16514, 16650, 16908, 18456, 32824, 32852, 32914, 33025, 33792, 34835, 36886,
    36912, 40988, 65640, 65700, 65792, 67621, 69674, 73780, 73824, 90156, 98416, 102450, 131272, 131396, 131650,
    132165, 132200, 133193, 135170, 135250, 139364, 140364, 147648, 151626, 164048, 165969, 196960, 197729, 56,
    84, 98, 140, 146, 161, 273, 290, 521, 548, 1029, 1064, 2061, 2096, 2577, 4118, 4144, 4163, 6178, 8220, 8264,
    8325, 8454, 9228, 10260, 12356, 16460, 16516, 16649, 18443, 20480, 20494, 20520, 24712, 32856, 32916, 33043,
    33297, 34837, 36890, 41040, 41300, 49208, 49424, 57372, 65648, 65704, 65828, 66049, 66083, 66092, 66593,
    67625, 69682, 82024, 82080, 86058, 98480, 98848, 98868, 100401, 131200, 131280, 131400, 132166, 133121,
    134225, 136266, 143430, 149577, 164160, 164945, 197217, 197696, 197730, 88, 100, 148, 162, 193, 268, 289,
    322, 522, 529, 580, 1030, 1096, 2083, 2128, 3089, 4122, 4227, 4614, 5130, 6162, 8236, 8272, 8326, 8453,
    10247, 12324, 16440, 16905, 18445, 24604, 24648, 32920, 33028, 33298, 33809, 34841, 41012, 49240, 49296,
    53274, 65712, 65832, 66050, 66100, 66560, 66594, 77862, 82476, 84009, 98592, 131296, 131328, 131408, 131656,
    132097, 132177, 132296, 133186, 133217, 134217, 139460, 141381, 147784, 148553, 148672, 164178, 164432,
    164946, 197184, 197218, 197732, 104, 152, 164, 194, 276, 321, 386, 530, 545, 644, 1041, 1160, 2054, 2085,
    2115, 2192, 2307, 2565, 3081, 4166, 4192, 4355, 4384, 8244, 8288, 8709, 8768, 12302, 16472, 16528, 16652,
    16906, 17417, 17536, 20506, 24576, 24620, 24780, 32880, 33032, 33046, 33810, 35072, 36864, 37170, 45078,
    51225, 65840, 66089, 66596, 67584, 67618, 68145, 69730, 73892, 75813, 82216, 98610, 98865, 131424, 131584,
    131664, 132098, 132178, 132193, 132424, 133188, 135233, 135362, 137283, 139588, 140357, 148040, 148554,
    164177, 164948, 198752, 230512, 112, 168, 196, 280, 292, 385, 546, 577, 772, 1042, 1057, 1288, 2058, 2089,
    2117, 2179, 2320, 4101, 4170, 4230, 4256, 4611, 6151, 6177, 8332, 8710, 9221, 10253, 12310, 12354, 16488,
    16544, 17418, 24708, 28686, 32944, 33045, 33050, 33304, 33812, 34834, 34865, 36946, 41108, 43029, 49416,
    65760, 66056, 66090, 66097, 66148, 67620, 67681, 69665, 69794, 70144, 70182, 71715, 74020, 82473, 98609,
    98832, 98866, 99376, 131680, 132100, 132180, 132194, 132680, 133192, 133313, 135490, 136259, 139844, 140358,
    148556, 165968, 197224, 197664, 197744, 214120, 230000, 176, 200, 296, 324, 536, 641, 770, 1044, 1058, 1089,
    1544, 2060, 2121, 2181, 4105, 4178, 4234, 4358, 8460, 9222, 10305, 16576, 17420, 18442, 18473, 20554, 20610,
    32976, 33049, 33056, 34836, 34897, 38931, 41220, 65888, 66098, 66608, 67624, 67745, 68133, 74277, 74336,
    82440, 82474, 131520, 132196, 132418, 133120, 133441, 134224, 135746, 164880, 181336, 197232, 205924, 229744 };


static int cbest_19[1023] = {
    1, 2, 262163, 4, 393242, 8, 196621, 16, 32, 360469, 64, 262162, 128, 131081, 3, 442393, 262161, 6, 256,
    393243, 5, 327703, 12, 196620, 512, 393240, 9, 24, 98310, 262167, 426008, 10, 458782, 48, 1024, 49155,
    196617, 17, 213004, 262171, 393234, 483359, 18, 96, 262147, 360471, 393246, 20, 2048, 16385, 196623, 286738,
    360468, 442392, 33, 192, 32770, 106502, 229391, 34, 65540, 196613, 270355, 360465, 36, 384, 4096, 131080,
```

```
    143369, 180234, 393226, 40, 65, 53251, 221196, 262195, 66, 768, 376852, 397338, 442395, 68, 8192, 24577,
    49154, 90117, 503836, 72, 129, 1536, 262160, 262227, 80, 110598, 198669, 288786, 393274, 132, 3072, 16384,
    98308, 188426, 196637, 274451, 327702, 333847, 483358, 7, 130, 136, 257, 131083, 262291, 307217, 144, 6144,
    49153, 163851, 213005, 262165, 286739, 327699, 360477, 393241, 393306, 426010, 13, 160, 55299, 131073,
    144393, 196616, 196653, 262166, 327701, 360453, 361493, 426009, 14, 26, 258, 264, 513, 12288, 53250, 94213,
    131085, 196609, 262419, 399386, 442385, 458783, 25, 52, 272, 26625, 98306, 98311, 251918, 344086, 360467,
    393370, 429080, 442397, 11, 104, 288, 24576, 32768, 90116, 106500, 196619, 196622, 196685, 229389, 262155,
    262170, 262175, 311312, 368661, 393218, 393235, 393238, 393244, 405530, 458778, 22, 28, 49, 208, 260, 320,
    514, 1025, 45058, 155656, 180235, 229390, 241679, 262146, 262169, 262675, 360470, 376853, 393232, 393247,
    442394, 458780, 44, 416, 528, 22529, 77828, 172043, 180232, 196612, 199693, 213000, 221197, 262145, 262179,
    289810, 360501, 393498, 415771, 442905, 483355, 491548, 19, 88, 97, 544, 832, 38914, 53249, 143368, 196749,
    275475, 309265, 442377, 475167, 483357, 21, 38, 50, 56, 176, 516, 576, 1026, 1664, 2049, 19457, 71684,
    114695, 131097, 135177, 202765, 214540, 263187, 270354, 288787, 334359, 360464, 426000, 76, 193, 352, 640,
    3328, 32771, 35842, 49152, 106498, 106503, 139273, 213006, 229387, 262151, 278546, 286736, 327711, 352278,
    360533, 393227, 393754, 426012, 446489, 458774, 35, 98, 152, 194, 704, 1056, 6656, 17921, 55298, 65536,
    65541, 196629, 196877, 245774, 262194, 262259, 273427, 294929, 319504, 327687, 348182, 376854, 393224,
    393258, 450585, 37, 42, 70, 112, 304, 385, 520, 1028, 1088, 1408, 2050, 4097, 13312, 16387, 49159, 94212,
    110596, 125959, 131113, 144905, 188427, 196615, 212996, 262193, 264211, 270353, 315408, 329751, 360473,
    362005, 372757, 382996, 397339, 483351, 503838, 41, 140, 608, 1152, 2816, 26624, 27649, 98318, 159752,
    188424, 198668, 221192, 262355, 271891, 331799, 360597, 394266, 399898, 442425, 499740, 503837, 67, 100, 280,
    388, 769, 1216, 1280, 5632, 47106, 65542, 98304, 99334, 107270, 157704, 176139, 196633, 196636, 197133,
    262226, 363541, 376848, 393230, 406554, 409627, 425992, 427032, 442384, 69, 74, 84, 224, 386, 560, 1032,
    2052, 2112, 2432, 4098, 8193, 11264, 16389, 49163, 49667, 79876, 90113, 122887, 131082, 131145, 143371,
    163849, 174091, 229383, 262199, 262225, 262547, 266259, 271123, 286742, 344087, 360479, 393275, 393338,
    397336, 428056, 429336, 442396, 458766, 470046, 483615, 503832, 73, 196, 1120, 1537, 2176, 4864, 22528,
    23553, 32774, 55297, 78852, 81925, 98326, 163850, 213516, 221198, 262203, 274450, 307219, 311313, 327735,
    333846, 360455, 360725, 393266, 393272, 395290, 398874, 416795, 442399, 442457, 460830, 81, 134, 770, 776,
    2240, 2304, 9728, 39938, 53248, 110594, 110599, 131072, 137225, 143361, 144392, 180226, 196649, 196652,
    196669, 197645, 198665, 214028, 251919, 262290, 262931, 270359, 286994, 303121, 307216, 327697, 356374,
    360449, 393434, 434200, 442387, 483343, 133, 148, 168, 448, 1040, 2056, 2560, 3073, 4100, 4480, 8194, 16393,
    16897, 19456, 24579, 39426, 49171, 90119, 98309, 106758, 114694, 131084, 131209, 143373, 180238, 191498,
    196608, 199949, 221188, 249870, 251916, 262164, 262211, 262231, 275987, 286746, 288784, 289811, 323600,
    327698, 333843, 360461, 360476, 361495, 393307, 398106, 415770, 426011, 442904, 443161, 448537, 466974,
    483354, 485407, 131, 137, 200, 1538, 4224, 8960, 17409, 19969, 32778, 53255, 53635, 57347, 72196, 98342,
    106510, 107014, 131077, 143497, 153608, 155657, 196639, 196717, 203277, 213020, 217100, 262235, 263699,
    270611, 274449, 286722, 325136, 327700, 327767, 333845, 350230, 360452, 360981, 361492, 393278, 393298,
    393304, 393626, 397330, 426040, 442376, 442521, 475166, 483356, 487455, 82, 138, 145, 268, 772, 1552, 4352,
    6145, 17920, 18433, 19713, 33794, 45056, 53379, 65548, 106496, 166923, 180233, 196618, 196645, 196681,
    196684, 198671, 213001, 229385, 229388, 230415, 241677, 262289, 262418, 270363, 270867, 317456, 334615,
    349206, 360485, 397342, 397722, 399387, 417819, 429082, 442369, 442389, 443929, 458814, 503828, 146, 161,
    296, 336, 392, 896, 2064, 3074, 4104, 4608, 8196, 16386, 16401, 20481, 24581, 34818, 36098, 49158, 49187,
    53507, 76804, 108550, 125958, 131087, 131089, 131337, 161800, 172041, 180746, 196813, 199437, 214668, 221452,
    241678, 262153, 262275, 262295, 265235, 270339, 271379, 273939, 278547, 286737, 290834, 311314, 327707,
    330775, 352279, 360466, 360503, 361489, 368663, 393216, 393371, 394010, 397466, 413723, 429081, 446488,
    458770, 458776, 15, 27, 54, 108, 162, 216, 259, 265, 432, 864, 1540, 1728, 3456, 5120, 6912, 12289, 13824,
    27648, 32786, 36866, 38912, 49157, 53259, 67588, 94209, 98374, 106518, 131075, 172042, 178187, 196655,
    198661, 199692, 213036, 214541, 229407, 233487, 262154, 262174, 262299, 272403, 319505, 327831, 348183,
    360497, 368660, 376855, 393219, 393239, 393245, 393290, 393310, 393362, 397594, 401434, 405531, 426014,
    426072, 442379, 442649, 450584, 458779, 483391, 503964, 514077, 53, 266, 273, 536, 3104, 6146, 8448, 18049,
    24833, 38402, 40962, 54275, 65556, 69636, 86021, 90373, 98307, 114693, 124935, 145417, 158728, 188418,
    196611, 196677, 196745, 196748, 197005, 198733, 199053, 200717, 207885, 212992, 213007, 235023, 237583,
    251914, 262168, 262173, 262417, 262674, 268307, 288790, 288850, 315409, 339991, 344082, 376860, 377364,
    382997, 393233, 393236, 394778, 397850, 399384, 426002, 442907, 458781, 458846, 483347, 483350, 483353,
    491550, 30, 105, 262, 274, 289, 400, 592, 672, 1792, 2080, 3076, 4112, 8200, 8704, 16388, 16417, 24585,
    32769, 35840, 49162, 49219, 49666, 73732, 80900, 90112, 90125, 95749, 98314, 99846, 106501, 110726, 122886,
    131096, 131593, 135176, 143385, 144395, 175115, 180250, 184330, 188430, 198797, 202764, 223244, 245775,
    262144, 262149, 262178, 262183, 262323, 262403, 272147, 274455, 286770, 288914, 309267, 333911, 344084,
    360500, 360535, 360565, 362133, 368657, 376836, 388116, 393368, 393499, 397322, 398362, 400154, 407066,
    427544, 438296, 442424, 491549, 23, 29, 46, 92, 106, 184, 209, 261, 276, 290, 321, 368, 515, 736, 1472, 1544,
    2944, 5888, 9216, 11776, 12290, 19201, 23552, 32802, 45059, 53267, 55296, 81924, 94215, 98316, 98438, 106534,
```

src/cauchy_best_r6.c lines 901 to 960

```
    139272, 163843, 174603, 196687, 197389, 198861, 198925, 212997, 213002, 213068, 221212, 225292, 229386,
    229423, 241675, 245772, 262427, 272019, 274579, 275474, 288978, 305169, 307221, 309264, 327959, 345622,
    360529, 362517, 363797, 376849, 378900, 393354, 393374, 393490, 396314, 405528, 409626, 426136, 436248,
    461854, 470558, 471070, 483423, 503820, 504860, 417, 529, 1072, 6148, 10240, 92165, 101382, 125957, 144425,
    147464, 196741, 204813, 262159, 262339, 262673, 263186, 286743, 288794, 307345, 327683, 425984, 426001,
    432152, 442409, 458910, 505884 };

static int cbest_20[1023] = {
    1, 2, 524292, 4, 262146, 8, 131073, 16, 32, 589828, 64, 128, 294914, 256, 512, 147457, 1024, 2048, 4096,
    598020, 8192, 16384, 32768, 299010, 65536, 131072, 149505, 262144, 3, 524288, 524293, 5, 262147, 524294,
    599044, 6, 9, 18, 36, 72, 144, 288, 576, 1152, 2304, 4608, 9216, 18432, 36864, 73728, 147456, 786438, 10, 17,
    131075, 262150, 294912, 524300, 655365, 12, 33, 131077, 262154, 393219, 524308, 589824, 589829, 20, 34, 65,
    131081, 262162, 299522, 524324, 589830, 24, 66, 129, 131089, 262178, 294915, 524356, 851974, 40, 68, 130,
    257, 131105, 262210, 524420, 589836, 720901, 819206, 48, 132, 258, 513, 131137, 262274, 294918, 524548,
    589844, 80, 136, 260, 514, 1025, 131201, 147459, 149761, 262402, 294922, 425987, 524804, 589860, 671749, 96,
    264, 516, 1026, 2049, 131329, 147461, 262658, 294930, 409603, 525316, 589892, 160, 272, 520, 1028, 2050,
    4097, 131585, 147465, 149504, 263170, 294946, 299008, 526340, 589956, 598016, 598021, 884742, 192, 528, 1032,
    2052, 4098, 8193, 74752, 132097, 147473, 264194, 294978, 528388, 590084, 598022, 320, 544, 1040, 2056, 4100,
    8194, 16385, 37376, 133121, 147489, 266242, 295042, 532484, 590340, 737285, 860166, 384, 1056, 2064, 4104,
    8196, 16386, 18688, 32769, 135169, 147521, 270338, 295170, 299011, 540676, 590852, 598028, 599172, 729093,
    640, 1088, 2080, 4112, 8200, 9344, 16388, 32770, 65537, 139265, 147585, 278530, 295426, 442371, 557060,
    591876, 598036, 823302, 768, 2112, 4128, 4672, 8208, 16392, 32772, 65538, 147713, 295938, 299014, 593924,
    598052, 1280, 2176, 2336, 4160, 8224, 16400, 32776, 65540, 131074, 147969, 163841, 296962, 299018, 327682,
    430083, 598084, 655364, 1168, 1536, 4224, 8256, 16416, 32784, 65544, 131076, 148481, 149507, 196609, 262145,
    299026, 393218, 598148, 606212, 673797, 892934, 584, 2560, 4352, 8320, 16448, 32800, 65552, 131080, 149509,
    299042, 299586, 303106, 411651, 598276, 622596, 786436, 888838, 292, 3072, 8448, 16512, 32832, 65568, 131088,
    149513, 151553, 262148, 262160, 299074, 311298, 524289, 598532, 745477, 146, 5120, 8704, 16640, 32896, 65600,
    131104, 149521, 155649, 262152, 299138, 393217, 524290, 524295, 524320, 599040, 599045, 720900, 7, 19, 37,
    73, 145, 289, 577, 1153, 2305, 4609, 6144, 9217, 16896, 18433, 33024, 36865, 65664, 73729, 131136, 149537,
    299266, 299520, 360450, 599046, 600068, 739333, 786434, 786439, 11, 38, 74, 290, 578, 1154, 2306, 4610, 9218,
    10240, 17408, 18434, 33280, 36866, 65792, 73730, 131200, 147458, 149569, 149760, 180225, 262151, 262176,
    294913, 425986, 446467, 524296, 524301, 524310, 524364, 524436, 524580, 524868, 525444, 526596, 528900,
    533508, 542724, 561156, 602116, 655361, 671748, 851972, 861190, 13, 22, 76, 148, 580, 1156, 2308, 4612, 9220,
    12288, 18436, 33792, 36868, 66048, 73732, 131328, 147460, 149633, 149793, 212993, 262155, 262182, 262208,
    262218, 262290, 262434, 262722, 263298, 264450, 266754, 271362, 280578, 300034, 335874, 409602, 444419,
    524302, 524304, 524309, 589825, 599052, 655367, 730117, 819204, 14, 21, 26, 35, 44, 152, 296, 1160, 2312,
    4616, 9224, 18440, 20480, 34816, 36872, 66560, 73736, 74880, 131079, 131091, 131109, 131145, 131217, 131361,
    131584, 131649, 132225, 133377, 135681, 140289, 147464, 167937, 204801, 262163, 262272, 294916, 294928,
    299523, 301058, 524325, 589826, 589831, 589856, 599060, 614404, 786446, 917511, 25, 52, 67, 88, 304, 592,
    2320, 4624, 9232, 18448, 24576, 36880, 67584, 73744, 131083, 132096, 147472, 150017, 262158, 262179, 262400,
    294920, 393223, 425985, 524326, 524352, 524357, 599076, 630788, 655373, 786454, 823814, 851970, 851975,
    897030, 41, 50, 69, 104, 131, 176, 608, 1184, 4640, 9248, 18464, 36896, 37440, 40960, 69632, 73760, 131085,
    133120, 147488, 150529, 262166, 262211, 262656, 299526, 307202, 393227, 409601, 524316, 524358, 524416,
    524421, 589832, 589837, 589846, 589900, 589972, 590116, 590404, 590980, 592132, 594436, 599108, 608260,
    626692, 655381, 720897, 737284, 786470, 819202, 819207, 28, 42, 49, 70, 82, 100, 133, 208, 259, 352, 1216,
    2368, 9280, 18496, 36928, 49152, 73792, 131093, 131107, 135168, 147520, 262170, 262275, 263168, 294919,
    294944, 299530, 315394, 393235, 430595, 524332, 524422, 524544, 524549, 589838, 589840, 589845, 655397,
    720903, 729092, 786502, 884740, 893958, 81, 134, 137, 164, 200, 261, 416, 515, 704, 2432, 4736, 18560, 18720,
    36992, 73856, 81920, 131097, 131139, 139264, 147584, 153601, 262186, 262214, 262403, 264192, 294923, 294950,
    294976, 294986, 295058, 295202, 295490, 296066, 297218, 299538, 304130, 313346, 368642, 393251, 442370,
    524340, 524550, 524800, 524805, 589861, 599300, 655429, 671745, 747525, 786566, 851982, 983047, 97, 138, 262,
    265, 274, 328, 400, 517, 832, 1027, 1408, 4864, 9472, 37120, 73984, 98304, 131113, 131141, 131203, 147712,
    149763, 157697, 262194, 262278, 262659, 266240, 294931, 295040, 299554, 364546, 393283, 524372, 524428,
    524806, 525312, 525317, 589862, 589888, 589893, 599188, 599556, 655493, 671751, 674053, 720909, 746501,
    786694, 819214, 851990, 860164, 889350, 950279, 56, 84, 98, 140, 161, 266, 273, 518, 521, 530, 548, 656, 800,
    1029, 1664, 2051, 2816, 9360, 9728, 18944, 74240, 131121, 131205, 131331, 147463, 147475, 147493, 147529,
    147601, 147745, 147968, 148033, 148609, 149765, 152065, 156673, 163840, 184321, 221185, 262226, 262282,
    262406, 263171, 270336, 294926, 294947, 295168, 299009, 393347, 411907, 425991, 430082, 524388, 524556,
    525318, 526336, 526341, 589852, 589894, 589952, 589957, 598017, 655621, 720917, 786950, 819222, 852006,
    884738, 884743, 933895, 162, 193, 268, 522, 529, 1030, 1033, 1042, 1060, 1096, 1312, 1600, 2053, 3328, 4099,
```

```
    5632, 19456, 37888, 74753, 131153, 131209, 131333, 131587, 147467, 148480, 149506, 149769, 182273, 196608,
    262242, 262410, 262662, 264195, 278528, 294934, 294979, 295424, 299650, 393475, 409607, 425995, 442369,
    448515, 524452, 524564, 524812, 526342, 528384, 528389, 589868, 589958, 590080, 590085, 598018, 598023,
    598048, 601092, 655877, 671757, 673796, 720933, 787462, 819238, 823300, 852038, 194, 276, 321, 524, 545,
    1034, 1041, 2054, 2057, 2066, 2084, 2120, 2192, 2624, 3200, 4101, 4680, 6656, 8195, 11264, 37377, 38912,
    74754, 75776, 131169, 131337, 131589, 132099, 147469, 149508, 149777, 215041, 262306, 262418, 262666, 263174,
    266243, 294938, 295043, 295936, 299012, 299024, 299778, 393731, 409611, 411650, 426003, 524484, 524820,
    525324, 528390, 532480, 532485, 589876, 590086, 590336, 590341, 603140, 656389, 671765, 720965, 737281,
    788486, 819270, 852102, 860162, 860167, 112, 168, 196, 280, 322, 385, 532, 546, 1036, 1057, 2058, 2065, 4102,
    4105, 4114, 4132, 4168, 4240, 4384, 5248, 6400, 8197, 13312, 16387, 18689, 22528, 37378, 74756, 77824,
    131233, 131345, 131593, 132101, 133123, 147477, 147491, 149512, 151552, 262338, 262674, 263178, 264198,
    270339, 294954, 294982, 295171, 296960, 299016, 327680, 336898, 394243, 409619, 426019, 430081, 446979,
    524612, 524836, 525332, 526348, 532486, 540672, 540677, 561668, 589908, 589964, 590342, 590848, 590853,
    598024, 598029, 598038, 598092, 598164, 598308, 598596, 599168, 599173, 600324, 602628, 616452, 634884,
    657413, 671781, 721029, 729089, 737287, 739589, 745476, 790534, 819334, 852230, 884750, 1015815, 386, 1089,
    4106, 8228, 8264, 8336, 10496, 131601, 147481, 147523, 149825, 155648, 262466, 263186, 295046, 300546,
    393216, 426051, 590092, 599174, 615428, 729095, 823303 };

static int cbest_21[1023] = {
    1, 2, 1048578, 4, 524289, 8, 16, 1310722, 32, 64, 655361, 128, 256, 512, 1376258, 1024, 2048, 4096, 688129,
    8192, 16384, 32768, 65536, 1392642, 131072, 262144, 524288, 696321, 1048576, 3, 1048579, 5, 10, 20, 40, 80,
    160, 320, 640, 1280, 2560, 5120, 10240, 20480, 40960, 81920, 163840, 327680, 655360, 6, 9, 524291, 1048582,
    1310720, 1572867, 17, 524293, 1048586, 1310723, 1396738, 12, 18, 33, 524297, 1048594, 34, 65, 524305,
    1048610, 1310726, 1835011, 24, 36, 66, 129, 524321, 655363, 1048642, 1310730, 1703939, 68, 130, 257, 524353,
    655365, 688128, 1048706, 1310738, 1376256, 48, 72, 132, 258, 513, 344064, 524417, 655369, 698369, 1048834,
    1310754, 1376259, 136, 260, 514, 1025, 172032, 524545, 655377, 1049090, 1310786, 1966083, 96, 144, 264, 516,
    1026, 2049, 86016, 524801, 655393, 1049602, 1310850, 1376262, 1900547, 272, 520, 1028, 2050, 4097, 43008,
    525313, 655425, 1050626, 1310978, 1376266, 192, 288, 528, 1032, 2052, 4098, 8193, 21504, 526337, 655489,
    688131, 1052674, 1311234, 1376274, 1736707, 544, 1040, 2056, 4100, 8194, 10752, 16385, 528385, 655617,
    688133, 1056770, 1311746, 1376290, 384, 576, 1056, 2064, 4104, 5376, 8196, 16386, 32769, 532481, 655873,
    688137, 1064962, 1312770, 1376322, 1392640, 1397762, 2031619, 1088, 2080, 2688, 4112, 8200, 16388, 32770,
    65537, 540673, 656385, 688145, 696320, 1081346, 1314818, 1376386, 1392643, 1998851, 768, 1152, 1344, 2112,
    4128, 8208, 16392, 32772, 65538, 131073, 557057, 657409, 688161, 1114114, 1318914, 1376514, 672, 2176, 4160,
    8224, 16400, 32776, 65540, 131074, 262145, 348160, 589825, 659457, 688193, 1179650, 1327106, 1376770,
    1392646, 1916931, 336, 1536, 2304, 4224, 8256, 16416, 32784, 65544, 131076, 262146, 663553, 688257, 1343490,
    1377282, 1392650, 168, 4352, 8320, 16448, 32800, 65552, 131080, 174080, 262148, 524290, 671745, 688385,
    786433, 1378306, 1392658, 1572866, 84, 3072, 4608, 8448, 16512, 32832, 65568, 131088, 262152, 524292, 688641,
    696323, 698881, 1048577, 1380354, 1392674, 1441794, 1744899, 2064387, 42, 8704, 16640, 32896, 65600, 87040,
    131104, 262160, 524296, 689153, 696325, 720897, 1048584, 1384450, 1392706, 2048003, 11, 21, 41, 81, 161, 321,
    641, 1281, 2561, 5121, 6144, 9216, 10241, 16896, 20481, 33024, 40961, 65664, 81921, 131136, 163841, 262176,
    327681, 524304, 690177, 696329, 1048580, 1392770, 1572865, 1835010, 7, 22, 82, 162, 322, 642, 1282, 2562,
    5122, 10242, 17408, 20482, 33280, 40962, 43520, 65792, 81922, 131200, 163842, 262208, 327682, 524320, 655362,
    692225, 696337, 917505, 1048583, 1048598, 1048618, 1048658, 1048738, 1048898, 1049218, 1049858, 1051138,
    1053698, 1058818, 1069058, 1089538, 1130498, 1212418, 1310721, 1392898, 1396736, 1409026, 1703938, 2007043,
    14, 44, 164, 324, 644, 1284, 2564, 5124, 10244, 12288, 18432, 20484, 33792, 40964, 66048, 81924, 131328,
    163844, 262272, 327684, 524299, 524309, 524329, 524352, 524369, 524449, 524609, 524929, 525569, 526849,
    529409, 534529, 544769, 565249, 606209, 655364, 696353, 851969, 1048587, 1048592, 1310728, 1393154, 1396739,
    13, 19, 28, 88, 328, 648, 1288, 2568, 5128, 10248, 20488, 21760, 34816, 40968, 66560, 81928, 131584, 163848,
    262400, 327688, 524295, 524416, 655368, 696385, 698368, 704513, 1048595, 1048608, 1310724, 1393666, 1507330,
    1572871, 1835009, 26, 35, 56, 176, 656, 1296, 2576, 5136, 10256, 20496, 24576, 36864, 40976, 67584, 81936,
    132096, 163856, 262656, 327696, 524544, 655376, 696449, 1048590, 1048611, 1048640, 1310727, 1310742, 1310762,
    1310802, 1310882, 1311042, 1311362, 1312002, 1313282, 1315842, 1320962, 1331202, 1351682, 1394690, 1396742,
    1474562, 1572875, 1703937, 1921027, 1966082, 25, 37, 52, 67, 112, 352, 1312, 2592, 5152, 10272, 10880, 20512,
    40992, 69632, 81952, 133120, 163872, 263168, 327712, 524301, 524307, 524800, 655392, 696577, 753665, 1048643,
    1048704, 1310731, 1310736, 1396746, 1398018, 1572883, 1900546, 2080771, 38, 69, 74, 104, 131, 224, 704, 2624,
    5184, 10304, 20544, 41024, 49152, 73728, 81984, 135168, 163904, 264192, 327744, 349184, 524323, 525312,
    655371, 655381, 655401, 655424, 655441, 655521, 655681, 656001, 656641, 657921, 660481, 665601, 675841,
    696833, 737281, 983041, 1048602, 1048614, 1048707, 1048832, 1310739, 1310752, 1376257, 1396754, 1400834,
    1572899, 1835015, 2072579, 49, 70, 73, 133, 138, 148, 208, 259, 448, 1408, 5248, 5440, 10368, 20608, 41088,
    82048, 139264, 163968, 266240, 327808, 344065, 524313, 524325, 524355, 526336, 655367, 655488, 688130,
```

src/cauchy_best_r6.c lines 1021 to 1080

```
    697345, 950273, 1048646, 1048835, 1049088, 1310734, 1310755, 1310784, 1376264, 1396770, 1572931, 1703943,
    1736706, 1835019, 1966081, 50, 134, 137, 261, 266, 276, 296, 416, 515, 896, 2816, 10496, 20736, 41216, 82176,
    98304, 147456, 164096, 172033, 270336, 327936, 344066, 524357, 524419, 528384, 655616, 688132, 698371,
    1048626, 1048650, 1048710, 1049091, 1049600, 1310787, 1310848, 1376260, 1396802, 1425410, 1572995, 1703947,
    1746947, 1835027, 1900545, 2052099, 76, 97, 145, 262, 265, 517, 522, 532, 552, 592, 832, 1027, 1792, 2720,
    5632, 20992, 41472, 82432, 86017, 164352, 172034, 174592, 278528, 328192, 344068, 524337, 524361, 524421,
    524547, 532480, 655373, 655379, 655872, 688136, 698373, 700417, 868353, 1048714, 1048838, 1049603, 1050624,
    1220610, 1310746, 1310758, 1310851, 1310976, 1376263, 1376278, 1376298, 1376338, 1376418, 1376578, 1376898,
    1377538, 1378818, 1381378, 1386498, 1396866, 1417218, 1540098, 1573123, 1703955, 1835043, 2031618, 98, 140,
    146, 273, 518, 521, 1029, 1034, 1044, 1064, 1104, 1184, 1664, 2051, 3584, 11264, 41984, 43009, 82944, 86018,
    164864, 172036, 196608, 294912, 328704, 344072, 524425, 524549, 524803, 540672, 655395, 656384, 688144,
    698377, 1048674, 1048722, 1048842, 1049094, 1050627, 1052672, 1134594, 1310790, 1310979, 1311232, 1376267,
    1376272, 1396994, 1523714, 1573379, 1703971, 1736705, 1835075, 1966087, 1998850, 100, 193, 268, 274, 289,
    529, 1030, 1033, 1360, 2053, 2058, 2068, 2088, 2128, 2208, 2368, 3328, 4099, 7168, 21505, 22528, 43010,
    83968, 86020, 165888, 172040, 329728, 344080, 524385, 524433, 524553, 524805, 525315, 557056, 610305, 655385,
    655397, 655427, 657408, 688160, 698385, 712705, 1048850, 1049098, 1049606, 1052675, 1056768, 1091586,
    1310770, 1310794, 1310854, 1311235, 1311744, 1376275, 1376288, 1397250, 1573891, 1704003, 1835139, 1900551,
    1966091, 2009091, 152, 194, 290, 524, 530, 545, 1041, 2054, 2057, 4101, 4106, 4116, 4136, 4176, 4256, 4416,
    4736, 6656, 8195, 10753, 14336, 21506, 43012, 45056, 86024, 87296, 167936, 172048, 331776, 344096, 393216,
    524561, 524809, 525317, 526339, 567297, 589824, 655429, 655491, 659456, 688139, 688149, 688169, 688192,
    688209, 688289, 688449, 688769, 689409, 690689, 693249, 698401, 699009, 708609, 770049, 1015809, 1048770,
    1048866, 1049106, 1049610, 1050630, 1056771, 1064960, 1070082, 1310858, 1310982, 1311747, 1312768, 1376270,
    1376291, 1376320, 1397760, 1482754, 1574915, 1704067, 1835267, 1900555, 1916930, 1966099, 2031617, 196, 280,
    292, 385, 546, 577, 680, 1036, 1042, 1057, 2065, 4102, 4105, 5377, 8197, 8202, 8212, 8232, 8272, 8352, 8512,
    8832, 9472, 10754, 13312, 16387, 21508, 28672, 43016, 86032, 90112, 172064, 335872, 344128, 524481, 524577,
    524817, 525321, 526341, 528387, 545793, 655409, 655433, 655493, 655619, 663552, 688135, 688256, 698433,
    761857, 999425, 1049122, 1049618, 1050634, 1052678, 1059330, 1064963, 1081344, 1310818, 1310866, 1310986,
    1311238, 1312771, 1314816, 1376323, 1376384, 1392641, 1397763, 1398786, 1576963, 1704195, 1736711, 1835523,
    1900563, 1966115, 1998849, 2088963, 200, 386, 548, 1058, 16394, 16424, 16544, 17664, 18944, 57344, 180224,
    535041, 671744, 698497, 1052682, 1056774, 1353730, 1392648, 1836035 };

static int cbest_22[1023] = {
    1, 2, 2097153, 4, 8, 3145729, 16, 32, 64, 3670017, 128, 256, 512, 1024, 3932161, 2048, 4096, 8192, 16384,
    32768, 4063233, 65536, 131072, 262144, 524288, 1048576, 2097152, 4128769, 3, 6, 12, 24, 48, 96, 192, 384,
    768, 1536, 3072, 6144, 12288, 24576, 49152, 98304, 196608, 393216, 786432, 1572864, 3145728, 5, 2097155, 9,
    2097157, 10, 17, 2097161, 3145731, 3670016, 18, 33, 1835008, 2097169, 3145733, 20, 34, 65, 917504, 2097185,
    3145737, 4161537, 36, 66, 129, 458752, 2097217, 3145745, 3670019, 40, 68, 130, 257, 229376, 2097281, 3145761,
    3670021, 3932160, 72, 132, 258, 513, 114688, 2097409, 3145793, 3670025, 80, 136, 260, 514, 1025, 57344,
    1966080, 2097665, 3145857, 3670033, 144, 264, 516, 1026, 2049, 28672, 2098177, 3145985, 3670049, 3932163,
    160, 272, 520, 1028, 2050, 4097, 14336, 983040, 2099201, 3146241, 3670081, 3932165, 288, 528, 1032, 2052,
    4098, 7168, 8193, 2101249, 3146753, 3670145, 3932169, 4063232, 320, 544, 1040, 2056, 3584, 4100, 8194, 16385,
    491520, 2105345, 3147777, 3670273, 3932177, 4177921, 576, 1056, 1792, 2064, 4104, 8196, 16386, 32769,
    2113537, 3149825, 3670529, 3932193, 640, 896, 1088, 2080, 4112, 8200, 16388, 32770, 65537, 245760, 2031616,
    2129921, 3153921, 3671041, 3932225, 4063235, 448, 1152, 2112, 4128, 8208, 16392, 32772, 65538, 131073,
    2162689, 3162113, 3672065, 3932289, 4063237, 224, 1280, 2176, 4160, 8224, 16400, 32776, 65540, 122880,
    131074, 262145, 2228225, 3178497, 3674113, 3932417, 4063241, 112, 2304, 4224, 8256, 16416, 32784, 65544,
    131076, 262146, 524289, 1015808, 2359297, 3211265, 3678209, 3932673, 4063249, 4128768, 56, 2560, 4352, 8320,
    16448, 32800, 61440, 65552, 131080, 262148, 524290, 1048577, 2621441, 3276801, 3686401, 3933185, 4063265, 28,
    4608, 8448, 16512, 32832, 65568, 131088, 262152, 524292, 1048578, 3407873, 3702785, 3934209, 4063297, 14,
    5120, 8704, 16640, 30720, 32896, 65600, 131104, 262160, 507904, 524296, 1048580, 2097154, 3735553, 3936257,
    4063361, 4128771, 7, 13, 25, 49, 97, 193, 385, 769, 1537, 3073, 6145, 9216, 12289, 16896, 24577, 33024,
    49153, 65664, 98305, 131136, 196609, 262176, 393217, 524304, 786433, 1048584, 1572865, 2064384, 2097156,
    3801089, 3940353, 4063489, 4128773, 4186113, 26, 50, 98, 194, 386, 770, 1538, 3074, 6146, 10240, 12290,
    15360, 17408, 24578, 33280, 49154, 65792, 98306, 131200, 196610, 262208, 393218, 524320, 786434, 1048592,
    1572866, 2097159, 2097160, 2097165, 2097177, 2097201, 2097249, 2097345, 2097537, 2097921, 2098689, 2100225,
    2103297, 2109441, 2121729, 2146305, 2195457, 2293761, 2490369, 2883585, 3145730, 3948545, 4063745, 4128777,
    52, 100, 196, 388, 772, 1540, 3076, 6148, 12292, 18432, 24580, 33792, 49156, 66048, 98308, 131328, 196612,
    253952, 262272, 393220, 524352, 786436, 1048608, 1572868, 2097168, 3145732, 3964929, 4064257, 4128785, 11,
    104, 200, 392, 776, 1544, 3080, 6152, 7680, 12296, 20480, 24584, 34816, 49160, 66560, 98312, 131584, 196616,
    262400, 393224, 524416, 786440, 1048640, 1572872, 2097184, 3145735, 3145736, 3145741, 3145753, 3145777,
```

src/cauchy_best_r6.c lines 1081 to 1140

```
    3145825, 3145921, 3146113, 3146497, 3147265, 3148801, 3151873, 3158017, 3170305, 3194881, 3244033, 3342337,
    3538945, 3997697, 4065281, 4128801, 4161536, 19, 22, 208, 400, 784, 1552, 3088, 6160, 12304, 24592, 36864,
    49168, 67584, 98320, 132096, 196624, 262656, 393232, 524544, 786448, 1032192, 1048704, 1572880, 1835009,
    2097163, 2097216, 3145744, 3670018, 4067329, 4128833, 21, 35, 38, 44, 416, 800, 1568, 3104, 3840, 6176,
    12320, 24608, 40960, 49184, 69632, 98336, 126976, 133120, 196640, 263168, 393248, 524800, 786464, 917505,
    1048832, 1572896, 1835010, 2097171, 2097280, 3145760, 3670020, 4071425, 4128897, 37, 67, 70, 76, 88, 832,
    1600, 3136, 6208, 12352, 24640, 49216, 73728, 98368, 135168, 196672, 264192, 393280, 458753, 525312, 786496,
    917506, 1049088, 1572928, 1835012, 2097173, 2097187, 2097408, 3014657, 3145739, 3145792, 3670023, 3670024,
    3670029, 3670041, 3670065, 3670113, 3670209, 3670401, 3670785, 3671553, 3673089, 3676161, 3682305, 3694593,
    3719169, 3768321, 3866625, 4079617, 4129025, 4161539, 41, 69, 131, 134, 140, 152, 176, 1664, 1920, 3200,
    6272, 12416, 24704, 49280, 81920, 98432, 139264, 196736, 229377, 266240, 393344, 458754, 526336, 786560,
    917508, 1049600, 1572992, 1835016, 2097189, 2097219, 2097664, 2555905, 3145747, 3145856, 3670032, 4096001,
    4129281, 4161541, 42, 73, 133, 259, 262, 268, 280, 304, 352, 3328, 6400, 12544, 24832, 49408, 63488, 98560,
    114689, 147456, 196864, 229378, 270336, 393472, 458756, 516096, 528384, 786688, 917512, 1050624, 1573120,
    1835024, 2080768, 2097193, 2097221, 2097283, 2098176, 2326529, 3145749, 3145763, 3145984, 3670048, 3932162,
    4129793, 4161545, 74, 81, 137, 261, 515, 518, 524, 536, 560, 608, 704, 960, 6656, 12800, 25088, 49664, 57345,
    98816, 114690, 163840, 197120, 229380, 278528, 393728, 458760, 532480, 786944, 917520, 1052672, 1573376,
    1835040, 1966081, 2097225, 2097285, 2097411, 2099200, 2211841, 3145765, 3145795, 3146240, 3604481, 3670027,
    3670080, 3932164, 4130817, 4161553, 82, 138, 145, 265, 517, 1027, 1030, 1036, 1048, 1072, 1120, 1216, 1408,
    13312, 25600, 28673, 50176, 57346, 99328, 114692, 197632, 229384, 294912, 394240, 458768, 540672, 787456,
    917536, 1056768, 1573888, 1835072, 1966082, 2097233, 2097289, 2097413, 2097667, 2101248, 2154497, 3145769,
    3145797, 3145859, 3146752, 3375105, 3670035, 3670144, 3932167, 3932168, 3932173, 3932185, 3932209, 3932257,
    3932353, 3932545, 3932929, 3933697, 3935233, 3938305, 3944449, 3956737, 3981313, 4030465, 4132865, 4161569,
    4190209, 84, 146, 161, 266, 273, 480, 521, 1029, 2051, 2054, 2060, 2072, 2096, 2144, 2240, 2432, 2816, 14337,
    26624, 28674, 31744, 51200, 57348, 100352, 114696, 198656, 229392, 327680, 395264, 458784, 557056, 788480,
    917568, 983041, 1064960, 1574912, 1835136, 1966084, 2097297, 2097417, 2097669, 2098179, 2105344, 2125825,
    3145801, 3145861, 3145987, 3147776, 3260417, 3670037, 3670051, 3670272, 3932176, 4136961, 4161601, 4177920,
    148, 162, 274, 289, 522, 529, 1033, 2053, 4099, 4102, 4108, 4120, 4144, 4192, 4288, 4480, 4864, 5632, 7169,
    14338, 28676, 53248, 57352, 102400, 114704, 200704, 229408, 258048, 397312, 458816, 589824, 790528, 917632,
    983042, 1081344, 1576960, 1835264, 1966088, 2097313, 2097425, 2097673, 2098181, 2099203, 2111489, 2113536,
    3080193, 3145809, 3145865, 3145989, 3146243, 3149824, 3203073, 3670053, 3670083, 3670528, 3932192, 4145153,
    4161665, 164, 240, 276, 290, 321, 530, 545, 1034, 1041, 2057, 3585, 4101, 7170, 8195, 8198, 8204, 8216, 8240,
    8288, 8384, 8576, 8960, 9728, 11264, 14340, 28680, 57360, 106496, 114720, 204800, 229440, 401408, 458880,
    491521, 655360, 794624, 917760, 983044, 1040384, 1114112, 1581056, 1835520, 1966096, 2097441, 2097681,
    2098185, 2099205, 2101251, 2104321, 2129920, 3145873, 3145993, 3146245, 3146755, 3153920, 3174401, 3670057,
    3670085, 3670147, 3671040, 3899393, 3932171, 3932224, 4063234, 4161793, 168, 292, 322, 532, 546, 577, 1042,
    1057, 1793, 2058, 2065, 3586, 4105, 7172, 8197, 14344, 15872, 16387, 16390, 16396, 16408, 16432, 16480,
    16576, 16768, 17152, 17920, 19456, 22528, 28688, 57376, 114752, 212992, 229504, 409600, 459008, 491522,
    802816, 918016, 983048, 1179648, 1589248, 1836032, 1966112, 2097473, 2097697, 2098193, 2099209, 2100737,
    2101253, 2105347, 2162688, 2588673, 3145889, 3146001, 3146249, 3146757, 3147779, 3160065, 3162112, 3670089,
    3670149, 3670275, 3672064, 3784705, 3932179, 3932288, 4063236, 4162049, 4177923, 120, 296, 897, 1058, 1089,
    2081, 4106, 4113, 8201, 32774, 32780, 33536, 34304, 45056, 491524, 983056, 1605632, 1966144, 2031617,
    2098209, 2098945, 2099217, 2228224, 3146257, 3146761, 3178496, 3670153, 3670277, 3670531, 3932181, 3932416,
    4063329, 4063425, 4069377, 4075521 };

static int cbest_23[1023] = {
    1, 2, 4194320, 4, 2097160, 8, 1048580, 16, 524290, 32, 262145, 64, 128, 4325392, 256, 512, 2162696, 1024,
    2048, 1081348, 4096, 8192, 540674, 16384, 32768, 270337, 65536, 131072, 262144, 4329488, 524288, 2164744,
    1048576, 3, 1082372, 2097152, 4194321, 5, 2097161, 4194322, 6, 9, 1048581, 2097162, 4194304, 4194324,
    6291480, 10, 17, 524291, 541186, 1048582, 2097164, 4194328, 5242900, 12, 18, 33, 66, 132, 264, 528, 1056,
    2112, 4224, 8448, 16896, 33792, 67584, 135168, 270336, 3145740, 4718610, 20, 34, 65, 262147, 524294, 540672,
    1048588, 2097176, 2621450, 4194352, 4456465, 24, 36, 129, 262149, 270593, 524298, 1048596, 1081344, 1572870,
    2097192, 2359305, 4194384, 4325393, 40, 68, 130, 257, 262153, 524306, 1048612, 1310725, 2097224, 2162688,
    4194448, 4325394, 48, 72, 258, 513, 262161, 524322, 786435, 1048644, 2097288, 2162697, 4194576, 4325376,
    4325396, 6422552, 80, 136, 260, 514, 1025, 262177, 524354, 1048708, 2097416, 2162698, 4194832, 4325400,
    5373972, 6357016, 96, 144, 516, 1026, 2049, 262209, 524418, 1048836, 1081349, 2097672, 2162700, 4195344,
    4329616, 4849682, 160, 272, 520, 1028, 2050, 4097, 262273, 524546, 1049092, 1081350, 2098184, 3211276,
    4196368, 4325424, 4587537, 5275668, 192, 288, 1032, 2052, 4098, 8193, 262401, 524802, 540675, 1049604,
    2099208, 2162712, 2686986, 3178508, 4198416, 4325456, 320, 544, 1040, 2056, 4100, 8194, 16385, 262657,
    525314, 1050628, 1081356, 2101256, 2162728, 2424841, 4202512, 4325520, 4734994, 384, 576, 2064, 4104, 8196,
```

```
    16386, 32769, 263169, 526338, 540678, 1052676, 1081364, 1605638, 2105352, 2162760, 2164808, 2637834, 4210704,
    4325648, 640, 1088, 2080, 4112, 8200, 16388, 32770, 65537, 264193, 270339, 528386, 540682, 1056772, 1081380,
    1343493, 1589254, 2113544, 2162824, 4227088, 4325904, 4464657, 6488088, 768, 1152, 4128, 8208, 16392, 32772,
    65538, 131073, 266241, 270341, 532482, 540690, 1064964, 1081412, 2129928, 2162952, 2367497, 4259856, 4326416,
    1280, 2176, 4160, 8224, 16400, 32776, 65540, 131074, 270345, 540706, 802819, 1081476, 1318917, 2163208,
    4327440, 4329489, 5406740, 1536, 2304, 8256, 16416, 32784, 65544, 131076, 262146, 270353, 278529, 540738,
    557058, 794627, 1081604, 1082404, 1114116, 2163720, 2228232, 4329490, 4456464, 2560, 4352, 8320, 16448,
    32800, 65552, 131080, 262148, 270369, 270592, 294913, 524289, 540802, 541184, 589826, 1081860, 1082368,
    1179652, 2164736, 2359304, 3244044, 4329472, 4329492, 4333584, 4866066, 6426648, 3072, 4608, 16512, 32832,
    65568, 131088, 135296, 262152, 270401, 327681, 540930, 655362, 1310724, 2164745, 2166792, 4329496, 4341776,
    4718608, 5378068, 5120, 8704, 16640, 32896, 65600, 67648, 131104, 262160, 270465, 393217, 524292, 786434,
    1048577, 1083396, 2164746, 2170888, 2621448, 2703370, 4358160, 4595729, 4853778, 6359064, 6144, 9216, 33024,
    33824, 65664, 131136, 262176, 524296, 541202, 541698, 1048578, 1085444, 1572868, 2164748, 2179080, 4329520,
    4390928, 4591633, 5242896, 10240, 16912, 17408, 33280, 65792, 131200, 262208, 270849, 524304, 524352, 542722,
    1082373, 1089540, 1622022, 2097153, 2195464, 2433033, 3145736, 3213324, 4329552, 8456, 12288, 18432, 66048,
    131328, 262272, 271361, 524320, 544770, 786433, 1048584, 1048704, 1082374, 1097732, 2097154, 2164760,
    2689034, 4194323, 4587536, 5276692, 6291472, 7, 4228, 20480, 34816, 66560, 131584, 262400, 272385, 548866,
    1048592, 1351685, 1572866, 2097156, 2097163, 2097408, 2164776, 2293768, 2426889, 3179532, 4194305, 4194325,
    4329744, 6291481, 11, 2114, 24576, 36864, 132096, 262656, 270601, 274433, 524416, 541187, 1048583, 1048608,
    1082380, 1146884, 1310721, 2097165, 2424840, 3145732, 4194306, 4194326, 4194329, 4194816, 4330000, 4849680,
    5242901, 6291482, 6492184, 13, 19, 67, 133, 265, 529, 1057, 2113, 4225, 8449, 16897, 33793, 40960, 67585,
    69632, 133120, 135169, 263168, 524544, 573442, 811011, 1048640, 1082388, 1212420, 1606662, 2097166, 2097168,
    2164872, 2621442, 3145741, 4194308, 4194330, 4330512, 4718611, 4735506, 5242902, 6291464, 6291484, 6490136,
    14, 21, 35, 134, 266, 530, 1058, 4226, 8450, 16898, 33794, 49152, 67586, 73728, 135170, 264192, 270338,
    286721, 524295, 524800, 540673, 541190, 606210, 1048589, 1343492, 1344517, 2097177, 2097184, 2165000,
    2359297, 2621451, 2638346, 2686984, 3145742, 4194312, 4194332, 4194353, 4194386, 4194452, 4194584, 4195376,
    4196432, 4198544, 4202768, 4211216, 4228112, 4261904, 4331536, 4464656, 5242884, 5373968, 5410836, 7340060,
    22, 25, 37, 70, 268, 532, 1060, 2116, 8452, 16900, 33796, 67588, 81920, 135172, 139264, 266240, 270340,
    303105, 524299, 525312, 541194, 671746, 1048590, 1048597, 1048832, 1081345, 1082436, 1572871, 1589766,
    2097178, 2097193, 2097216, 2097226, 2097292, 2097688, 2098216, 2099272, 2101384, 2105608, 2114056, 2130952,
    2165256, 2232328, 2367496, 4194354, 4194385, 4456467, 4718594, 4718614, 4734992, 5242908, 6815770, 26, 38,
    41, 69, 74, 131, 140, 536, 1064, 2120, 4232, 16904, 33800, 67592, 98304, 135176, 147456, 262151, 270344,
    270595, 335873, 524307, 526336, 540676, 802818, 1048598, 1048613, 1048646, 1048844, 1049088, 1049108,
    1049636, 1050692, 1052804, 1057028, 1065476, 1081346, 1082500, 1116164, 1183748, 1318916, 1605636, 2097180,
    2097194, 2097225, 2097280, 2162689, 2165768, 2359307, 2621454, 2637832, 3211272, 4194336, 4194356, 4194449,
    4325395, 4337680, 4456449, 4456469, 4464913, 4718618, 4870162, 5275664, 5407764, 5767190, 6291512, 6422544,
    6553625, 28, 42, 49, 73, 82, 148, 259, 280, 1072, 2128, 4240, 8464, 33808, 67600, 135184, 163840, 262155,
    270352, 270597, 278528, 401409, 524302, 524323, 524422, 524554, 524818, 525346, 526402, 528384, 528514,
    532738, 540680, 541218, 558082, 591874, 659458, 794626, 803331, 1048614, 1048645, 1049600, 1082628, 1310727,
    1589252, 2097196, 2097289, 2162690, 2359309, 2367753, 3145756, 3178504, 3246092, 3670030, 4194360, 4194368,
    4194388, 4194450, 4194577, 4325377, 4325397, 4329620, 4345872, 4456473, 5242932, 5505045, 6291544, 6357008,
    6422553, 44, 50, 81, 98, 137, 164, 261, 296, 515, 560, 2144, 4256, 8480, 16928, 67616, 135200, 196608,
    262157, 262163, 262211, 262277, 262409, 262673, 263201, 264257, 266369, 270368, 279041, 294912, 295937,
    329729, 397313, 524310, 524355, 532480, 540688, 540736, 541250, 1048604, 1048709, 1050624, 1081352, 1081472,
    1082884, 1319173, 1572878, 2097228, 2097290, 2097417, 2097664, 2162692, 2162699, 2162944, 2168840, 2621466,
    3145772, 3245068, 3407885, 4194392, 4194432, 4194578, 4194833, 4325378, 4325398, 4325401, 4325888, 4362256,
    4599825, 4718642, 4980755, 5242964, 5373973, 6291608, 6357017, 6422554, 52, 76, 97, 138, 145, 196, 262, 328,
    517, 592, 1027, 1120, 4288, 8512, 16960, 33856, 135232, 262165, 262179, 270400, 270609, 327680, 524314,
    524326, 524419, 540704, 541314, 786439, 794883, 802817, 1048620, 1048710, 1048837, 1052672, 1081360, 1310733,
    1572886, 1605634, 2097208, 2097418, 2097673, 2098176, 2162701, 2172936, 2359321, 2621482, 2705418, 2883595,
    3145804, 3211268, 4194456, 4194560, 4194580, 4194834, 4195345, 4325380, 4325402, 4329617, 4395024, 4456497,
    4718674, 4849683, 4866578, 5243028, 5373974, 6291736, 6357018, 6422536, 6422556, 56, 84, 146, 161, 194, 273,
    392, 518, 521, 656, 1029, 1184, 2051, 2240, 8576, 17024, 33920, 67712, 262169, 262181, 270464, 270625,
    393216, 524330, 524358, 524547, 541442, 557056, 786443, 794625, 1048628, 1048652, 1048838, 1049093, 1056768,
    1081351, 1081376, 1084420, 1310741, 1343489, 1572902, 1589250, 1835015, 2097240, 2097420, 2097674, 2098185,
    2099200, 2162702, 2162704, 2181128, 2359337, 2621514, 2686978, 3145868, 3178500, 3211277, 4194416, 4194836,
    4195346, 4196369, 4325384, 4325404, 4325425, 4325458, 4325524, 4325656, 4326448, 4327504, 4329618, 4333840,
    4342288, 4359184, 4392976, 4456529, 4595728, 4718738, 5243156, 5275669, 5373956, 6291992, 6357000, 6357020,
    7471132, 162, 193, 289, 1030, 1312, 2053, 4099, 34048, 262185, 262213, 524338, 524362, 524803, 786451,
    1048716, 1049605, 1064960, 1086468, 1310757, 2097256, 2097304, 2097676, 2098186, 2099209, 2101248, 2162713,
```

```
    2424833, 2621578, 2703882, 3145996, 3178509, 4194480, 4198417, 4325426, 4456593, 4587539, 4718866, 4849666,
    6292504, 6426776, 7405596 };

static int cbest_24[1023] = {
    1, 2, 8388675, 4, 12583010, 8, 6291505, 16, 32, 64, 11534427, 128, 256, 8388674, 4194337, 14155886, 3, 512,
    8388673, 6, 12583011, 5, 1024, 10485843, 12, 6291504, 12583008, 9, 24, 2048, 3145752, 8388679, 10, 7077943,
    14680178, 48, 1572876, 13631594, 17, 4096, 8388683, 96, 786438, 12583014, 20, 6291507, 18, 33, 192, 8192,
    393219, 7340089, 8388691, 6815797, 11534426, 12583018, 384, 131073, 6291489, 6291509, 11927640, 34, 40, 65,
    16384, 262146, 8388707, 8585282, 12582978, 66, 768, 524292, 8388611, 8454211, 11534419, 11534425, 12583026,
    1048584, 6291513, 36, 129, 1536, 32768, 2097168, 4292641, 5767213, 80, 130, 4194336, 6291473, 11796569,
    12615778, 14155882, 132, 3072, 12058719, 12582946, 68, 257, 65536, 5963820, 8388803, 11534411, 11534431,
    14155887, 258, 6144, 196609, 393218, 6307889, 7077941, 160, 260, 8388672, 72, 264, 513, 12288, 131072,
    8388931, 10534995, 11272277, 14155878, 14155884, 786436, 8486979, 10485842, 12583138, 7, 516, 24576, 4194339,
    6291569, 11927641, 136, 320, 514, 520, 1025, 393217, 5242921, 7077939, 7077942, 8388677, 8389187, 8585283,
    12583009, 14418028, 13, 528, 49152, 1572872, 2981910, 8388678, 10485841, 11542619, 12583266, 14286959,
    15466612, 14, 4194341, 6291633, 12632162, 14680179, 25, 144, 1026, 1032, 2049, 98304, 262144, 786434,
    3145753, 8389699, 11534459, 13631595, 11, 26, 640, 1040, 3145744, 6291506, 8388682, 8388687, 11534363,
    11927642, 12583012, 12583522, 12681314, 13656170, 14024809, 28, 49, 1056, 196608, 1572877, 4194345, 6291761,
    7340088, 8388681, 11010135, 12583015, 14680176, 272, 1028, 2050, 4097, 1572868, 3145754, 3538971, 6291457,
    6316081, 6815793, 6815796, 8388699, 8390723, 10485827, 10485847, 13631586, 13631592, 22, 52, 97, 2064,
    786439, 4194305, 6291488, 6291508, 7209014, 8388690, 11534403, 12584034, 14155854, 21, 50, 56, 98, 1280,
    2080, 1490955, 1572878, 3670044, 4194353, 5963821, 6291497, 6292017, 6340657, 7733306, 8388643, 8388723,
    8781888, 9175109, 9961551, 12582914, 12582994, 12583019, 12583022, 14155902, 14159982, 15728762, 19, 193,
    288, 2052, 2112, 4098, 8193, 524288, 3145736, 3145756, 3407898, 4390944, 8392771, 10485851, 11534555,
    11796571, 12583016, 13893736, 14155874, 14352495, 15204470, 15532148, 44, 104, 393216, 2195472, 6291511,
    6291512, 7077927, 7340081, 8388689, 8388706, 8388739, 12582979, 12583034, 12585058, 14680162, 14680182, 112,
    196, 385, 4128, 1097736, 1835022, 4227105, 4292640, 5767209, 5767212, 6292529, 6828085, 7077937, 8388610,
    8388695, 8454210, 11534418, 11534423, 11534424, 12583027, 13631598, 35, 41, 194, 544, 2056, 2560, 4100, 4160,
    8194, 16385, 262147, 548868, 1703949, 2146320, 4259873, 6291472, 6291517, 7340091, 8388609, 8396867, 8519746,
    11534417, 11534683, 11927632, 12976225, 13369444, 14155822, 42, 67, 88, 100, 208, 769, 4224, 274434, 524293,
    1073160, 6291493, 6815799, 6946868, 7077951, 8388705, 8585280, 8650817, 10158158, 11546715, 12058715,
    12582976, 12587106, 14155880, 14286958, 14680186, 38, 134, 224, 386, 392, 770, 131075, 137217, 536580,
    786432, 917511, 1048585, 2883606, 3604507, 4194401, 5963816, 6291491, 6291537, 6293553, 7864381, 8389059,
    8454209, 10502227, 11272279, 11796568, 12583024, 12583074, 12615779, 14155883, 15401079, 37, 70, 268, 576,
    1537, 4104, 8196, 8256, 16386, 32769, 268290, 393223, 1048576, 2097169, 3866653, 6291515, 6307888, 7077911,
    7077940, 7079991, 7340073, 7340093, 7602235, 8388715, 8405059, 10485875, 10518611, 11534429, 11534939,
    11559003, 12058718, 12582947, 12582986, 12583030, 14680146, 81, 131, 176, 416, 536, 5120, 8320, 134145,
    524294, 786446, 2981908, 3153944, 3538970, 5767205, 6684722, 6815781, 8388802, 8389443, 8847424, 8912967,
    9134150, 9240645, 10485779, 11370581, 11534410, 11534430, 11927643, 12582944, 12591202, 12714083, 13631562,
    14024808, 82, 133, 262, 388, 448, 784, 1072, 1540, 3073, 8448, 131077, 1048586, 1441803, 1572892, 1576972,
    3473434, 4194465, 6295601, 7766074, 8388627, 8388801, 9437259, 11272273, 11534443, 11796561, 11927624,
    11927644, 12615776, 12845152, 13631610, 14156014, 14418030, 69, 84, 140, 200, 524, 1088, 1538, 2144, 4112,
    8200, 16388, 32770, 65537, 262150, 393227, 788486, 1490954, 1572864, 2097170, 3145784, 4292643, 4423712,
    5767215, 5898284, 6422576, 8388711, 8390211, 8421443, 8457283, 8585286, 11535451, 12058711, 12058717,
    12582982, 12583394, 13639786, 14696562, 15466614, 15564916, 76, 259, 352, 772, 832, 1048, 4288, 6145, 16512,
    394243, 786454, 3211288, 4194338, 5963812, 6291475, 6291481, 6291568, 6815805, 8388615, 8388930, 8454215,
    10534994, 11272276, 11534409, 11802713, 12599394, 13647978, 14155879, 14155885, 14876787, 74, 161, 261, 266,
    896, 1568, 2096, 3074, 3080, 8576, 10240, 16640, 131081, 655365, 1048588, 1310730, 1572908, 1605644, 1769485,
    2211856, 2621460, 3342361, 4194593, 5079079, 5242920, 6291697, 6299697, 7077938, 7143479, 7340057, 8388807,
    8388929, 8391747, 8455747, 8486978, 8781889, 11534415, 12583139, 12583778, 13107302, 14155876, 14156142,
    14418024, 14680114, 14745714, 73, 162, 265, 280, 1152, 4192, 8208, 12289, 16392, 16896, 17152, 32772, 65538,
    132097, 262154, 393235, 802822, 1736717, 2097152, 2097172, 2981906, 3145816, 3538969, 4243489, 4292645,
    5963822, 6029359, 6291553, 6815765, 7012404, 7209015, 8585290, 8585794, 8716353, 9109574, 10485971, 11536475,
    11928664, 12320861, 12582962, 12583136, 13008993, 13631530, 14024811, 14155894, 14162030, 14286955, 14352494,
    15466608, 164, 518, 704, 776, 1664, 6146, 8384, 34304, 133121, 196611, 401411, 524300, 786437, 786470,
    1105928, 1703948, 2949142, 3145728, 4194340, 6291477, 6291632, 6291889, 6307891, 6819893, 7078007, 7209012,
    7348281, 7733307, 8388619, 8388676, 8389186, 8394819, 8454219, 8454723, 10190926, 10534993, 11534451,
    11796553, 11796573, 12189790, 12584546, 12615782, 14418029, 138, 168, 400, 517, 532, 1792, 3076, 3136, 6160,
    16768, 24577, 33024, 68608, 131089, 135169, 264194, 745477, 851974, 1490953, 1572940, 4194849, 4292897,
    4390945, 5267497, 5636138, 6553651, 6823989, 7733304, 8388835, 8388935, 8455235, 8486977, 10321997, 10485840,
```

src/cauchy_best_r6.c lines 1261 to 1320

```
    11534395, 11542618, 12058703, 12582950, 12583106, 12583267, 12617314, 14156398, 14168174, 15466613, 16089208,
    137, 152, 321, 515, 521, 560, 1544, 2176, 8224, 12290, 16400, 20480, 32776, 33280, 33536, 65540, 131074,
    137216, 262162, 266242, 393251, 425987, 552964, 917510, 1572874, 2097176, 3145880, 3407896, 3604506, 4292649,
    4567075, 6291571, 6292273, 7372857, 8388939, 8400963, 8456259, 8519747, 8585298, 8589378, 10486099, 11538523,
    11927633, 12583142, 12583264, 12586082, 12616034, 12632163, 12779619, 12976224, 14155898, 148, 522, 529,
    1036, 1408, 3328, 6148, 33792, 49153, 67072, 196613, 270338, 393222, 458755, 524308, 528388, 786502, 1572873,
    2981911, 3866652, 3883037, 4194343, 4194344, 4554787, 5242913, 6291760, 6307873, 6307893, 7077925, 7077947,
    7078071, 8388867, 8389185, 8389698, 8454227, 8585281, 8781890, 9175111, 9273413, 10158159, 10510419,
    11010131, 11534458, 11534491, 12615786, 12616290, 12616546, 12648546, 12746851, 14073961, 14418020, 14680306,
    15, 324, 1064, 3584, 6272, 12320, 24578, 131105, 134144, 163841, 276482, 393221, 532484, 1048600, 1474571,
    1573004, 1835020, 3145748, 3506202, 4195361, 4292609, 4294689, 5767181, 6291456, 6293041, 6308017, 6316080,
    6324273, 6488112, 6815792, 7077923, 7077949, 8388686, 8389191, 8413251, 8601666, 9257029, 10928208, 11403348,
    11534347, 11534362, 11542611, 11542617, 11927672, 11960408, 12582954, 12583013, 12583523, 12589154, 12616802,
    12681315, 13631722, 13656171, 13893738, 14155846, 14156910, 14286951, 14286957, 15401078, 15532150, 145, 322,
    328, 1027, 1033, 1120, 1552, 2304, 12292, 16416, 32784, 65544, 66048, 98305, 131076, 262145, 262178, 393283,
    540676, 786435, 1056776, 1441802, 2818069, 3146008, 4194304, 4194321, 4194349, 4292657, 5242923, 5767229,
    5771309, 5964332, 6291601, 6291635, 6308145, 6946869, 7209010, 7340153, 7782458, 8388680, 8388685, 8388811,
    8389195, 8470595, 8585314, 8618050, 10059855, 10485835, 10485845, 10486355, 10534979, 10534999, 10584147,
    11010134, 11272261, 11534361, 11665498, 11798617, 11927576, 11927634, 11993176, 12583270, 12615746, 12632160,
    14155790, 14155850, 14155870, 14680177, 15827066, 146, 276, 336, 641, 800, 1034, 2816, 6656, 49154, 66560,
    196617, 197121, 524324, 720901, 786442, 786566, 3158040, 3670040, 5963817, 6291637, 6292016, 6294577,
    6307897, 6308657, 6340656, 6815861, 7078199, 8389203, 8390722, 8454243, 8486983, 8585218, 8847425, 9134151,
    10485846, 10535123, 12583274, 12615794, 13631587, 14159978 };

static int cbest_25[1023] = {
    1, 2, 16777220, 4, 8388610, 8, 4194305, 16, 32, 18874372, 64, 128, 9437186, 256, 512, 4718593, 1024, 2048,
    4096, 19136516, 8192, 16384, 32768, 9568258, 65536, 131072, 262144, 4784129, 524288, 1048576, 2097152,
    4194304, 19169284, 8388608, 3, 9584642, 16777216, 16777221, 5, 8388611, 16777222, 6, 9, 18, 36, 72, 144, 288,
    576, 1152, 2304, 4608, 9216, 18432, 36864, 73728, 147456, 294912, 589824, 1179648, 2359296, 4718592,
    25165830, 10, 17, 4194307, 8388614, 9437184, 16777228, 20971525, 12, 33, 4194309, 4792321, 8388618, 12582915,
    16777236, 18874368, 18874373, 20, 34, 65, 4194313, 8388626, 16777252, 18874374, 24, 66, 129, 4194321,
    8388642, 9437187, 16777284, 27262982, 40, 68, 130, 257, 4194337, 8388674, 16777348, 18874380, 23068677,
    26214406, 48, 132, 258, 513, 4194369, 8388738, 9437190, 16777476, 18874388, 80, 136, 260, 514, 1025, 4194433,
    4718595, 8388866, 9437194, 13631491, 16777732, 18874404, 19173380, 21495813, 96, 264, 516, 1026, 2049,
    4194561, 4718597, 8389122, 9437202, 13107203, 16778244, 18874436, 160, 272, 520, 1028, 2050, 4097, 4194817,
    4718601, 4784128, 8389634, 9437218, 9568256, 16779268, 18874500, 19136512, 19136517, 28311558, 192, 528,
    1032, 2052, 4098, 8193, 2392064, 4195329, 4718609, 8390658, 9437250, 16781316, 18874628, 19136518, 320, 544,
    1040, 2056, 4100, 8194, 16385, 1196032, 4196353, 4718625, 8392706, 9437314, 16785412, 18874884, 23592965,
    27525126, 384, 1056, 2064, 4104, 8196, 16386, 32769, 598016, 4198401, 4718657, 8396802, 9437442, 9568259,
    9586690, 16793604, 18875396, 19136524, 23330821, 640, 1088, 2080, 4112, 8200, 16388, 32770, 65537, 299008,
    4202497, 4718721, 8404994, 9437698, 14155779, 16809988, 18876420, 19136532, 26345478, 768, 2112, 4128, 8208,
    16392, 32772, 65538, 131073, 149504, 4210689, 4718849, 8421378, 9438210, 9568262, 16842756, 18878468,
    19136548, 1280, 2176, 4160, 8224, 16400, 32776, 65540, 74752, 131074, 262145, 4227073, 4719105, 8454146,
    9439234, 9568266, 13762563, 16908292, 18882564, 19136580, 1536, 4224, 8256, 16416, 32784, 37376, 65544,
    131076, 262146, 524289, 4259841, 4719617, 4784131, 8519682, 9441282, 9568274, 17039364, 18890756, 19136644,
    21561349, 28573702, 2560, 4352, 8320, 16448, 18688, 32800, 65552, 131080, 262148, 524290, 1048577, 4325377,
    4720641, 4784133, 4793345, 8650754, 9445378, 9568290, 13172739, 17301508, 18907140, 19136772, 28442630, 3072,
    8448, 9344, 16512, 32832, 65568, 131088, 262152, 524292, 1048578, 2097153, 4456449, 4722689, 4784137,
    8912898, 9453570, 9568322, 17825796, 18939908, 19137028, 23855109, 4672, 5120, 8704, 16640, 32896, 65600,
    131104, 262160, 524296, 1048580, 2097154, 4726785, 4784145, 9469954, 9568386, 19005444, 19137540, 19169280,
    19169285, 2336, 6144, 16896, 33024, 65664, 131136, 262176, 524304, 1048584, 2097156, 4194306, 4734977,
    4784161, 5242881, 9502722, 9568514, 9584640, 10485762, 19138564, 19169286, 20971524, 23658501, 1168, 10240,
    17408, 33280, 65792, 131200, 262208, 524320, 1048592, 2097160, 4194308, 4751361, 4784193, 4792320, 6291457,
    8388609, 9568770, 12582914, 14286851, 19140612, 19398660, 27557894, 584, 12288, 33792, 66048, 131328, 262272,
    524352, 1048608, 2097168, 4194312, 4784257, 9569282, 9699330, 14221315, 19144708, 19169292, 19922948,
    23363589, 25165828, 292, 20480, 34816, 66560, 131584, 262400, 524416, 1048640, 2097184, 2396160, 4194320,
    4784385, 4849665, 8388612, 8388624, 9570306, 9584643, 9961474, 16777217, 19152900, 19169300, 19173892, 146,
    24576, 67584, 132096, 262656, 524544, 1048704, 2097216, 4194336, 4784641, 4980737, 8388616, 9572354,
    12582913, 16777218, 16777223, 16777248, 19169316, 23068676, 26361862, 28704774, 7, 19, 37, 73, 145, 289, 577,
    1153, 2305, 4609, 9217, 18433, 36865, 40960, 69632, 73729, 133120, 147457, 263168, 294913, 524800, 589825,
```

src/cauchy_best_r6.c lines 1321 to 1380

```
    1048832, 1179649, 1198080, 2097280, 2359297, 4194368, 4785153, 9576450, 9584646, 11534338, 19169348,
    19202052, 25165826, 25165831, 11, 38, 74, 290, 578, 1154, 2306, 4610, 9218, 18434, 36866, 49152, 73730,
    135168, 147458, 264192, 294914, 525312, 589826, 1049088, 1179650, 2097408, 2359298, 4194432, 4718594,
    4786177, 5767169, 8388615, 8388640, 9437185, 9584650, 13631490, 13778947, 16777224, 16777229, 16777238,
    16777292, 16777364, 16777508, 16777796, 16778372, 16779524, 16781828, 16786436, 16795652, 16814084, 16850948,
    16924676, 17072132, 17367044, 17956868, 19169412, 19267588, 20971521, 21495812, 27262980, 28606470, 13, 22,
    76, 148, 580, 1156, 2308, 4612, 9220, 18436, 36868, 73732, 81920, 139264, 147460, 266240, 294916, 526336,
    589828, 599040, 1049600, 1179652, 2097664, 2359300, 4194560, 4718596, 4788225, 6815745, 8388619, 8388646,
    8388672, 8388682, 8388754, 8388898, 8389186, 8389762, 8390914, 8393218, 8397826, 8407042, 8425474, 8462338,
    8536066, 8683522, 8978434, 9584658, 9601026, 10747906, 13107202, 16777230, 16777232, 16777237, 18874369,
    19169540, 20971527, 23920645, 26214404, 14, 21, 26, 35, 44, 152, 296, 1160, 2312, 4616, 9224, 18440, 36872,
    73736, 98304, 147464, 270336, 294920, 528384, 589832, 1050624, 1179656, 2098176, 2359304, 4194311, 4194323,
    4194341, 4194377, 4194449, 4194593, 4194816, 4194881, 4195457, 4196609, 4198913, 4203521, 4212737, 4231169,
    4268033, 4341761, 4489217, 4718600, 4792323, 5373953, 6553601, 8388627, 8388736, 9437188, 9437200, 9584674,
    9633794, 16777253, 18874370, 18874375, 18874400, 19169796, 19660804, 21569541, 23887877, 25165838, 28459014,
    29360135, 25, 52, 67, 88, 304, 592, 2320, 4624, 9232, 18448, 36880, 73744, 147472, 163840, 278528, 294928,
    299520, 532480, 589840, 1052672, 1179664, 2099200, 2359312, 4194315, 4195328, 4718608, 4792325, 4800513,
    8388622, 8388643, 8388864, 9437192, 9584706, 9586946, 12582919, 13180931, 13631489, 16777254, 16777280,
    16777285, 19170308, 20185092, 20971533, 25165846, 27262978, 27262983, 41, 50, 69, 104, 131, 176, 608, 1184,
    4640, 9248, 18464, 36896, 73760, 147488, 196608, 294944, 540672, 589856, 1056768, 1179680, 2101248, 2359328,
    4194317, 4196352, 4718624, 4792329, 4816897, 8388630, 8388675, 8389120, 9584770, 9830402, 12582923, 13107201,
    14352387, 16777244, 16777286, 16777344, 16777349, 18874376, 18874381, 18874390, 18874444, 18874516, 18874660,
    18874948, 18875524, 18876676, 18878980, 18883588, 18892804, 18911236, 18948100, 19021828, 19171332, 19464196,
    20054020, 20971541, 23068673, 23592964, 25165862, 26214402, 26214407, 28, 42, 49, 70, 82, 100, 133, 208, 259,
    352, 1216, 2368, 9280, 18496, 36928, 73792, 147520, 149760, 294976, 327680, 557056, 589888, 1064960, 1179712,
    2105344, 2359360, 4194325, 4194339, 4198400, 4718656, 4792337, 8388634, 8388739, 8389632, 9437191, 9437216,
    9584898, 10092546, 12582931, 16777260, 16777350, 16777472, 16777477, 18874382, 18874384, 18874389, 20971557,
    23068679, 23330820, 25165894, 28311556, 81, 134, 137, 164, 200, 261, 416, 515, 704, 2432, 4736, 18560, 36992,
    73856, 147584, 295040, 393216, 589952, 1081344, 1179776, 2113536, 2359424, 4194329, 4194371, 4202496,
    4718720, 4792353, 4915201, 8388650, 8388678, 8388867, 8390656, 9437195, 9437222, 9437248, 9437258, 9437330,
    9437474, 9437762, 9438338, 9439490, 9441794, 9446402, 9455618, 9474050, 9510914, 9585154, 9732098, 10027010,
    11796482, 12582947, 14155778, 14303235, 16777268, 16777478, 16777728, 16777733, 18874405, 19173376, 19173381,
    19177476, 20971589, 21495809, 23666693, 25165958, 27262990, 31457287, 97, 138, 262, 265, 274, 328, 400, 517,
    832, 1027, 1408, 4864, 9472, 37120, 73984, 74880, 147712, 295168, 590080, 655360, 1114112, 1179904, 2129920,
    2359552, 4194345, 4194373, 4194435, 4210688, 4718848, 4792385, 5046273, 8388658, 8388742, 8389123, 8392704,
    9437203, 9437312, 9585666, 11665410, 12582979, 16777300, 16777356, 16777734, 16778240, 16778245, 18874406,
    18874432, 18874437, 19173382, 19185668, 20971653, 21495815, 23068685, 25166086, 26214414, 27262998, 27525124,
    30408711, 56, 84, 98, 140, 161, 266, 518, 521, 530, 656, 800, 1029, 1664, 2051, 9728, 18944, 74240, 147968,
    295424, 590336, 786432, 1180160, 2162688, 2359808, 4194353, 4194437, 4194563, 4227072, 4718599, 4718611,
    4718629, 4718665, 4718737, 4718881, 4719104, 4719169, 4720897, 4723201, 4727809, 4737025, 4755457, 4792449,
    4866049, 5013505, 5898241, 7077889, 8388690, 8388870, 8389635, 8396800, 9437198, 9437219, 9437440, 9568257,
    12583043, 13631495, 13762562, 14229507, 16777316, 16777484, 16778246, 16779264, 16779269, 18874396, 18874438,
    18874496, 18874501, 19136513, 20971781, 23068693, 25166342, 26214422, 27263014, 27561990, 28311554, 28311559,
    28737542, 29884423 };

static int cbest_26[1023] = {
    1, 2, 33554467, 4, 50331698, 8, 25165849, 16, 32, 64, 46137391, 128, 256, 56623156, 512, 33554466, 16777233,
    3, 1024, 28311578, 33554465, 6, 50331699, 5, 41943083, 12, 2048, 25165848, 50331696, 9, 24, 12582924,
    33554471, 10, 4096, 14155789, 58720314, 48, 6291462, 54526006, 17, 33554475, 96, 8192, 3145731, 50331702, 20,
    25165841, 25165851, 18, 33, 192, 29360157, 33554483, 50331682, 34, 16384, 27263003, 33554435, 35127330,
    46137387, 46137390, 50331706, 384, 25165853, 40632357, 40, 65, 66, 768, 32768, 17563665, 25165833, 46137389,
    56623158, 68, 1048577, 50331666, 36, 129, 1536, 2097154, 23068695, 33554531, 46137383, 48234541, 80, 130,
    65536, 4194308, 28311579, 34078755, 47185966, 56623152, 56623157, 132, 3072, 8388616, 28311576, 136, 257,
    14155788, 16777232, 33554595, 6144, 131072, 42336299, 45088808, 50331762, 50593842, 53870641, 160, 260,
    25165881, 72, 258, 264, 513, 12288, 1572865, 3145730, 33554723, 272, 262144, 7077894, 23592983, 25296921,
    33554464, 50331826, 57671733, 24576, 22544404, 25165913, 47710254, 61866041, 320, 514, 520, 1025, 6291460,
    33554979, 34340899, 41943082, 46137407, 7, 528, 49152, 524288, 16777235, 46137359, 50331954, 544, 3145729,
    3538947, 20971541, 25165977, 33554469, 35127331, 50331697, 54722614, 56623140, 13, 144, 516, 1026, 2049,
    98304, 11272202, 12582920, 28311570, 33554470, 33555491, 41943081, 45350952, 56623164, 14, 640, 1040,
    1048576, 14155785, 16777237, 46202927, 50332210, 50724914, 58720315, 25, 1056, 196608, 6291458, 12582925,
```

```
    25166105, 27263001, 41943075, 44040233, 54526002, 54526007, 11, 26, 1028, 1088, 2050, 4097, 16777217,
    23855127, 25165825, 25165840, 25165850, 28311582, 33554474, 33554479, 33556515, 46137455, 50331700, 51118130,
    60489787, 62390329, 28, 49, 393216, 5636101, 6291463, 16777241, 29360156, 33554473, 35323938, 46137379,
    50331703, 50332722, 56623124, 58720312, 50, 1280, 2080, 12582916, 12582926, 22675476, 25165845, 25166361,
    25362457, 27263002, 28311562, 33554451, 33554491, 36700192, 39845925, 40632359, 41943087, 47185967, 50331650,
    50331690, 54526004, 22, 52, 97, 288, 1032, 2052, 2112, 4098, 8193, 786432, 14155781, 14155791, 18350096,
    25165852, 27361307, 29360153, 33554482, 33558563, 46137385, 46137519, 50331683, 56623154, 56655924, 58720306,
    60817464, 64487487, 21, 56, 2176, 2097152, 9175048, 14680078, 23068693, 25559065, 33554434, 33554499,
    46137386, 50331707, 50331710, 50333746, 62914622, 19, 100, 193, 1572864, 4587524, 13631501, 17563664,
    25165832, 25165843, 25166873, 28311577, 33554433, 40632353, 40632356, 45088809, 50331704, 53870640, 55574583,
    35, 44, 98, 104, 2056, 2560, 4100, 4160, 8194, 16385, 2293762, 8781832, 17661969, 25165855, 33554481,
    33562659, 36372513, 45482024, 46137647, 48234543, 50331680, 53477425, 56623220, 58720318, 42, 70, 112, 385,
    4224, 1146881, 4390916, 7340039, 11337738, 16777265, 23068694, 30408732, 33554487, 35127328, 46137388,
    50331686, 50335794, 56623159, 57147444, 38, 41, 140, 194, 200, 386, 4352, 2195458, 25165865, 25167897,
    28327962, 29360149, 29360159, 33554659, 41943099, 50331667, 50331730, 57671732, 58720298, 67, 88, 208, 280,
    576, 769, 2064, 4104, 8196, 16386, 32769, 1097729, 3145735, 23068691, 27262995, 28311610, 33554443, 33554530,
    33570851, 37224480, 40370213, 41943051, 46137382, 46137903, 46235695, 47710255, 48234537, 48234540, 50331664,
    54525990, 56623284, 61866040, 69, 84, 134, 224, 560, 5120, 8320, 3145728, 3538946, 4194304, 6291470,
    11534347, 16777297, 17039377, 20316178, 31457311, 33554529, 33554851, 34078754, 45088810, 46137381, 47185962,
    50339890, 53870643, 54526014, 56623153, 61341752, 37, 76, 196, 268, 400, 772, 1120, 1537, 8448, 2097155,
    7077892, 12582940, 15204366, 17301521, 18612240, 22544405, 23592981, 25165837, 25169945, 26935320, 34127907,
    34603042, 46137399, 46333999, 47235118, 64749631, 81, 131, 176, 416, 536, 770, 2240, 4112, 8200, 8704, 16388,
    32770, 65537, 1769473, 3145739, 4194309, 5668869, 14155784, 14155805, 17563667, 22741012, 25165835, 25165880,
    27263007, 28311642, 28573722, 33554439, 33554594, 33555235, 33587235, 35127334, 35651617, 46138415, 50331890,
    56098871, 56623412, 63799358, 65798204, 82, 133, 138, 168, 388, 448, 1072, 3073, 4480, 1048579, 6291478,
    8388617, 9306120, 14163981, 16777361, 24117270, 25165873, 28835866, 29360141, 33554535, 33554593, 34078753,
    34103331, 42074155, 42336298, 45350953, 47185964, 50331674, 50331763, 50348082, 50593843, 56623160, 57671735,
    58720282, 60293179, 137, 152, 800, 1538, 1544, 2144, 8960, 10240, 16640, 6291456, 12582956, 23592982,
    25165945, 25174041, 25296920, 27262987, 28311568, 30933020, 33556003, 39321638, 41943147, 42205227, 42385451,
    46137403, 48234533, 50331760, 50332082, 51740723, 53739569, 54525974, 61866043, 262, 352, 832, 1152, 4128,
    4288, 6145, 8208, 16392, 16896, 17920, 32772, 65538, 131073, 3145747, 3538945, 4194310, 4653060, 5636100,
    7602183, 10158089, 12648460, 14155821, 17563669, 25165912, 28311571, 28311580, 28311706, 30670876, 33554547,
    33554722, 33620003, 35127338, 37748775, 46139439, 47185958, 49283116, 50331746, 51380275, 56623668, 56672308,
    57409588, 74, 161, 261, 276, 336, 392, 896, 3074, 8576, 17408, 35840, 1048581, 6291494, 6324230, 7077890,
    8388608, 8388618, 11272200, 16777489, 23592979, 25166041, 33554599, 33557539, 40632365, 42336297, 44826665,
    50331670, 50331827, 50332466, 50364466, 50593840, 50618418, 52428848, 56623136, 56623166, 57671729, 60489786,
    62390328, 73, 259, 265, 304, 1540, 1600, 3088, 12289, 17152, 71680, 2097158, 2326530, 2818050, 3162115,
    12058635, 12582988, 13467660, 14155780, 14155790, 14286861, 16777234, 17563649, 21168149, 22544400, 25165883,
    25182233, 26214424, 33554563, 33554603, 34439203, 35323939, 41943211, 46137375, 47710250, 50331766, 50331824,
    54591542, 56623142, 56721460, 58851386, 164, 266, 273, 524, 704, 776, 1664, 6146, 8224, 16400, 20480, 32776,
    33280, 34304, 65540, 131074, 143360, 262145, 3145763, 5242885, 7077895, 10485770, 11370506, 11796491,
    12582912, 13107212, 14155853, 14417933, 17563673, 19660819, 20971540, 25165976, 25166233, 28311583, 28311834,
    33554721, 33554978, 33560611, 33685539, 34078759, 34340898, 35127346, 36388897, 36700193, 37355552, 42336291,
    45088800, 45350954, 46137351, 46137406, 46141487, 50333234, 50606130, 54657078, 56624180, 58720378, 61603896,
    162, 552, 672, 1792, 3076, 24577, 33792, 68608, 286720, 1048585, 1163265, 1409025, 6291526, 6553606, 8388620,
    13631500, 14155777, 15466510, 16777745, 17170449, 23068679, 23617559, 23855125, 25165897, 25165969, 28311554,
    33554727, 35127298, 35135522, 36175905, 40632373, 40636453, 46137355, 46137358, 46137423, 50331955, 50397234,
    54526070, 56623108, 56623148, 59244602, 64487486, 321, 515, 521, 608, 3200, 6176, 12290, 34816, 137216,
    573440, 1572867, 2097162, 3276803, 6291461, 6815750, 12583052, 15335438, 16777236, 20971537, 22675477,
    23068703, 24641558, 25165915, 25166617, 25198617, 28311563, 29360189, 33554468, 33554539, 33554731, 33566755,
    40632325, 41943339, 44040235, 46137357, 46137451, 47710252, 50331830, 50334770, 54722610, 54722615, 56623126,
    56623141, 148, 274, 322, 522, 529, 784, 1048, 1408, 2304, 3328, 6148, 8256, 16416, 32784, 49153, 65544,
    131076, 262146, 274432, 524289, 1146880, 3145795, 3407875, 4194316, 7340038, 14155917, 17567761, 18350097,
    18677776, 25165917, 25166104, 25296913, 25296923, 25309209, 27263000, 27263035, 27295771, 28311560, 28311574,
    28312090, 28336154, 28704794, 29425693, 32899102, 33554739, 33554977, 33555490, 33816611, 34078763, 34340897,
    37093408, 41943080, 42336303, 45088812, 46137405, 46145583, 50331794, 50331834, 50331938, 50593846, 53870645,
    55574582, 56229943, 56623165, 56625204, 58720442, 62390331, 63701054, 65929276, 518, 545, 1104, 1344, 3080,
    3584, 24578, 40960, 66560, 548864, 1048593, 3670019, 6291590, 12582922, 16777216, 16778257, 22544406,
    23855126, 25165824, 25165885, 25165905, 25166097, 25167385, 27328539, 27361305, 31195164, 33579043, 35192866,
    39583782, 40632355, 44138537, 45482025, 46202923, 46202926, 47710246, 50331952, 50332211, 50337842, 50462770,
```

```
    50724915, 51904563, 53477424, 54526000, 54526134, 60424251, 61866033, 145, 328, 517, 532, 546, 1027, 1216,
    1552, 6400, 12292, 12352, 67584, 98305, 1097728, 1572869, 2097170, 3145734, 6733830, 11272203, 12582921,
    12583180, 14155787, 14680076, 16777225, 16777239, 16777240, 25165979, 25231385, 25303065, 28360730, 29360221,
    29622301, 33554787, 33554947, 33554987, 35127329, 35258402, 36700194, 38633511, 39452710, 39845927, 40697893,
    41943074, 41943079, 41943595, 42139691, 43384874, 44040232, 46137515, 46137583, 48234557, 50331958, 50593826,
    50774066, 51216434, 51642419, 53870625, 54526003, 54722612, 54747190, 59424826, 15, 641, 6152, 6656, 49154,
    69632, 131080, 4194324, 11534346, 14156045, 17596433, 25168921, 25296925, 25362456, 27263067, 33556514,
    35389474, 41943073, 50331701, 50331770, 50332194, 50343986, 50593850, 60817466 };

static int cbest_27[1023] = {
    1, 2, 67108883, 4, 100663322, 8, 50331661, 16, 32, 92274709, 64, 128, 113246233, 256, 512, 67108882,
    33554441, 3, 1024, 67108881, 123731999, 6, 100663323, 5, 2048, 83886103, 12, 50331660, 100663320, 9, 24,
    4096, 25165830, 67108887, 109051928, 10, 117440542, 48, 12582915, 50331657, 17, 8192, 54525964, 67108891,
    100663314, 128974876, 18, 96, 67108867, 92274711, 100663326, 20, 50331663, 73400338, 92274708, 113246232, 33,
    192, 16384, 27262982, 58720271, 34, 50331653, 92274705, 36, 384, 36700169, 46137354, 100663306, 40, 65,
    32768, 13631491, 56623116, 67108915, 66, 768, 96469012, 113246235, 68, 23068677, 64487438, 72, 129, 1536,
    65536, 67108947, 80, 4194305, 28311558, 73924626, 100663354, 132, 3072, 8388610, 48234506, 50331677,
    85458967, 123731998, 130, 136, 257, 131072, 16777220, 67109011, 69206035, 78643217, 144, 6144, 33554440,
    92274717, 100663386, 160, 14155779, 36962313, 50331693, 92274693, 258, 264, 513, 12288, 262144, 24117253,
    67109139, 101711898, 113246225, 272, 6291457, 12582914, 32243719, 100663450, 109838360, 113246237, 288,
    24576, 50331725, 67108880, 260, 320, 514, 1025, 524288, 50855949, 61865999, 67109395, 528, 49152, 25165828,
    70254611, 74186770, 83886102, 92274741, 100663578, 106430491, 123731995, 7, 544, 33554443, 50331789,
    79167505, 113246217, 123731997, 516, 576, 1026, 2049, 98304, 1048576, 12582913, 41943051, 54525965, 54919180,
    67108885, 67109907, 73400339, 83886099, 85590039, 100663321, 109051930, 13, 640, 33554433, 50331656,
    67108886, 83886101, 92274773, 92536853, 100663834, 109051929, 14, 26, 1056, 196608, 13631490, 33554445,
    50331649, 50331917, 83230736, 102236186, 117440543, 25, 52, 520, 1028, 1088, 2050, 4097, 2097152, 6815745,
    25165826, 25165831, 37093385, 67110931, 88080406, 92274707, 98041876, 123731991, 128974878, 11, 104, 1152,
    393216, 23068676, 27262980, 50331659, 50331662, 58720269, 67108875, 67108890, 67108895, 79691792, 92274837,
    94371861, 100663298, 100663315, 100663318, 100663324, 100664346, 103809050, 113246265, 117440538, 128974877,
    22, 28, 49, 208, 1280, 11534338, 27459590, 39845896, 46137355, 50332173, 58720270, 67108866, 67108889,
    92274710, 96469013, 100663312, 100663327, 113246234, 117440540, 44, 416, 1032, 2052, 2112, 4098, 8193,
    786432, 4194304, 5767169, 19922948, 44040203, 46137352, 50331652, 51118093, 54525960, 56623117, 67108865,
    67108899, 67112979, 109903896, 113377305, 120324126, 125829148, 128974872, 19, 88, 97, 832, 2176, 9961474,
    13631489, 36700168, 70516755, 92274965, 100665370, 106692635, 113246297, 121634847, 21, 38, 50, 56, 176,
    1664, 2304, 1572864, 4980737, 18350084, 29360135, 33554457, 50332685, 51904525, 64487439, 73924627, 92274704,
    109051920, 123731983, 76, 193, 352, 1040, 2056, 2560, 3328, 4100, 8194, 16385, 9175042, 27262978, 27262983,
    41615368, 49020938, 54525966, 58720267, 64487436, 67108871, 67117075, 73400336, 83886111, 90177558,
    100663307, 109051932, 114294809, 117440534, 35, 98, 152, 194, 704, 4224, 6656, 3145728, 4587521, 13729795,
    14155778, 50331669, 62914574, 67108914, 67108979, 69992467, 81788944, 83886087, 89128982, 92275221, 96469014,
    100663304, 100663338, 100667418, 113246361, 115343385, 37, 42, 70, 112, 304, 385, 1408, 4352, 13312,
    12582919, 24117252, 28311556, 33554473, 48234507, 50331655, 50333709, 54525956, 67108913, 80740368, 85655575,
    92274713, 92667925, 95420437, 128974868, 41, 140, 608, 2064, 2816, 4104, 4608, 8196, 16386, 26624, 32769,
    6291456, 7077889, 8388608, 25165838, 32243718, 40894472, 48234504, 54951948, 56623112, 67109075, 67125267,
    69599251, 102367258, 127926300, 67, 100, 280, 388, 769, 1216, 5120, 5632, 53248, 12058626, 40370184,
    45088779, 50331673, 50331676, 67108946, 92275733, 93061141, 96469008, 100663310, 100671514, 104071194,
    109051912, 113246224, 113246489, 123732031, 131596317, 69, 74, 84, 224, 386, 560, 2432, 8448, 11264, 106496,
    12582923, 20447236, 23068673, 31457287, 33554505, 36700171, 44564491, 50335757, 58720263, 60162063, 64487434,
    67108919, 67108945, 67109267, 69402643, 73400342, 92274719, 100663355, 100663418, 113246236, 117440526,
    123797535, 73, 196, 1120, 1537, 2080, 4112, 4864, 8200, 8704, 16388, 22528, 32770, 65537, 212992, 6029313,
    14155777, 20185092, 24510469, 25165846, 56623118, 67108923, 67141651, 78643219, 83886135, 85458966, 92274695,
    99352596, 100663346, 100663352, 102105114, 113246239, 81, 134, 770, 776, 2240, 9216, 9728, 45056, 425984,
    10223618, 20807684, 28311554, 28311559, 34603017, 36700161, 36962312, 46137346, 50331689, 50331692, 50331709,
    67109010, 67109651, 69206034, 73973778, 78643216, 91226134, 92274689, 92276757, 100663514, 100679706,
    113246227, 113246745, 120455198, 123732063, 128974860, 133, 148, 168, 448, 3073, 4480, 10240, 19456, 90112,
    851968, 8388611, 10092546, 12582912, 12582931, 23068679, 32243717, 33554569, 35651593, 36700173, 46137358,
    50339853, 51183629, 56623108, 63963150, 67108931, 67108951, 70647827, 71303186, 73400346, 73924624, 74186771,
    82837520, 85458963, 92274701, 92274716, 100663387, 101908506, 106430490, 113442841, 114819097, 123731994,
    124256287, 131, 137, 200, 1538, 4128, 8208, 8960, 16392, 16896, 32772, 38912, 65538, 131073, 180224, 1703936,
    5111809, 13631495, 16121859, 16777216, 16777221, 18481156, 25165862, 27262990, 27475974, 39321608, 50331679,
    50331757, 52035597, 54525980, 67108955, 67110419, 67174419, 73400322, 75497489, 83886167, 85458965, 89653270,
```

```
    92274692, 100663358, 100663378, 100663384, 100663706, 109051960, 109936664, 113246216, 123731996, 124780575,
    82, 138, 145, 268, 772, 1552, 6145, 17408, 17920, 77824, 360448, 3407872, 4194307, 5046273, 42729483,
    50331685, 50331721, 50331724, 61865997, 64487430, 67109009, 67109138, 69206033, 81264656, 84410391, 89391126,
    92274725, 92278805, 100696090, 101711899, 101810202, 109838362, 113246209, 113246229, 113247257, 113639449,
    117440574, 123732127, 146, 161, 296, 336, 392, 896, 3074, 18432, 35840, 155648, 720896, 6815744, 9240578,
    12582947, 19660804, 33554697, 41418760, 50331853, 50348045, 50855948, 51052557, 61865998, 67108995, 67109015,
    67111955, 70123539, 84934679, 92274743, 100663451, 100664090, 109838361, 162, 259, 265, 1540, 4160, 8224,
    12289, 16400, 20480, 32776, 65540, 71680, 131074, 262145, 311296, 1441792, 13631499, 16777222, 24117249,
    25165824, 25165894, 25427974, 27262998, 45613067, 50331695, 54525996, 54919181, 58720287, 67109019, 67239955,
    83230737, 83886231, 92274737, 97189908, 100663370, 100663390, 100663442, 104857627, 109051992, 109314072,
    113246219, 128974908, 129007644, 266, 273, 536, 3104, 6146, 33792, 143360, 622592, 2883584, 4194309, 4620289,
    9830402, 12713987, 31981575, 32243715, 33554442, 36986889, 40632328, 41943049, 48234498, 49676298, 50331717,
    50331785, 50331788, 50332045, 50954253, 53215245, 67109137, 67109394, 67115027, 73924630, 79364113, 88080407,
    92282901, 96469020, 98041877, 100664858, 100728858, 101711896, 109576216, 113248281, 117440606, 123731987,
    123731990, 123731993, 123732255, 262, 274, 289, 400, 592, 672, 1792, 3076, 24577, 34816, 286720, 1245184,
    5767168, 8388614, 10403842, 12582979, 20709380, 20971525, 23068685, 33554953, 36700185, 36962315, 41943050,
    44826635, 46137370, 48234510, 50364429, 54657036, 67109043, 67109123, 69664787, 70254610, 73400370, 79167507,
    79691793, 92274740, 92274775, 92274805, 92700693, 96468996, 100663448, 100663579, 102432794, 104202266,
    113246264, 117964830, 132907037, 261, 276, 290, 321, 515, 1544, 8256, 12290, 16416, 32784, 36864, 65544,
    131076, 262146, 524289, 573440, 2490368, 4915201, 13631488, 13631507, 13737987, 24117255, 25165958, 27263014,
    33554432, 35127305, 44695563, 50331727, 50332429, 50855945, 50905101, 54526028, 54788108, 56623132, 58720303,
    61865995, 67109147, 67121171, 67371027, 69206039, 69632019, 73465874, 77594641, 78643221, 79167504, 83886097,
    83886359, 88473622, 92274769, 93126677, 100663434, 100663454, 100663570, 100666394, 109052056, 111149080,
    128974940, 129237020, 292, 322, 529, 784, 1072, 6148, 6208, 40960, 49153, 1146880, 4194313, 4980736, 6291459,
    20316164, 25165829, 27328518, 28311566, 29360134, 30932999, 33554444, 36962305, 50331648, 50331781, 50331916,
    54968332, 60227599, 67108884, 67109393, 67109906, 73924634, 75169810, 83886098, 85590038, 92274901, 92291093,
    92536855, 100794394, 109051931, 113246249, 113250329, 113377304, 117440670, 119537694, 123732511, 129499164,
    164, 324, 518, 530, 545, 1184, 1344, 3080, 3584, 24578, 67584, 2293760, 8388618, 10354690, 12583043,
    14155783, 14680067, 23068693, 27394054, 33554437, 33555465, 36700201, 36732937, 36962317, 37093384, 39845897,
    46137386, 50333197, 50397197, 55574540, 64487454, 67109171, 67109379, 67133459, 70254609, 73400402, 73924610,
    74186768, 83099664, 83230738, 83886100, 85458975, 92274772, 92536852, 92635157, 100663576, 100663835,
    100669466, 101711890, 102170650, 106430489, 106692634, 113246267, 113246296, 121634846, 123830303, 124518431,
    128974874, 517, 532, 16448, 69632, 524290, 1048577, 4587520, 27262976, 46137353, 50331658, 50331741,
    54525961, 56623148, 58720268, 58982415, 67109143, 67109203, 67633171, 78643225, 100663482, 102236187,
    106954779, 123731982 };

static int cbest_28[1023] = {
    1, 2, 134217732, 4, 67108866, 8, 33554433, 16, 32, 150994948, 64, 128, 75497474, 256, 512, 37748737, 1024,
    2048, 4096, 153092100, 8192, 16384, 32768, 76546050, 65536, 131072, 262144, 38273025, 524288, 1048576,
    2097152, 4194304, 153354244, 8388608, 16777216, 33554432, 76677122, 67108864, 3, 134217728, 134217733, 5,
    38338561, 67108867, 134217734, 6, 9, 18, 36, 72, 144, 288, 576, 1152, 2304, 4608, 9216, 18432, 36864, 73728,
    147456, 294912, 589824, 1179648, 2359296, 4718592, 9437184, 18874368, 37748736, 201326598, 10, 17, 33554435,
    67108870, 75497472, 134217740, 167772165, 12, 33, 33554437, 67108874, 100663299, 134217748, 150994944,
    150994949, 20, 34, 65, 33554441, 67108882, 134217764, 150994950, 24, 66, 129, 33554449, 67108898, 75497475,
    134217796, 153387012, 218103814, 40, 68, 130, 257, 33554465, 67108930, 134217860, 150994956, 184549381,
    209715206, 48, 132, 258, 513, 33554497, 67108994, 75497478, 134217988, 150994964, 80, 136, 260, 514, 1025,
    33554561, 37748739, 67109122, 75497482, 109051907, 134218244, 150994980, 171966469, 96, 264, 516, 1026, 2049,
    33554689, 37748741, 67109378, 75497490, 104857603, 134218756, 150995012, 160, 272, 520, 1028, 2050, 4097,
    33554945, 37748745, 38273024, 67109890, 75497506, 76546048, 76693506, 134219780, 150995076, 153092096,
    153092101, 226492422, 192, 528, 1032, 2052, 4098, 8193, 19136512, 33555457, 37748753, 67110914, 75497538,
    134221828, 150995204, 153092102, 320, 544, 1040, 2056, 4100, 8194, 16385, 9568256, 33556481, 37748769,
    67112962, 75497602, 134225924, 150995460, 188743685, 220200966, 384, 1056, 2064, 4104, 8196, 16386, 32769,
    4784128, 33558529, 37748801, 67117058, 75497730, 76546051, 134234116, 150995972, 153092108, 186646533, 640,
    1088, 2080, 4112, 8200, 16388, 32770, 65537, 2392064, 33562625, 37748865, 67125250, 75497986, 113246211,
    134250500, 150996996, 153092116, 210763782, 768, 2112, 4128, 8208, 16392, 32772, 65538, 131073, 1196032,
    33570817, 37748993, 38346753, 67141634, 75498498, 76546054, 134283268, 150999044, 153092132, 1280, 2176,
    4160, 8224, 16400, 32776, 65540, 131074, 262145, 598016, 33587201, 37749249, 67174402, 75499522, 76546058,
    110100483, 134348804, 151003140, 153092164, 1536, 4224, 8256, 16416, 32784, 65544, 131076, 262146, 299008,
    524289, 33619969, 37749761, 38273027, 67239938, 75501570, 76546066, 134479876, 151011332, 153092228,
    172490757, 228589574, 2560, 4352, 8320, 16448, 32800, 65552, 131080, 149504, 262148, 524290, 1048577,
```

src/cauchy_best_r6.c lines 1561 to 1620

```
    33685505, 37750785, 38273029, 67371010, 75505666, 76546082, 105381891, 134742020, 151027716, 153092356,
    227540998, 3072, 8448, 16512, 32832, 65568, 74752, 131088, 262152, 524292, 1048578, 2097153, 33816577,
    37752833, 38273033, 67633154, 75513858, 76546114, 135266308, 151060484, 153092612, 190840837, 5120, 8704,
    16640, 32896, 37376, 65600, 131104, 262160, 524296, 1048580, 2097154, 4194305, 34078721, 37756929, 38273041,
    68157442, 75530242, 76546178, 136314884, 151126020, 153093124, 153354240, 153354245, 6144, 16896, 18688,
    33024, 65664, 131136, 262176, 524304, 1048584, 2097156, 4194306, 8388609, 34603009, 37765121, 38273057,
    69206018, 75563010, 76546306, 76677120, 138412036, 151257092, 153094148, 153354246, 153391108, 189267973,
    9344, 10240, 17408, 33280, 65792, 131200, 262208, 524320, 1048592, 2097160, 4194308, 8388610, 16777217,
    35651585, 37781505, 38273089, 38338560, 71303170, 75628546, 76546562, 114294787, 142606340, 151519236,
    153096196, 220463110, 4672, 12288, 33792, 66048, 131328, 262272, 524352, 1048608, 2097168, 4194312, 8388612,
    16777218, 37814273, 38273153, 75759618, 76547074, 113770499, 152043524, 153100292, 153354252, 186908677,
    2336, 20480, 34816, 66560, 131584, 262400, 524416, 1048640, 2097184, 4194320, 8388616, 16777220, 19169280,
    33554434, 37879809, 38273281, 41943041, 76021762, 76548098, 76677123, 83886082, 153108484, 153354260,
    167772164, 1168, 24576, 67584, 132096, 262656, 524544, 1048704, 2097216, 4194336, 8388624, 16777224,
    33554436, 38010881, 38273537, 50331649, 67108865, 76550146, 100663298, 153124868, 153354276, 155189252,
    210894854, 229638150, 584, 40960, 69632, 133120, 263168, 524800, 1048832, 2097280, 4194368, 8388640, 9584640,
    16777232, 33554440, 38274049, 76554242, 76677126, 77594626, 153157636, 153354308, 159383556, 201326596, 292,
    49152, 135168, 264192, 525312, 1049088, 2097408, 4194432, 8388672, 16777248, 33554448, 38275073, 38797313,
    67108868, 67108880, 76562434, 76677130, 76695554, 79691778, 110231555, 134217729, 153223172, 153354372,
    228851718, 146, 81920, 139264, 266240, 526336, 1049600, 2097664, 4194560, 4792320, 8388736, 16777280,
    33554464, 38277121, 39845889, 67108872, 76578818, 76677138, 100663297, 134217730, 134217735, 134217760,
    153354500, 184549380, 191365125, 7, 19, 37, 73, 145, 289, 577, 1153, 2305, 4609, 9217, 18433, 36865, 73729,
    98304, 147457, 270336, 294913, 528384, 589825, 1050624, 1179649, 2098176, 2359297, 4194816, 4718593, 8388864,
    9437185, 16777344, 18874369, 33554496, 38281217, 38338563, 76611586, 76677154, 92274690, 153354756,
    153616388, 172556293, 191102981, 201326594, 201326599, 227672070, 11, 38, 74, 290, 578, 1154, 2306, 4610,
    9218, 18434, 36866, 73730, 147458, 163840, 278528, 294914, 532480, 589826, 1052672, 1179650, 2099200,
    2359298, 2396160, 4195328, 4718594, 8389120, 9437186, 16777472, 18874370, 33554560, 37748738, 38289409,
    38338565, 46137345, 67108871, 67108896, 75497473, 76677186, 105447427, 109051906, 134217736, 134217741,
    134217750, 134217804, 134217876, 134218020, 134218308, 134218884, 134220036, 134222340, 134226948, 134236164,
    134254596, 134291460, 134365188, 134512644, 134807556, 135397380, 136577028, 138936324, 143654916, 153355268,
    154140676, 167772161, 171966468, 218103812, 13, 22, 76, 148, 580, 1156, 2308, 4612, 9220, 18436, 36868,
    73732, 147460, 196608, 294916, 540672, 589828, 1056768, 1179652, 2101248, 2359300, 4196352, 4718596, 8389632,
    9437188, 16777728, 18874372, 33554688, 37748740, 38305793, 38338569, 54525953, 67108875, 67108902, 67108928,
    67108938, 67109010, 67109154, 67109442, 67110018, 67111170, 67113474, 67118082, 67127298, 67145730, 67182594,
    67256322, 67403778, 67698690, 68288514, 69468162, 71827458, 76677250, 76808194, 85983234, 104857602,
    114819075, 134217742, 134217744, 134217749, 150994945, 153356292, 167772167, 209715204, 14, 21, 26, 35, 44,
    152, 296, 1160, 2312, 4616, 9224, 18440, 36872, 73736, 147464, 294920, 327680, 557056, 589832, 1064960,
    1179656, 1198080, 2105344, 2359304, 4198400, 4718600, 8390656, 9437192, 16778240, 18874376, 33554439,
    33554451, 33554469, 33554505, 33554577, 33554721, 33554944, 33555009, 33555585, 33556737, 33559041, 33563649,
    33572865, 33591297, 33628161, 33701889, 33849345, 34144257, 34734081, 35913729, 37748744, 38338577, 42991617,
    52428801, 67108883, 67108992, 75497476, 75497488, 76677378, 77070338, 134217765, 150994946, 150994951,
    150994976, 153358340, 157286404, 201326606, 234881031, 25, 52, 67, 88, 304, 592, 2320, 4624, 9232, 18448,
    36880, 73744, 147472, 294928, 393216, 589840, 1081344, 1179664, 2113536, 2359312, 4202496, 4718608, 8392704,
    9437200, 16779264, 18874384, 33554443, 33555456, 37748752, 38338593, 38347777, 38404097, 67108878, 67108899,
    67109120, 75497480, 76677634, 100663303, 109051905, 114425859, 134217766, 134217792, 134217797, 153362436,
    153387008, 153387013, 161480708, 167772173, 189333509, 201326614, 218103810, 218103815, 41, 50, 69, 104, 131,
    176, 608, 1184, 4640, 9248, 18464, 36896, 73760, 147488, 294944, 589856, 599040, 655360, 1114112, 1179680,
    2129920, 2359328, 4210688, 4718624, 8396800, 9437216, 16781312, 18874400, 33554445, 33556480, 37748768,
    38338625, 38535169, 67108886, 67108931, 67109376, 76678146, 78643202, 100663307, 104857601, 134217756,
    134217798, 134217856, 134217861, 150994952, 150994957, 150994966, 150995020, 150995092, 150995236, 150995524,
    150996100, 150997252, 150999556, 151004164, 151013380, 151031812, 151068676, 151142404, 151289860, 151584772,
    152174596, 153370628, 153387014, 155713540, 160432132, 167772181, 184549377, 188743684, 201326630, 209715202,
    209715207, 28, 42, 49, 70, 82, 100, 133, 208, 259, 352, 1216, 2368, 9280, 18496, 36928, 73792, 147520,
    294976, 589888, 786432, 1179712, 2162688, 2359360, 4227072, 4718656, 8404992, 9437248, 16785408, 18874432,
    33554453, 33554467, 33558528, 37748800, 38338689, 67108890, 67108995, 67109888, 75497479, 75497504, 76679170,
    76693504, 80740354, 100663315, 113836035, 134217772, 134217862, 134217984, 134217989, 150994958, 150994960,
    150994965, 167772197, 184549383, 186646532, 201326662, 220495878, 226492420, 229900294, 81, 134, 137, 164,
    200, 261, 416, 515, 704, 4736, 36992, 73856, 147584, 295040, 299520, 1179776, 2228224, 2359424, 8421376,
    9437312, 18874496, 33554457, 33554499, 33562624, 37748864, 67108906, 67108934, 67109123, 67110912, 75497483,
    75497510, 75497536, 75497546, 75497762, 75498050, 75499778, 75502082, 75506690, 75515906, 75534338, 75571202,
```

src/cauchy_best_r6.c lines 1621 to 1680

```
    75644930, 75792386, 76087298, 76681218, 77856770, 80216066, 94371842, 100663331, 113246210, 134217780,
    134217990, 134218240, 150994981, 153419780, 167772229, 171966465, 201326726, 218103822, 229769222, 251658247 };

static int cbest_29[1023] = {
    1, 2, 268435458, 4, 134217729, 8, 16, 335544322, 32, 64, 167772161, 128, 256, 512, 352321538, 1024, 2048,
    4096, 176160769, 8192, 16384, 32768, 65536, 356515842, 131072, 262144, 524288, 1048576, 178257921, 2097152,
    4194304, 8388608, 16777216, 33554432, 357564418, 67108864, 134217728, 268435456, 3, 178782209, 268435459, 5,
    10, 20, 40, 80, 160, 320, 640, 1280, 2560, 5120, 10240, 20480, 40960, 81920, 163840, 327680, 655360, 1310720,
    2621440, 5242880, 10485760, 20971520, 41943040, 83886080, 167772160, 6, 9, 134217731, 268435462, 335544320,
    402653187, 17, 134217733, 268435466, 335544323, 12, 18, 33, 134217737, 268435474, 34, 65, 134217745,
    268435490, 335544326, 469762051, 24, 36, 66, 129, 134217761, 167772163, 268435522, 335544330, 357826562,
    436207619, 68, 130, 257, 134217793, 167772165, 176160768, 268435586, 335544338, 352321536, 48, 72, 132, 258,
    513, 88080384, 134217857, 167772169, 268435714, 335544354, 352321539, 136, 260, 514, 1025, 44040192,
    134217985, 167772177, 268435970, 335544386, 503316483, 96, 144, 264, 516, 1026, 2049, 22020096, 134218241,
    167772193, 268436482, 335544450, 352321542, 486539267, 272, 520, 1028, 2050, 4097, 11010048, 134218753,
    167772225, 268437506, 335544578, 352321546, 192, 288, 528, 1032, 2052, 4098, 8193, 5505024, 134219777,
    167772289, 176160771, 178913281, 268439554, 335544834, 352321554, 444596227, 544, 1040, 2056, 4100, 8194,
    16385, 2752512, 134221825, 167772417, 176160773, 268443650, 335545346, 352321570, 384, 576, 1056, 2064, 4104,
    8196, 16386, 32769, 1376256, 134225921, 167772673, 176160777, 268451842, 335546370, 352321602, 356515840,
    520093699, 1088, 2080, 4112, 8200, 16388, 32770, 65537, 688128, 134234113, 167773185, 176160785, 178257920,
    268468226, 335548418, 352321666, 356515843, 511705091, 768, 1152, 2112, 4128, 8208, 16392, 32772, 65538,
    131073, 344064, 134250497, 167774209, 176160801, 268500994, 335552514, 352321794, 2176, 4160, 8224, 16400,
    32776, 65540, 131074, 172032, 262145, 89128960, 134283265, 167776257, 176160833, 268566530, 335560706,
    352322050, 356515846, 490733571, 1536, 2304, 4224, 8256, 16416, 32784, 65544, 86016, 131076, 262146, 524289,
    134348801, 167780353, 176160897, 268697602, 335577090, 352322562, 356515850, 4352, 8320, 16448, 32800, 43008,
    65552, 131080, 262148, 524290, 1048577, 44564480, 134479873, 167788545, 176161025, 268959746, 335609858,
    352323586, 356515858, 357892098, 3072, 4608, 8448, 16512, 21504, 32832, 65568, 131088, 262152, 524292,
    1048578, 2097153, 134742017, 167804929, 176161281, 178257923, 269484034, 335675394, 352325634, 356515874,
    446693379, 528482307, 8704, 10752, 16640, 32896, 65600, 131104, 262160, 524296, 1048580, 2097154, 4194305,
    22282240, 135266305, 167837697, 176161793, 178257925, 270532610, 335806466, 352329730, 356515906, 524288003,
    5376, 6144, 9216, 16896, 33024, 65664, 131136, 262176, 524304, 1048584, 2097156, 4194306, 8388609, 136314881,
    167903233, 176162817, 178257929, 272629762, 336068610, 352337922, 356515970, 2688, 17408, 33280, 65792,
    131200, 262208, 524320, 1048592, 2097160, 4194308, 8388610, 11141120, 16777217, 138412033, 168034305,
    176164865, 178257937, 276824066, 336592898, 352354306, 356516098, 357564416, 513802243, 1344, 12288, 18432,
    33792, 66048, 131328, 262272, 524352, 1048608, 2097168, 4194312, 8388612, 16777218, 33554433, 142606337,
    168296449, 176168961, 178257953, 285212674, 337641474, 352387074, 356516354, 357564419, 672, 34816, 66560,
    131584, 262400, 524416, 1048640, 2097184, 4194320, 5570560, 8388616, 16777220, 33554434, 67108865, 150994945,
    168820737, 176177153, 178257985, 178782208, 301989890, 339738626, 352452610, 356516866, 336, 24576, 36864,
    67584, 132096, 262656, 524544, 1048704, 2097216, 4194336, 8388624, 16777224, 33554436, 67108866, 169869313,
    176193537, 178258049, 178946049, 343932930, 352583682, 356517890, 357564422, 491782147, 168, 69632, 133120,
    263168, 524800, 1048832, 2097280, 2785280, 4194368, 8388640, 16777232, 33554440, 67108868, 134217730,
    171966465, 176226305, 178258177, 201326593, 352845826, 356519938, 357564426, 402653186, 532676611, 84, 49152,
    73728, 135168, 264192, 525312, 1049088, 2097408, 4194432, 8388672, 16777248, 33554448, 67108872, 89391104,
    134217732, 176291841, 178258433, 268435457, 353370114, 356524034, 357564434, 369098754, 530579459, 42,
    139264, 266240, 526336, 1049600, 1392640, 2097664, 4194560, 8388736, 16777280, 33554464, 67108880, 134217736,
    176422913, 178258945, 184549377, 268435464, 354418690, 356532226, 357564450, 11, 21, 41, 81, 161, 321, 641,
    1281, 2561, 5121, 10241, 20481, 40961, 81921, 98304, 147456, 163841, 270336, 327681, 528384, 655361, 1050624,
    1310721, 2098176, 2621441, 4194816, 5242881, 8388864, 10485761, 16777344, 20971521, 33554496, 41943041,
    67108896, 83886081, 134217744, 176685057, 178259969, 178782211, 268435460, 356548610, 357564482, 402653185,
    447217667, 469762050, 525336579, 7, 22, 82, 162, 322, 642, 1282, 2562, 5122, 10242, 20482, 40962, 81922,
    163842, 278528, 327682, 532480, 655362, 696320, 1052672, 1310722, 2099200, 2621442, 4195328, 5242882,
    8389120, 10485762, 16777472, 20971522, 33554560, 41943042, 44695552, 67108928, 83886082, 134217760,
    167772162, 177209345, 178262017, 178782213, 234881025, 268435463, 268435478, 268435498, 268435538, 268435618,
    268435778, 268436098, 268436738, 268438018, 268440578, 268445698, 268455938, 268476418, 268517378, 268599298,
    268763138, 269090818, 269746178, 271056898, 273678338, 278921218, 289406978, 310378498, 335544321, 356581378,
    357564546, 360710146, 436207618, 14, 44, 164, 324, 644, 1284, 2564, 5124, 10244, 20484, 40964, 81924, 163844,
    196608, 294912, 327684, 540672, 655364, 1056768, 1310724, 2101248, 2621444, 4196352, 5242884, 8389632,
    10485764, 16777728, 20971524, 33554688, 41943044, 67108992, 83886084, 134217739, 134217749, 134217769,
    134217792, 134217809, 134217889, 134218049, 134218369, 134219009, 134220289, 134222849, 134227969, 134238209,
    134258689, 134299649, 134381569, 134545409, 134873089, 135528449, 136839169, 139460609, 144703489, 155189249,
```

src/cauchy_best_r6.c lines 1681 to 1740

```
    167772164, 178266113, 178782217, 218103809, 268435467, 268435472, 335544328, 356646914, 357564674, 13, 19,
    28, 88, 328, 648, 1288, 2568, 5128, 10248, 20488, 40968, 81928, 163848, 327688, 348160, 557056, 655368,
    1064960, 1310728, 2105344, 2621448, 4198400, 5242888, 8390656, 10485768, 16778240, 20971528, 33554944,
    41943048, 67109120, 83886088, 134217735, 134217856, 167772168, 178274305, 178782225, 180355073, 268435475,
    268435488, 335544324, 356777986, 357564930, 385875970, 402653191, 469762049, 514326531, 26, 35, 56, 176, 656,
    1296, 2576, 5136, 10256, 20496, 40976, 81936, 163856, 327696, 393216, 589824, 655376, 1081344, 1310736,
    2113536, 2621456, 4202496, 5242896, 8392704, 10485776, 16779264, 20971536, 22347776, 33555456, 41943056,
    67109376, 83886096, 134217984, 167772176, 178290689, 178782241, 268435470, 268435491, 268435520, 335544327,
    335544342, 335544362, 335544402, 335544482, 335544642, 335544962, 335545602, 335546882, 335549442, 335554562,
    335564802, 335585282, 335626242, 335708162, 335872002, 336199682, 336855042, 338165762, 340787202, 346030082,
    357040130, 357565442, 357826560, 357908482, 377487362, 402653195, 436207617, 503316482, 25, 37, 52, 67, 112,
    352, 1312, 2592, 5152, 10272, 20512, 40992, 81952, 163872, 174080, 327712, 655392, 1114112, 1310752, 2129920,
    2621472, 4210688, 5242912, 8396800, 10485792, 16781312, 20971552, 33556480, 41943072, 67109888, 83886112,
    134217741, 134217747, 134218240, 167772192, 178323457, 178782273, 192937985, 268435523, 268435584, 335544331,
    335544336, 357566466, 357826563, 402653203, 486539266, 534773763, 38, 69, 74, 104, 131, 224, 704, 2624, 5184,
    10304, 20544, 41024, 81984, 163904, 327744, 655424, 786432, 1179648, 1310784, 2162688, 2621504, 4227072,
    5242944, 8404992, 10485824, 16785408, 20971584, 33558528, 41943104, 67110912, 83886144, 134217763, 134218752,
    167772171, 167772181, 167772201, 167772224, 167772241, 167772321, 167772481, 167772801, 167773441, 167774721,
    167777281, 167782401, 167792641, 167813121, 167854081, 167936001, 168099841, 168427521, 169082881, 170393601,
    173015041, 178388993, 178782337, 188743681, 251658241, 268435482, 268435494, 268435587, 268435712, 335544339,
    335544352, 352321537, 357568514, 358612994, 402653219, 469762055, 533725187, 49, 70, 73, 133, 138, 148, 208,
    259, 448, 1408, 5248, 10368, 20608, 41088, 82048, 87040, 163968, 327808, 655488, 1310848, 2228224, 2621568,
    4259840, 5243008, 8421376, 10485888, 11173888, 16793600, 20971648, 33562624, 41943168, 67112960, 83886208,
    88080385, 134217753, 134217765, 134217795, 134219776, 167772167, 167772288, 176160770, 178520065, 178782465,
    178913280, 243269633, 268435526, 268435715, 268435968, 335544334, 335544355, 335544384, 352321544, 357572610,
    357826566, 402653251, 436207623, 444596226, 469762059, 492044291, 503316481, 50, 134, 137, 261, 266, 296,
    416, 515, 896, 2816, 20736, 41216, 82176, 164096, 327936, 655616, 1310976, 1572864, 2359296, 2621696,
    8454144, 10486016, 16809984, 20971776, 33570816, 44040193, 67117056, 83886336, 134217797, 167772416,
    178782721, 268435506, 268435530, 268435590, 268435971, 268436480, 335544387, 335544448, 352321540, 357826570,
    364904450, 402653315, 436207627, 469762067, 486539265 };

static int cbest_30[1023] = {
    1, 2, 541065219, 4, 811597826, 8, 405798913, 16, 32, 64, 743964675, 128, 913047554, 256, 541065218,
    270532609, 3, 512, 456523777, 541065217, 6, 811597827, 5, 676331523, 12, 129, 405798912, 258, 811597824, 9,
    24, 516, 1024, 202899456, 541065223, 541065283, 10, 1032, 946864130, 48, 2064, 101449728, 879230978, 17,
    4128, 541065227, 811597858, 96, 8256, 50724864, 811597830, 20, 16512, 405798915, 18, 33, 192, 2048, 33024,
    25362432, 405798929, 473432065, 541065235, 769327107, 66048, 439615489, 743964674, 811597834, 132096,
    12681216, 405798917, 34, 40, 65, 264192, 541065251, 528384, 6340608, 743964673, 811597842, 384, 1056768,
    405798921, 36, 66, 4096, 2113536, 3170304, 371982337, 743964683, 80, 4227072, 1585152, 8454144, 777781251,
    68, 130, 768, 16908288, 541065347, 743964679, 760872963, 913047555, 792576, 33816576, 811597890, 917340162,
    160, 67633152, 405798945, 72, 132, 257, 8192, 396288, 135266304, 913047552, 1536, 270532608, 811597954,
    913047558, 260, 198144, 405798977, 727056387, 136, 456523776, 541065475, 743964691, 193, 386, 99072, 514,
    3072, 405799041, 541065216, 929955842, 144, 320, 513, 16384, 49536, 743964707, 921501698, 997588994, 264,
    520, 772, 541065315, 676331522, 811598082, 7, 24768, 270532611, 456523779, 458670081, 913047562, 385, 6144,
    228261888, 338165761, 541065221, 541065411, 541065731, 743964739, 811597825, 13, 1028, 1544, 12384,
    270532641, 541065222, 541065282, 676331521, 14, 259, 270532613, 270532673, 405799169, 541065346, 811597874,
    904593410, 913047570, 946864131, 25, 272, 517, 770, 1025, 1040, 6192, 202899457, 541065473, 743964803,
    879230979, 11, 26, 131, 640, 1026, 1033, 3088, 12288, 405798914, 456523781, 541065226, 541065231, 541065281,
    676331539, 811597828, 811597859, 811597922, 811598338, 28, 49, 528, 2065, 3096, 101449729, 270532617,
    405798928, 464977921, 473432064, 541065225, 676331555, 710148099, 811597831, 913047586, 946864128, 161, 288,
    518, 1540, 2056, 4129, 32768, 114130944, 202899458, 202899464, 405798937, 439615488, 460750849, 498794497,
    541065243, 541065735, 541066243, 676331527, 811597955, 879230976, 22, 52, 97, 133, 1034, 1548, 6176, 8257,
    50724865, 101449732, 405798916, 456523785, 541065234, 541065299, 541066251, 769327106, 811597856, 811598080,
    21, 50, 56, 262, 322, 2066, 2080, 16513, 24576, 50724866, 101449730, 236716032, 270532625, 405798961,
    405799040, 405799425, 541065267, 541067283, 642514947, 811597835, 811597838, 879230986, 913047618, 946864146,
    1014497282, 19, 774, 2049, 3080, 4130, 33025, 25362433, 202899460, 202899520, 219807744, 541065287,
    541069347, 676331531, 743964931, 811597832, 811597866, 879230994, 980680706, 44, 98, 104, 137, 644, 1036,
    8258, 12352, 66049, 101449760, 405798919, 405798920, 456523793, 541065233, 541065250, 541073475, 591790083,
    769327105, 811597850, 811598342, 811598850, 946864134, 946864194, 112, 266, 387, 2052, 2068, 4112, 16514,
    132097, 12681217, 50724880, 118358016, 371982336, 405798933, 452296705, 541065239, 541081731, 743964672,
```

src/cauchy_best_r6.c lines 1741 to 1800

```
    811597843, 811598858, 879230982, 896139266, 913047682, 35, 41, 194, 524, 544, 1056, 1280, 2050, 4132, 6160,
    33026, 264193, 25362434, 25362440, 57065472, 109903872, 405798925, 405798931, 405800961, 439615493,
    473432067, 473432073, 541065291, 541067267, 541073411, 541098243, 545357827, 566427651, 770400259, 811597862,
    811599890, 811601922, 42, 88, 100, 145, 208, 4160, 8260, 16385, 24704, 66050, 528385, 6340609, 12681220,
    50724868, 405799171, 439615491, 439615497, 456523809, 541065249, 541131267, 743964682, 743964687, 811601954,
    946864138, 38, 224, 274, 2072, 16516, 132098, 1056769, 6340610, 12681218, 59179008, 101449736, 185991168,
    405799429, 405799937, 473432097, 507248641, 539017219, 541065410, 541197315, 743964681, 805355522, 811597840,
    811606082, 862322690, 37, 67, 196, 532, 576, 1288, 4097, 4136, 12320, 33028, 49152, 65536, 264194, 2113537,
    3170305, 25362436, 54951936, 202899472, 405798923, 405799945, 473432069, 490340353, 541065259, 541329411,
    743964677, 743964699, 743965187, 743965699, 765165571, 769327111, 777781250, 811597846, 811614338, 81, 176,
    388, 1048, 8224, 8264, 49408, 66052, 528386, 4227073, 50724872, 384663553, 402677761, 405798944, 405800977,
    456523841, 541593603, 743964678, 760872962, 782073859, 811598018, 811599874, 811630850, 815890434, 836960258,
    70, 290, 16520, 32770, 132100, 1056770, 1585153, 4227074, 8454145, 29589504, 92995584, 101449744, 270532705,
    270532737, 371982341, 405803041, 448069633, 536903683, 541065603, 542121987, 743964802, 760872961, 811597891,
    811597986, 811663874, 913047810, 917340163, 69, 84, 200, 548, 769, 4098, 4104, 4144, 8320, 24640, 33032,
    264196, 2113538, 3170306, 8454146, 16908289, 27475968, 28532736, 202899488, 371982339, 405799009, 405807169,
    541065255, 541065539, 541069315, 543178755, 544235523, 676331587, 743964929, 769327115, 771440643, 777781249,
    809517058, 810573826, 811729922, 76, 82, 416, 1064, 2112, 8272, 66056, 528388, 792577, 6340612, 8454148,
    16908290, 33816577, 404758529, 405798976, 405798993, 405815425, 456523905, 541065345, 777781255, 811862018,
    913047553, 913047556, 913048066, 946864162, 134, 448, 641, 2096, 2560, 2576, 16528, 132104, 1056772, 1585154,
    12681224, 14794752, 16908292, 46497792, 67633153, 405799105, 405831937, 407945217, 431161345, 542650371,
    549519363, 760872967, 811597888, 812126210, 879231010, 913047559, 917340160, 73, 392, 580, 776, 1088, 4100,
    8193, 16448, 33040, 65540, 98304, 264200, 396289, 2113540, 3170308, 13737984, 16908296, 25362448, 33816580,
    135266305, 371982401, 388890625, 405864961, 473432081, 541065987, 557973507, 676331651, 727056386, 742404099,
    743965191, 758824963, 769327123, 777781267, 807436290, 811598210, 812654594, 74, 140, 352, 1096, 1537, 8288,
    66064, 98816, 528392, 792578, 4227076, 6340616, 33816578, 33816584, 50724896, 380436481, 403718145,
    405286913, 405931009, 418480129, 439615505, 541065474, 541857795, 574881795, 727056385, 743964690, 743965707,
    162, 168, 261, 1538, 2128, 16544, 132112, 198145, 1056776, 1585156, 7397376, 12681232, 23248896, 33816592,
    67633154, 67633160, 101449792, 270532865, 405798947, 405798953, 406063105, 456524033, 541065379, 608698371,
    676331571, 743444483, 743964723, 743966739, 794689539, 811597906, 811597952, 879231042, 913047566, 152, 832,
    1152, 4192, 8194, 24577, 33056, 49280, 131072, 264208, 396290, 2113544, 3170312, 6868992, 14266368, 25362464,
    67633168, 135266306, 202899584, 371982345, 406327297, 541065351, 541461507, 743964715, 743968803, 756711427,
    769327139, 786235395, 813182978, 820051970, 915259394, 917876738, 929955843, 138, 896, 1160, 1282, 5152,
    8208, 66080, 99073, 131080, 528400, 792580, 4227080, 6340624, 50724928, 67633184, 135266320, 270532610,
    405799297, 439615521, 456523778, 458670080, 541066755, 541077507, 743964689, 743964706, 743964771, 743972931,
    811597894, 811598594, 828506114, 913047683, 917340166, 921501699, 923713538, 929955840, 946864258, 997588995,
    164, 268, 400, 515, 645, 784, 1290, 1552, 2192, 2580, 3073, 4224, 5160, 10320, 16576, 16640, 20640, 41280,
    49154, 82560, 132128, 165120, 198146, 330240, 660480, 1056784, 1320960, 1585160, 2641920, 3698688, 5283840,
    8454152, 10567680, 11624448, 12681248, 21135360, 42270720, 67633156, 84541440, 101449856, 135266336,
    169082880, 338165760, 405798949, 405798979, 405803009, 541065314, 541065479, 541065601, 541263363, 725008387,
    742924291, 743964695, 743981187, 777781259, 811598083, 812390402, 845414402, 879231106, 913047808, 921501696,
    991346690, 997588992, 321, 3074, 4256, 8196, 33088, 49537, 196608, 264224, 396292, 2113552, 3170320, 3434496,
    25362496, 135266308, 135266368, 270532640, 270532657, 363528193, 371982353, 410025985, 473432129, 541065355,
    541065537, 541065729, 676331779, 743964867, 743966723, 743997699, 745037827, 748257283, 760872971, 769327171,
    811603970, 912267266, 913047563, 195, 265, 280, 521, 704, 773, 3076, 5120, 8384, 66112, 99074, 197632,
    528416, 792584, 4227088, 6340640, 50724992, 270532612, 270532672, 405799168, 414253057, 439615553, 541065220,
    541065730, 541164291, 727056419, 743964705, 743964738, 744030723, 811597898, 811597958, 811606018, 811994114,
    912787458, 913047560, 917340170, 148, 209, 324, 522, 1664, 24769, 32896, 132160, 198148, 262160, 1056800,
    1585168, 1849344, 5812224, 8454160, 12681280, 270532736, 270532801, 270533121, 405798981, 405799043,
    405799681, 405801985, 422707201, 541065313, 541068291, 541089795, 676331520, 722894851, 727056391, 741916675,
    743708675, 744096771, 811599362, 913047578, 913048070, 913048578, 918396930, 929955850, 146, 225, 418, 1546,
    2176, 2564, 6145, 8200, 10304, 16386, 33152, 49538, 98560, 264256, 396296, 1717248, 2113568, 3170336,
    7133184, 16908304, 25362560, 135266312, 202899712, 228261889, 371982369, 405799057, 406591489, 456523780,
    473432193, 541065363, 541081603, 541114755, 735510531, 744228867, 760872979, 769327235, 811597875, 811598019,
    811598086, 811598208, 811795970, 879231002, 904593411, 913047571, 913047574, 913048586, 197, 336, 1029, 1545,
    1568, 1792, 12385, 49153, 66176, 98308, 99076, 792592, 4227104, 270532615, 270532616, 456524289, 464977920,
    541065409, 541065472, 541065602, 541065991, 676331547, 743964737, 811597962, 811597987, 811598336, 811610114,
    912527362, 913049618, 917340178, 946864386 };

static int cbest_31[1023] = {
```

src/cauchy_best_r6.c lines 1801 to 1860

```
    1, 2, 1073741828, 4, 536870914, 8, 268435457, 16, 32, 1207959556, 64, 128, 603979778, 256, 512, 301989889,
    1024, 2048, 4096, 1224736772, 8192, 16384, 32768, 612368386, 65536, 131072, 262144, 306184193, 524288,
    1048576, 2097152, 4194304, 1226833924, 8388608, 16777216, 33554432, 67108864, 613416962, 134217728,
    268435456, 306708481, 536870912, 3, 1073741824, 1073741829, 5, 536870915, 1073741830, 6, 9, 18, 36, 72, 144,
    288, 576, 1152, 2304, 4608, 9216, 18432, 36864, 73728, 147456, 294912, 589824, 1179648, 2359296, 4718592,
    9437184, 18874368, 37748736, 75497472, 150994944, 301989888, 1610612742, 10, 17, 268435459, 536870918,
    603979776, 1073741836, 1227096068, 1342177285, 12, 33, 268435461, 536870922, 805306371, 1073741844,
    1207959552, 1207959557, 20, 34, 65, 268435465, 536870930, 1073741860, 1207959558, 24, 66, 129, 268435473,
    536870946, 603979779, 1073741892, 1744830470, 40, 68, 130, 257, 268435489, 536870978, 1073741956, 1207959564,
    1476395013, 1677721606, 48, 132, 258, 513, 268435521, 536871042, 603979782, 613548034, 1073742084,
    1207959572, 80, 136, 260, 514, 1025, 268435585, 301989891, 536871170, 603979786, 872415235, 1073742340,
    1207959588, 1375731717, 96, 264, 516, 1026, 2049, 268435713, 301989893, 536871426, 603979794, 838860803,
    1073742852, 1207959620, 160, 272, 520, 1028, 2050, 4097, 268435969, 301989897, 306184192, 536871938,
    603979810, 612368384, 1073743876, 1207959684, 1224736768, 1224736773, 1811939334, 192, 528, 1032, 2052, 4098,
    8193, 153092096, 268436481, 301989905, 536872962, 603979842, 1073745924, 1207959812, 1224736774, 320, 544,
    1040, 2056, 4100, 8194, 16385, 76546048, 268437505, 301989921, 306774017, 536875010, 603979906, 1073750020,
    1207960068, 1509949445, 1761607686, 384, 1056, 2064, 4104, 8196, 16386, 32769, 38273024, 268439553,
    301989953, 536879106, 603980034, 612368387, 1073758212, 1207960580, 1224736780, 1493172229, 640, 1088, 2080,
    4112, 8200, 16388, 32770, 65537, 19136512, 268443649, 301990017, 536887298, 603980290, 905969667, 1073774596,
    1207961604, 1224736788, 1686110214, 768, 2112, 4128, 8208, 16392, 32772, 65538, 131073, 9568256, 268451841,
    301990145, 536903682, 603980802, 612368390, 1073807364, 1207963652, 1224736804, 1280, 2176, 4160, 8224,
    16400, 32776, 65540, 131074, 262145, 4784128, 268468225, 301990401, 536936450, 603981826, 612368394,
    880803843, 1073872900, 1207967748, 1224736836, 1536, 4224, 8256, 16416, 32784, 65544, 131076, 262146, 524289,
    2392064, 268500993, 301990913, 306184195, 537001986, 603983874, 612368402, 1074003972, 1207975940,
    1224736900, 1379926021, 1828716550, 2560, 4352, 8320, 16448, 32800, 65552, 131080, 262148, 524290, 1048577,
    1196032, 268566529, 301991937, 306184197, 537133058, 603987970, 612368418, 843055107, 1074266116, 1207992324,
    1224737028, 1227128836, 1820327942, 3072, 8448, 16512, 32832, 65568, 131088, 262152, 524292, 598016, 1048578,
    2097153, 268697601, 301993985, 306184201, 537395202, 603996162, 612368450, 1074790404, 1208025092,
    1224737284, 1526726661, 5120, 8704, 16640, 32896, 65600, 131104, 262160, 299008, 524296, 1048580, 2097154,
    4194305, 268959745, 301998081, 306184209, 537919490, 604012546, 612368514, 1075838980, 1208090628,
    1224737796, 1226833920, 1226833925, 6144, 16896, 33024, 65664, 131136, 149504, 262176, 524304, 1048584,
    2097156, 4194306, 8388609, 269484033, 302006273, 306184225, 538968066, 604045314, 612368642, 613416960,
    1077936132, 1208221700, 1224738820, 1226833926, 1514143749, 10240, 17408, 33280, 65792, 74752, 131200,
    262208, 524320, 1048592, 2097160, 4194308, 8388610, 16777217, 270532609, 302022657, 306184257, 306708480,
    541065218, 604110850, 612368898, 914358275, 1082130436, 1208483844, 1224740868, 1763704838, 12288, 33792,
    37376, 66048, 131328, 262272, 524352, 1048608, 2097168, 4194312, 8388612, 16777218, 33554433, 272629761,
    302055425, 306184321, 545259522, 604241922, 612369410, 910163971, 1090519044, 1209008132, 1224744964,
    1226833932, 1495269381, 18688, 20480, 34816, 66560, 131584, 262400, 524416, 1048640, 2097184, 4194320,
    8388616, 16777220, 33554434, 67108865, 153354240, 276824065, 302120961, 306184449, 553648130, 604504066,
    612370434, 613416963, 613564418, 1107296260, 1210056708, 1224753156, 1226833940, 9344, 24576, 67584, 132096,
    262656, 524544, 1048704, 2097216, 4194336, 8388624, 16777224, 33554436, 67108866, 134217729, 285212673,
    302252033, 306184705, 570425346, 605028354, 612372482, 1140850692, 1212153860, 1224769540, 1226833956,
    1687158790, 1837105158, 4672, 40960, 69632, 133120, 263168, 524800, 1048832, 2097280, 4194368, 8388640,
    16777232, 33554440, 67108868, 76677120, 134217730, 302514177, 306185217, 606076930, 612376578, 613416966,
    1216348164, 1224802308, 1226833988, 2336, 49152, 135168, 264192, 525312, 1049088, 2097408, 4194432, 8388672,
    16777248, 33554448, 67108872, 134217732, 268435458, 303038465, 306186241, 335544321, 608174082, 612384770,
    613416970, 671088642, 881852419, 1224867844, 1226834052, 1342177284, 1830813702, 1168, 81920, 139264, 266240,
    526336, 1049600, 2097664, 4194560, 8388736, 16777280, 33554464, 38338560, 67108880, 134217736, 268435460,
    304087041, 306188289, 402653185, 536870913, 612401154, 613416978, 805306370, 1224998916, 1226834180,
    1241513988, 1530920965, 584, 98304, 270336, 528384, 1050624, 2098176, 4194816, 8388864, 16777344, 33554496,
    67108896, 134217744, 268435464, 306192385, 306708483, 612433922, 613416994, 620756994, 1225261060,
    1226834436, 1275068420, 1380450309, 1528823813, 1610612740, 1821376518, 292, 163840, 278528, 532480, 1052672,
    2099200, 4195328, 8389120, 16777472, 19169280, 33554560, 67108928, 134217760, 268435472, 306200577,
    306708485, 306782209, 310378497, 536870916, 536870928, 612499458, 613417026, 637534210, 843579395,
    1073741825, 1225785348, 1226834948, 146, 196608, 540672, 1056768, 2101248, 4196352, 8389632, 16777728,
    33554688, 67108992, 134217792, 268435488, 306216961, 306708489, 318767105, 536870920, 612630530, 613417090,
    805306369, 918552579, 1073741826, 1073741831, 1073741856, 1226835972, 1476395012, 7, 19, 37, 73, 145, 289,
    577, 1153, 2305, 4609, 9217, 18433, 36865, 73729, 147457, 294913, 327680, 557056, 589825, 1064960, 1179649,
    2105344, 2359297, 4198400, 4718593, 8390656, 9437185, 9584640, 16778240, 18874369, 33554944, 37748737,
    67109120, 75497473, 134217856, 150994945, 268435520, 306249729, 306708497, 612892674, 613417218, 738197506,
```

src/cauchy_best_r6.c lines 1861 to 1920

```
    1226838020, 1228931076, 1610612738, 1610612743, 11, 38, 74, 290, 578, 1154, 2306, 4610, 9218, 18434, 36866,
    73730, 147458, 294914, 393216, 589826, 1081344, 1179650, 2113536, 2359298, 4202496, 4718594, 8392704,
    9437186, 16779264, 18874370, 33555456, 37748738, 67109376, 75497474, 134217984, 150994946, 268435584,
    301989890, 306315265, 306708513, 369098753, 536870919, 536870944, 603979777, 613417474, 872415234, 915406851,
    1073741832, 1073741837, 1073741846, 1073741900, 1073741972, 1073742116, 1073742404, 1073742980, 1073744132,
    1073746436, 1073751044, 1073760260, 1073778692, 1073815556, 1073889284, 1074036740, 1074331652, 1074921476,
    1076101124, 1078460420, 1083179012, 1092616196, 1111490564, 1149239300, 1226842116, 1227096064, 1227096069,
    1233125380, 1342177281, 1375731716, 1514668037, 1744830468, 13, 22, 76, 148, 580, 1156, 2308, 4612, 9220,
    18436, 36868, 73732, 147460, 294916, 589828, 655360, 1114112, 1179652, 2129920, 2359300, 4210688, 4718596,
    4792320, 8396800, 9437188, 16781312, 18874372, 33556480, 37748740, 67109888, 75497476, 134218240, 150994948,
    268435712, 301989892, 306446337, 306708545, 436207617, 536870923, 536870950, 536870976, 536870986, 536871058,
    536871202, 536871490, 536872066, 536873218, 536875522, 536880130, 536889346, 536907778, 536944642, 537018370,
    537165826, 537460738, 538050562, 539230210, 541589506, 546308098, 555745282, 574619650, 613417986, 614465538,
    687865858, 838860802, 1073741838, 1073741840, 1073741845, 1207959553, 1226850308, 1227096070, 1342177287,
    1677721604, 14, 21, 26, 35, 44, 152, 296, 1160, 2312, 4616, 9224, 18440, 36872, 73736, 147464, 294920,
    589832, 786432, 1179656, 2162688, 2359304, 4227072, 4718600, 8404992, 9437192, 16785408, 18874376, 33558528,
    37748744, 67110912, 75497480, 134218752, 150994952, 268435463, 268435475, 268435493, 268435529, 268435601,
    268435745, 268435968, 268436033, 268436609, 268437761, 268440065, 268444673, 268453889, 268472321, 268509185,
    268582913, 268730369, 269025281, 269615105, 270794753, 273154049, 277872641, 287309825, 301989896, 306708609,
    343932929, 419430401, 536870931, 536871040, 603979780, 603979792, 613419010, 613548032, 616562690, 910688259,
    1073741861, 1207959554, 1207959559, 1207959584, 1226866692, 1258291204, 1610612750, 1763966982, 1839202310,
    1879048199, 25, 52, 67, 88, 304, 592, 2320, 4624, 9232, 18448, 36880, 73744, 147472, 294928, 589840, 1179664,
    1310720, 2228224, 2359312, 2396160, 4259840, 4718608, 8421376, 9437200, 16793600, 18874384, 33562624,
    37748752, 67112960, 75497488, 134219776, 150994960, 268435467, 268436480, 301989904, 306708737, 307232769,
    536870926, 536870947, 536871168, 603979784, 613421058, 805306375, 872415233, 1073741862, 1073741888,
    1073741893, 1226899460, 1227096076, 1291845636, 1342177293, 1495531525, 1610612758, 1744830466, 1744830471,
    1838153734, 41, 50, 69, 104, 131, 176, 608, 18464, 73760, 147488, 589856, 1179680, 1572864, 2359328, 4325376,
    4718624, 8454144, 9437216, 16809984, 18874400, 33570816, 67117056, 75497504, 134221824, 301989920, 306708993,
    306774016, 308281345, 536870934, 536870979, 629145602, 805306379, 1073741894, 1073741952, 1207959560,
    1207959700, 1207959844, 1207960708, 1207961860, 1207964164, 1207968772, 1207977988, 1207996420, 1208033284,
    1208107012, 1208549380, 1209139204, 1210318852, 1212678148, 1217396740, 1226964996, 1283457028, 1342177301,
    1476395009, 1509949444, 1610612774, 1677721607 };

static unsigned int cbest_32[1023] = {
    1, 2149580803, 2, 4, 3224371202, 8, 1612185601, 16, 32, 2955673603, 64, 128, 256, 3627417602, 512, 1024,
    2149580802, 1074790401, 1813708801, 3, 2149580801, 2048, 3224371203, 6, 2686976003, 5, 12, 1612185600,
    3224371200, 2149580807, 9, 24, 806092800, 3761766402, 10, 1025, 4096, 3493068802, 48, 2050, 403046400,
    2149580811, 2149581315, 17, 4100, 3056435203, 3224371206, 96, 8200, 201523200, 20, 16400, 1612185603,
    2149580819, 3224371458, 18, 33, 192, 32800, 100761600, 1880883201, 2955673602, 3224371210, 8192, 65600,
    1746534401, 384, 131200, 50380800, 1612185605, 2149580835, 34, 40, 65, 262400, 1612185729, 2955673601,
    3224371218, 768, 524800, 25190400, 1049600, 1612185609, 2149580867, 36, 66, 129, 1536, 2099200, 12595200,
    1477836801, 3224371234, 80, 16384, 4198400, 3090022403, 6297600, 8396800, 1612185617, 2149580931, 2955673607,
    2955673667, 3022848003, 3627417603, 68, 130, 257, 16793600, 3224371266, 3677798402, 3148800, 33587200, 160,
    3072, 67174400, 1612185633, 2149581059, 2955673611, 3627417600, 72, 132, 258, 513, 1574400, 134348800,
    3224371330, 32768, 268697600, 2888499203, 787200, 537395200, 1612185665, 2955673619, 136, 260, 320, 514,
    6144, 1074790400, 1813708800, 3627417634, 393600, 3627417606, 3694592002, 2149581827, 2955673635, 3661004802,
    3963289602, 144, 264, 516, 1026, 1537, 3074, 196800, 2149580800, 3224371714, 640, 12288, 65536, 3627417610,
    98400, 1612185857, 2149581571, 2686976002, 272, 520, 1028, 2049, 6148, 906854400, 3224372226, 7, 49200,
    1074790403, 1813708803, 1838899201, 2149580805, 2149582339, 3224371201, 3593830402, 3627417618, 1280, 2052,
    3073, 24576, 1343488001, 1612186113, 1813708817, 2149580806, 2149582851, 2686976001, 2955673731, 13, 288,
    528, 1032, 12296, 24600, 3224371586, 3761766403, 14, 131072, 1074790405, 1813708805, 3493068803, 25, 4098,
    6146, 12300, 806092801, 1612186625, 1847296001, 2149580810, 2149580815, 2149581314, 2955673859, 3224371204,
    3224371970, 11, 26, 544, 1040, 4097, 24592, 49152, 453427200, 1074790657, 1612185602, 1830502401, 1981644801,
    2149580809, 2149581826, 2821324803, 3056435202, 3224371207, 3224373250, 3761766400, 28, 49, 2051, 2056, 4104,
    6150, 403046401, 1074790409, 1074790913, 1813708809, 1880883200, 2149580827, 2149582849, 2686976007,
    3493068800, 3627417666, 4101, 12292, 806092802, 1612185793, 1746534400, 2149580818, 2149581313, 2149584899,
    2686976131, 2955674115, 3224371459, 22, 52, 97, 576, 1027, 1056, 2560, 3075, 8201, 49184, 201523201,
    1612185604, 2149580851, 2552627203, 2686976259, 3056435201, 3224371211, 3224371214, 4030464002, 21, 50, 56,
    8196, 16401, 98304, 262144, 403046402, 940441600, 1074790417, 1612185728, 1612185985, 2149584903, 2686976011,
    3224371208, 3224372227, 3627417730, 3896115202, 19, 193, 4102, 24584, 32801, 100761601, 806092804, 806092864,
```

```
    873267200, 1612187649, 1796915201, 2149580817, 2149580834, 2149580899, 2149582338, 2149589003, 2351104003,
    2955674627, 3056435211, 3224371226, 3224371456, 3224373248, 3761766406, 44, 98, 104, 1029, 1088, 2064, 8193,
    8202, 8208, 65601, 98368, 201523202, 226713600, 403046432, 1612185607, 1612185608, 2149580823, 2149597203,
    2955673600, 3224371219, 3224375298, 3493068806, 3493068866, 3560243202, 3761766530, 112, 385, 2054, 8194,
    16402, 131201, 50380801, 201523216, 403046404, 470220800, 1074790433, 1477836800, 1612186624, 1813708833,
    2149580995, 2149581319, 2149613603, 2250342403, 2686976019, 3224371250, 3493068930, 3627417858, 35, 41, 194,
    1281, 4112, 32802, 49168, 196608, 262401, 100761602, 100761608, 436633600, 806092808, 806093312, 1612185613,
    1880883203, 2149580833, 2149580866, 2149581443, 2149588995, 2149646403, 2955673699, 3056435207, 3224375302,
    3761766410, 3761766914, 42, 88, 100, 208, 769, 1033, 1152, 2080, 8204, 16392, 65602, 196736, 524801,
    25190401, 50380804, 201523204, 403046656, 1074791169, 1612185616, 1746534403, 2149581187, 2149712003,
    2199961603, 3069030403, 3224371216, 3224371235, 3224371298, 3224379402, 3425894402, 3493068810, 38, 224, 386,
    2058, 2562, 5120, 16404, 131202, 524288, 1049601, 25190402, 50380802, 201523328, 235110400, 403046408,
    738918400, 1074790465, 1528217601, 1612185625, 1813708865, 2015232001, 2149580843, 2149581323, 2149843203,
    2686976035, 2955673605, 2955673795, 3073228803, 3090022402, 3224371222, 3224371462, 3224371522, 3224387602,
    3627418114, 37, 67, 196, 4108, 16416, 32804, 98336, 262402, 2099201, 12595201, 100761604, 100761664,
    218316800, 806092816, 1612185611, 1612185731, 1612189697, 1746534433, 1880883205, 1880883265, 1948057601,
    2147485699, 2149580865, 2149580930, 2150105603, 2174771203, 2955673606, 2955673666, 2955675651, 3022848002,
    3064832003, 3140403203, 3224371394, 3224404002, 3325132802, 3761766418, 81, 176, 416, 770, 1041, 2112, 5124,
    16385, 65604, 393216, 393472, 524802, 2099202, 4198401, 50380832, 113356800, 201523208, 1612185632,
    1612185649, 1612187651, 1746534405, 1746534465, 2150630403, 3223323650, 3224371232, 3224371267, 3224379394,
    3224436802, 3493068818, 3677798403, 70, 388, 448, 2066, 16408, 131204, 1049602, 4198402, 6297601, 8396801,
    25190416, 117555200, 369459200, 403046416, 1074790529, 1612185761, 1612189701, 1780121601, 1813708929,
    1880883457, 2149580839, 2149581331, 2151684099, 2162176003, 2686976067, 3090022401, 3224371466, 3224502402,
    3274752002, 3627418626, 69, 84, 131, 200, 1538, 4116, 16388, 32784, 32808, 196672, 262404, 4198404, 8396802,
    12595202, 12595208, 16793601, 109158400, 806092832, 1477836803, 1611661825, 1612185697, 1612185733,
    1612193801, 1880883209, 2149580883, 2149580929, 2149581058, 2149597187, 2153779203, 2686976387, 2955673610,
    2955673615, 2955673665, 3022848001, 3056435219, 3224371242, 3224633602, 3627417601, 3761766434, 76, 82, 352,
    772, 832, 1057, 2176, 4128, 8216, 8224, 16386, 65608, 524804, 786944, 3148801, 6297604, 8396804, 25190404,
    33587201, 1612185621, 1612185664, 1612202001, 1746534409, 2148535299, 2157977603, 2955673609, 2955674626,
    3221228546, 3222276098, 3224371264, 3224371331, 3224896002, 3249561602, 3493068834, 3677798400, 134, 161,
    392, 896, 2082, 32832, 131208, 786432, 1048576, 1049604, 3148802, 6297602, 8396808, 16793604, 50380808,
    58777600, 67174401, 184729600, 1477836833, 1612185619, 1612218401, 1712947201, 1813709057, 2149580871,
    2149581347, 2166374403, 2888499202, 2955673627, 2955675649, 3048038403, 3224371474, 3627417650, 3677798406,
    73, 133, 259, 1540, 4132, 10248, 32816, 262408, 393344, 1574401, 2099204, 12595204, 16793602, 16793608,
    54579200, 100761616, 134348801, 1477836805, 1545011201, 1611138049, 1612185737, 1612193793, 1612251201,
    1880883217, 2149581057, 2183168003, 2955411715, 2955673618, 2955673683, 2955677699, 3022848019, 3056435235,
    3090022435, 3224371238, 3224372738, 3226474498, 3236966402, 3761766466, 74, 140, 704, 776, 1089, 1664, 2304,
    3076, 4160, 8232, 10240, 32769, 65616, 524808, 1573888, 16793616, 25190408, 33587202, 33587208, 56678400,
    201523232, 268697601, 1511424001, 1610614273, 1612316801, 1662566401, 1746534417, 2149580963, 2149583875,
    2216755203, 2955673651, 3022848035, 3090022407, 3157196803, 3224371282, 3224371328, 3224372482, 3224387586,
    3228569602, 3627417604, 3627417635, 3627417698, 162, 168, 262, 400, 1792, 2114, 16432, 65568, 131216, 787201,
    1049608, 29388800, 33587216, 50380816, 67174402, 92364800, 403046464, 537395201, 1612185635, 1612185641,
    1612448001, 1813709313, 2149580875, 2149580935, 2149581379, 2149583363, 2151155203, 2283929603, 2888499201,
    2955673671, 2955677703, 3022848007, 3090022531, 3123609603, 3223848450, 3224371490, 3232768002, 3627417607,
    3627419650, 3694592003, 137, 152, 261, 321, 515, 1544, 4164, 6145, 262416, 786688, 1572864, 1574402, 2099208,
    27289600, 33587232, 67174416, 100761632, 134348802, 806092928, 1477836809, 1477837313, 1612185745,
    1612186369, 1637376001, 1880883233, 2149613571, 2418278403, 2955149827, 2955673617, 2955673634, 2955681803,
    3056435267, 3224371270, 3241164802, 3661004803, 3963289603, 138, 268, 784, 1153, 1408, 4224, 8264, 32770,
    32776, 65632, 65664, 393601, 524816, 3147776, 3148804, 4198408, 33587204, 67174432, 201523264, 268697602,
    1612185856, 1612186241, 2150368003, 2913689603, 2955673623, 2955690003, 3039641603, 3090022411, 3224371715,
    3257958402, 3493068994, 3627417632, 164, 518, 2178, 5121, 16448, 16464, 20496, 131232, 787202, 1049616,
    6297608, 14694400, 46182400, 67174404, 67174464, 134348832, 403046528, 537395202, 1074791425, 1612185637,
    1612185667, 1612187137, 1613237249, 1614284801, 1624780801, 1813709825, 2149580939, 2149581063, 2149581123,
    2149586947, 2686976515, 2888499219, 2954887939, 2955673675, 2955706403, 3022848011, 3224371362, 3224374274,
    3291545602, 3627286658, 3627417611, 3627417614, 3694592000, 145, 265, 280, 322, 517, 1552, 3080, 4228, 6152,
    32864, 196801, 262432, 1573376, 1574404, 2097152, 2099216, 12595216, 13644800, 134348804, 134348864,
    806093056, 1074790402, 1444249601, 1477836817, 1612185681, 1612201985, 1616384001, 1813708802, 1838899200,
    2149581570, 2149581825, 2149583873, 2149974403, 2955673633, 2955681795, 2955739203, 3056435331, 3224371274,
    3224371334, 3358720002, 3627417608, 3627417642, 3627418627, 3661004800, 3677798410, 3679901698, 3686195202,
    3761766658, 3963289600, 266, 336, 524, 641, 800, 3328, 4352, 5125, 8256, 8328, 10250, 12289, 20500, 32772,
```

src/cauchy_best_r6.c lines 1981 to 1986

```
    41000, 65537, 82000, 131136, 164000, 328000, 393602, 524832, 656000, 1312000, 2624000, 3148808, 4198416,
    5248000, 10496000, 20992000, 25190432, 28339200, 41984000, 83968000, 134348928, 167936000, 268697604,
    268697664, 335872000, 671744000, 1343488000, 1611924225, 1612186112, 1618483201, 1620582401, 1813708816,
    2149583361, 2955673987, 2955804803, 3006054403, 3090022419, 3224371712, 3224403970, 3493069058, 3627417626,
    3627419648, 3681996802, 3719782402, 148, 304, 2306, 3584, 12290, 16528, 98401, 131264, 787204, 1049632,
    3145728, 6297616, 8396816, 23091200, 50380864, 67174408, 268697728, 1074790785, 1612185669, 1813708825 };
```

src/galois.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "galois.h"

#define MAX_GF_INSTANCES 64
gf_t *gfp_array[MAX_GF_INSTANCES] = { 0 };
int  gfp_is_composite[MAX_GF_INSTANCES] = { 0 };

gf_t *galois_get_field_ptr(int w)
{
  if (gfp_array[w] != NULL) {
```

src/galois.c lines 61 to 120

```
    return gfp_array[w];
  }

  return NULL;
}

gf_t* galois_init_field(int w,
                        int mult_type,
                        int region_type,
                        int divide_type,
                        uint64_t prim_poly,
                        int arg1,
                        int arg2)
{
  int scratch_size;
  void *scratch_memory;
  gf_t *gfp;

  if (w <= 0 || w > 32) {
    fprintf(stderr, "ERROR -- cannot init default Galois field for w=%d\n", w);
    exit(1);
  }

  gfp = (gf_t *) malloc(sizeof(gf_t));
  if (!gfp) {
    fprintf(stderr, "ERROR -- cannot allocate memory for Galois field w=%d\n", w);
    exit(1);
  }

  scratch_size = gf_scratch_size(w, mult_type, region_type, divide_type, arg1, arg2);
  if (!scratch_size) {
    fprintf(stderr, "ERROR -- cannot get scratch size for base field w=%d\n", w);
    exit(1);
  }

  scratch_memory = malloc(scratch_size);
  if (!scratch_memory) {
    fprintf(stderr, "ERROR -- cannot get scratch memory for base field w=%d\n", w);
    exit(1);
  }

  if(!gf_init_hard(gfp,
                   w,
                   mult_type,
                   region_type,
                   divide_type,
                   prim_poly,
                   arg1,
                   arg2,
                   NULL,
                   scratch_memory))
  {
    fprintf(stderr, "ERROR -- cannot init default Galois field for w=%d\n", w);
    exit(1);
  }

  gfp_is_composite[w] = 0;
  return gfp;
}
```

src/galois.c lines 121 to 180

```
gf_t* galois_init_composite_field(int w,
                                int region_type,
                                int divide_type,
                                int degree,
                                gf_t* base_gf)
{
  int scratch_size;
  void *scratch_memory;
  gf_t *gfp;

  if (w <= 0 || w > 32) {
    fprintf(stderr, "ERROR -- cannot init composite field for w=%d\n", w);
    exit(1);
  }

  gfp = (gf_t *) malloc(sizeof(gf_t));
  if (!gfp) {
    fprintf(stderr, "ERROR -- cannot allocate memory for Galois field w=%d\n", w);
    exit(1);
  }

  scratch_size = gf_scratch_size(w, GF_MULT_COMPOSITE, region_type, divide_type, degree, 0);
  if (!scratch_size) {
    fprintf(stderr, "ERROR -- cannot get scratch size for composite field w=%d\n", w);
    exit(1);
  }

  scratch_memory = malloc(scratch_size);
  if (!scratch_memory) {
    fprintf(stderr, "ERROR -- cannot get scratch memory for composite field w=%d\n", w);
    exit(1);
  }

  if(!gf_init_hard(gfp,
                   w,
                   GF_MULT_COMPOSITE,
                   region_type,
                   divide_type,
                   0,
                   degree,
                   0,
                   base_gf,
                   scratch_memory))
  {
    fprintf(stderr, "ERROR -- cannot init default composite field for w=%d\n", w);
    exit(1);
  }
  gfp_is_composite[w] = 1;
  return gfp;
}

int galois_init_default_field(int w)
{
  if (gfp_array[w] == NULL) {
    gfp_array[w] = (gf_t*)malloc(sizeof(gf_t));
    if(gfp_array[w] == NULL)
      return ENOMEM;
    if (!gf_init_easy(gfp_array[w], w))
      return EINVAL;
  }
```

```
  return 0;
}

int galois_uninit_field(int w)
{
  int ret = 0;
  if (gfp_array[w] != NULL) {
    int recursive = 1;
    ret = gf_free(gfp_array[w], recursive);
    free(gfp_array[w]);
    gfp_array[w] = NULL;
  }
  return ret;
}

static void galois_init(int w)
{
  if (w <= 0 || w > 32) {
    fprintf(stderr, "ERROR -- cannot init default Galois field for w=%d\n", w);
    exit(1);
  }

  switch (galois_init_default_field(w)) {
  case ENOMEM:
    fprintf(stderr, "ERROR -- cannot allocate memory for Galois field w=%d\n", w);
    exit(1);
    break;
  case EINVAL:
    fprintf(stderr, "ERROR -- cannot init default Galois field for w=%d\n", w);
    exit(1);
    break;
  }
}


static int is_valid_gf(gf_t *gf, int w)
{
  // TODO: I assume we may eventually
  // want to do w=64 and 128, so w
  // will be needed to perform this check
  (void)w;

  if (gf == NULL) {
    return 0;
  }
  if (gf->multiply.w32 == NULL) {
    return 0;
  }
  if (gf->multiply_region.w32 == NULL) {
    return 0;
  }
  if (gf->divide.w32 == NULL) {
    return 0;
  }
  if (gf->inverse.w32 == NULL) {
    return 0;
  }
  if (gf->extract_word.w32 == NULL) {
    return 0;
  }
```

```
  return 1;
}

void galois_change_technique(gf_t *gf, int w)
{
  if (w <= 0 || w > 32) {
    fprintf(stderr, "ERROR -- cannot support Galois field for w=%d\n", w);
    exit(1);
  }

  if (!is_valid_gf(gf, w)) {
    fprintf(stderr, "ERROR -- overriding with invalid Galois field for w=%d\n", w);
    exit(1);
  }

  if (gfp_array[w] != NULL) {
    gf_free(gfp_array[w], gfp_is_composite[w]);
  }

  gfp_array[w] = gf;
}

int galois_single_multiply(int x, int y, int w)
{
  if (x == 0 || y == 0) return 0;

  if (gfp_array[w] == NULL) {
    galois_init(w);
  }

  if (w <= 32) {
    return gfp_array[w]->multiply.w32(gfp_array[w], x, y);
  } else {
    fprintf(stderr, "ERROR -- Galois field not implemented for w=%d\n", w);
    return 0;
  }
}

int galois_single_divide(int x, int y, int w)
{
  if (x == 0) return 0;
  if (y == 0) return -1;

  if (gfp_array[w] == NULL) {
    galois_init(w);
  }

  if (w <= 32) {
    return gfp_array[w]->divide.w32(gfp_array[w], x, y);
  } else {
    fprintf(stderr, "ERROR -- Galois field not implemented for w=%d\n", w);
    return 0;
  }
}

void galois_w08_region_multiply(char *region,      /* Region to multiply */
                                  int multby,       /* Number to multiply by */
                                  int nbytes,        /* Number of bytes in region */
                                  char *r2,          /* If r2 != NULL, products go here */
```

```
                                    int add)
{
  if (gfp_array[8] == NULL) {
    galois_init(8);
  }
  gfp_array[8]->multiply_region.w32(gfp_array[8], region, r2, multby, nbytes, add);
}

void galois_w16_region_multiply(char *region,      /* Region to multiply */
                                    int multby,       /* Number to multiply by */
                                    int nbytes,        /* Number of bytes in region */
                                    char *r2,          /* If r2 != NULL, products go here */
                                    int add)
{
  if (gfp_array[16] == NULL) {
    galois_init(16);
  }
  gfp_array[16]->multiply_region.w32(gfp_array[16], region, r2, multby, nbytes, add);
}


void galois_w32_region_multiply(char *region,      /* Region to multiply */
                                    int multby,       /* Number to multiply by */
                                    int nbytes,        /* Number of bytes in region */
                                    char *r2,          /* If r2 != NULL, products go here */
                                    int add)
{
  if (gfp_array[32] == NULL) {
    galois_init(32);
  }
  gfp_array[32]->multiply_region.w32(gfp_array[32], region, r2, multby, nbytes, add);
}

void galois_w8_region_xor(void *src, void *dest, int nbytes)
{
  if (gfp_array[8] == NULL) {
    galois_init(8);
  }
  gfp_array[8]->multiply_region.w32(gfp_array[32], src, dest, 1, nbytes, 1);
}

void galois_w16_region_xor(void *src, void *dest, int nbytes)
{
  if (gfp_array[16] == NULL) {
    galois_init(16);
  }
  gfp_array[16]->multiply_region.w32(gfp_array[16], src, dest, 1, nbytes, 1);
}

void galois_w32_region_xor(void *src, void *dest, int nbytes)
{
  if (gfp_array[32] == NULL) {
    galois_init(32);
  }
  gfp_array[32]->multiply_region.w32(gfp_array[32], src, dest, 1, nbytes, 1);
}

void galois_region_xor(char *src, char *dest, int nbytes)
{
  if (nbytes >= 16) {
```

src/galois.c lines 361 to 376

```
    galois_w32_region_xor(src, dest, nbytes);
  } else {
    int i = 0;
    for (i = 0; i < nbytes; i++) {
      *dest ^= *src;
      dest++;
      src++;
    }
  }
}

int galois_inverse(int y, int w)
{
  if (y == 0) return -1;
  return galois_single_divide(1, y, w);
}
```

src/jerasure.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "galois.h"
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static double jerasure_total_xor_bytes = 0;
static double jerasure_total_gf_bytes = 0;
static double jerasure_total_memcpy_bytes = 0;

void jerasure_print_matrix(int *m, int rows, int cols, int w)
```

src/jerasure.c lines 61 to 120

```
{
  int i, j;
  int fw;
  char s[30];
  unsigned int w2;

  if (w == 32) {
    fw = 10;
  } else {
    w2 = (1 << w);
    sprintf(s, "%u", w2-1);
    fw = strlen(s);
  }

  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      if (j != 0) printf(" ");
      printf("%*u", fw, m[i*cols+j]);
    }
    printf("\n");
  }
}

void jerasure_print_bitmatrix(int *m, int rows, int cols, int w)
{
  int i, j;

  for (i = 0; i < rows; i++) {
    if (i != 0 && i%w == 0) printf("\n");
    for (j = 0; j < cols; j++) {
      if (j != 0 && j%w == 0) printf(" ");
      printf("%d", m[i*cols+j]);
    }
    printf("\n");
  }
}

int jerasure_make_decoding_matrix(int k, int m, int w, int *matrix, int *erased, int *decoding_matrix, int *dm_ids)
{
  int i, j, *tmpmat;

  j = 0;
  for (i = 0; j < k; i++) {
    if (erased[i] == 0) {
      dm_ids[j] = i;
      j++;
    }
  }

  tmpmat = talloc(int, k*k);
  if (tmpmat == NULL) { return -1; }
  for (i = 0; i < k; i++) {
    if (dm_ids[i] < k) {
      for (j = 0; j < k; j++) tmpmat[i*k+j] = 0;
      tmpmat[i*k+dm_ids[i]] = 1;
    } else {
      for (j = 0; j < k; j++) {
        tmpmat[i*k+j] = matrix[(dm_ids[i]-k)*k+j];
      }
    }
```

```
  }

  i = jerasure_invert_matrix(tmpmat, decoding_matrix, k, w);
  free(tmpmat);
  return i;
}

/* Internal Routine */
int jerasure_make_decoding_bitmatrix(int k, int m, int w, int *matrix, int *erased, int *decoding_matrix, int *dm_ids)
{
  int i, j, *tmpmat;
  int index, mindex;

  j = 0;
  for (i = 0; j < k; i++) {
    if (erased[i] == 0) {
      dm_ids[j] = i;
      j++;
    }
  }

  tmpmat = talloc(int, k*k*w*w);
  if (tmpmat == NULL) { return -1; }
  for (i = 0; i < k; i++) {
    if (dm_ids[i] < k) {
      index = i*k*w*w;
      for (j = 0; j < k*w*w; j++) tmpmat[index+j] = 0;
      index = i*k*w*w+dm_ids[i]*w;
      for (j = 0; j < w; j++) {
        tmpmat[index] = 1;
        index += (k*w+1);
      }
    } else {
      index = i*k*w*w;
      mindex = (dm_ids[i]-k)*k*w*w;
      for (j = 0; j < k*w*w; j++) {
        tmpmat[index+j] = matrix[mindex+j];
      }
    }
  }

  i = jerasure_invert_bitmatrix(tmpmat, decoding_matrix, k*w);
  free(tmpmat);
  return i;
}

int jerasure_matrix_decode(int k, int m, int w, int *matrix, int row_k_ones, int *erasures,
                          char **data_ptrs, char **coding_ptrs, int size)
{
  int i, edd, lastdrive;
  int *tmpids;
  int *erased, *decoding_matrix, *dm_ids;

  if (w != 8 && w != 16 && w != 32) return -1;

  erased = jerasure_erasures_to_erased(k, m, erasures);
  if (erased == NULL) return -1;

  /* Find the number of data drives failed */
```

src/jerasure.c lines 181 to 240

```
  lastdrive = k;

  edd = 0;
  for (i = 0; i < k; i++) {
    if (erased[i]) {
      edd++;
      lastdrive = i;
    }
  }

  /* You only need to create the decoding matrix in the following cases:

      1. edd > 0 and row_k_ones is false.
      2. edd > 0 and row_k_ones is true and coding device 0 has been erased.
      3. edd > 1

      We're going to use lastdrive to denote when to stop decoding data.
      At this point in the code, it is equal to the last erased data device.
      However, if we can't use the parity row to decode it (i.e. row_k_ones=0
         or erased[k] = 1, we're going to set it to k so that the decoding
         pass will decode all data.
   */

  if (!row_k_ones || erased[k]) lastdrive = k;

  dm_ids = NULL;
  decoding_matrix = NULL;

  if (edd > 1 || (edd > 0 && (!row_k_ones || erased[k]))) {
    dm_ids = talloc(int, k);
    if (dm_ids == NULL) {
      free(erased);
      return -1;
    }

    decoding_matrix = talloc(int, k*k);
    if (decoding_matrix == NULL) {
      free(erased);
      free(dm_ids);
      return -1;
    }

    if (jerasure_make_decoding_matrix(k, m, w, matrix, erased, decoding_matrix, dm_ids) < 0) {
      free(erased);
      free(dm_ids);
      free(decoding_matrix);
      return -1;
    }
  }

  /* Decode the data drives.
     If row_k_ones is true and coding device 0 is intact, then only decode edd-1 drives.
     This is done by stopping at lastdrive.
     We test whether edd > 0 so that we can exit the loop early if we're done.
   */

  for (i = 0; edd > 0 && i < lastdrive; i++) {
    if (erased[i]) {
      jerasure_matrix_dotprod(k, w, decoding_matrix+(i*k), dm_ids, i, data_ptrs, coding_ptrs, size);
      edd--;
```

```
    }
  }

  /* Then if necessary, decode drive lastdrive */

  if (edd > 0) {
    tmpids = talloc(int, k);
    if (!tmpids) {
      free(erased);
      free(dm_ids);
      free(decoding_matrix);
      return -1;
    }
    for (i = 0; i < k; i++) {
      tmpids[i] = (i < lastdrive) ? i : i+1;
    }
    jerasure_matrix_dotprod(k, w, matrix, tmpids, lastdrive, data_ptrs, coding_ptrs, size);
    free(tmpids);
  }

  /* Finally, re-encode any erased coding devices */

  for (i = 0; i < m; i++) {
    if (erased[k+i]) {
      jerasure_matrix_dotprod(k, w, matrix+(i*k), NULL, i+k, data_ptrs, coding_ptrs, size);
    }
  }

  free(erased);
  if (dm_ids != NULL) free(dm_ids);
  if (decoding_matrix != NULL) free(decoding_matrix);

  return 0;
}


int *jerasure_matrix_to_bitmatrix(int k, int m, int w, int *matrix)
{
  int *bitmatrix;
  int rowelts, rowindex, colindex, elt, i, j, l, x;

  if (matrix == NULL) { return NULL; }

  bitmatrix = talloc(int, k*m*w*w);
  if (!bitmatrix) return NULL;

  rowelts = k * w;
  rowindex = 0;

  for (i = 0; i < m; i++) {
    colindex = rowindex;
    for (j = 0; j < k; j++) {
      elt = matrix[i*k+j];
      for (x = 0; x < w; x++) {
        for (l = 0; l < w; l++) {
          bitmatrix[colindex+x+l*rowelts] = ((elt & (1 << l)) ? 1 : 0);
        }
        elt = galois_single_multiply(elt, 2, w);
      }
      colindex += w;
```

src/jerasure.c lines 301 to 360

```
    }
    rowindex += rowelts * w;
  }
  return bitmatrix;
}

void jerasure_matrix_encode(int k, int m, int w, int *matrix,
                          char **data_ptrs, char **coding_ptrs, int size)
{
  int i;

  if (w != 8 && w != 16 && w != 32) {
    fprintf(stderr, "ERROR: jerasure_matrix_encode() and w is not 8, 16 or 32\n");
    exit(1);
  }

  for (i = 0; i < m; i++) {
    jerasure_matrix_dotprod(k, w, matrix+(i*k), NULL, k+i, data_ptrs, coding_ptrs, size);
  }
}

void jerasure_bitmatrix_dotprod(int k, int w, int *bitmatrix_row,
                             int *src_ids, int dest_id,
                             char **data_ptrs, char **coding_ptrs, int size, int packetsize)
{
  int j, sindex, pstarted, index, x, y;
  char *dptr, *pptr, *bdptr, *bpptr;

  if (size%(w*packetsize) != 0) {
    fprintf(stderr, "jerasure_bitmatrix_dotprod - size%c(w*packetsize)) must = 0\n", '%');
    exit(1);
  }

  bpptr = (dest_id < k) ? data_ptrs[dest_id] : coding_ptrs[dest_id-k];

  for (sindex = 0; sindex < size; sindex += (packetsize*w)) {
    index = 0;
    for (j = 0; j < w; j++) {
      pstarted = 0;
      pptr = bpptr + sindex + j*packetsize;
      for (x = 0; x < k; x++) {
        if (src_ids == NULL) {
          bdptr = data_ptrs[x];
        } else if (src_ids[x] < k) {
          bdptr = data_ptrs[src_ids[x]];
        } else {
          bdptr = coding_ptrs[src_ids[x]-k];
        }
        for (y = 0; y < w; y++) {
          if (bitmatrix_row[index]) {
            dptr = bdptr + sindex + y*packetsize;
            if (!pstarted) {
              memcpy(pptr, dptr, packetsize);
              jerasure_total_memcpy_bytes += packetsize;
              pstarted = 1;
            } else {
              galois_region_xor(dptr, pptr, packetsize);
              jerasure_total_xor_bytes += packetsize;
            }
          }
```

src/jerasure.c lines 361 to 420

```
          index++;
        }
      }
    }
  }
}

void jerasure_do_parity(int k, char **data_ptrs, char *parity_ptr, int size)
{
  int i;

  memcpy(parity_ptr, data_ptrs[0], size);
  jerasure_total_memcpy_bytes += size;

  for (i = 1; i < k; i++) {
    galois_region_xor(data_ptrs[i], parity_ptr, size);
    jerasure_total_xor_bytes += size;
  }
}

int jerasure_invert_matrix(int *mat, int *inv, int rows, int w)
{
  int cols, i, j, k, x, rs2;
  int row_start, tmp, inverse;

  cols = rows;

  k = 0;
  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      inv[k] = (i == j) ? 1 : 0;
      k++;
    }
  }

  /* First -- convert into upper triangular  */
  for (i = 0; i < cols; i++) {
    row_start = cols*i;

    /* Swap rows if we ave a zero i,i element.  If we can't swap, then the
       matrix was not invertible  */

    if (mat[row_start+i] == 0) {
      for (j = i+1; j < rows && mat[cols*j+i] == 0; j++) ;
      if (j == rows) return -1;
      rs2 = j*cols;
      for (k = 0; k < cols; k++) {
        tmp = mat[row_start+k];
        mat[row_start+k] = mat[rs2+k];
        mat[rs2+k] = tmp;
        tmp = inv[row_start+k];
        inv[row_start+k] = inv[rs2+k];
        inv[rs2+k] = tmp;
      }
    }

    /* Multiply the row by 1/element i,i  */
    tmp = mat[row_start+i];
    if (tmp != 1) {
      inverse = galois_single_divide(1, tmp, w);
```

src/jerasure.c lines 421 to 480

```
      for (j = 0; j < cols; j++) {
        mat[row_start+j] = galois_single_multiply(mat[row_start+j], inverse, w);
        inv[row_start+j] = galois_single_multiply(inv[row_start+j], inverse, w);
      }
    }

    /* Now for each j>i, add A_ji*Ai to Aj  */
    k = row_start+i;
    for (j = i+1; j != cols; j++) {
      k += cols;
      if (mat[k] != 0) {
        if (mat[k] == 1) {
          rs2 = cols*j;
          for (x = 0; x < cols; x++) {
            mat[rs2+x] ^= mat[row_start+x];
            inv[rs2+x] ^= inv[row_start+x];
          }
        } else {
          tmp = mat[k];
          rs2 = cols*j;
          for (x = 0; x < cols; x++) {
            mat[rs2+x] ^= galois_single_multiply(tmp, mat[row_start+x], w);
            inv[rs2+x] ^= galois_single_multiply(tmp, inv[row_start+x], w);
          }
        }
      }
    }
  }

  /* Now the matrix is upper triangular.  Start at the top and multiply down  */

  for (i = rows-1; i >= 0; i--) {
    row_start = i*cols;
    for (j = 0; j < i; j++) {
      rs2 = j*cols;
      if (mat[rs2+i] != 0) {
        tmp = mat[rs2+i];
        mat[rs2+i] = 0;
        for (k = 0; k < cols; k++) {
          inv[rs2+k] ^= galois_single_multiply(tmp, inv[row_start+k], w);
        }
      }
    }
  }
  return 0;
}

int jerasure_invertible_matrix(int *mat, int rows, int w)
{
  int cols, i, j, k, x, rs2;
  int row_start, tmp, inverse;

  cols = rows;

  /* First -- convert into upper triangular  */
  for (i = 0; i < cols; i++) {
    row_start = cols*i;

    /* Swap rows if we ave a zero i,i element.  If we can't swap, then the
       matrix was not invertible  */
```

src/jerasure.c lines 481 to 540

```
    if (mat[row_start+i] == 0) {
      for (j = i+1; j < rows && mat[cols*j+i] == 0; j++) ;
      if (j == rows) return 0;
      rs2 = j*cols;
      for (k = 0; k < cols; k++) {
        tmp = mat[row_start+k];
        mat[row_start+k] = mat[rs2+k];
        mat[rs2+k] = tmp;
      }
    }

    /* Multiply the row by 1/element i,i  */
    tmp = mat[row_start+i];
    if (tmp != 1) {
      inverse = galois_single_divide(1, tmp, w);
      for (j = 0; j < cols; j++) {
        mat[row_start+j] = galois_single_multiply(mat[row_start+j], inverse, w);
      }
    }

    /* Now for each j>i, add A_ji*Ai to Aj  */
    k = row_start+i;
    for (j = i+1; j != cols; j++) {
      k += cols;
      if (mat[k] != 0) {
        if (mat[k] == 1) {
          rs2 = cols*j;
          for (x = 0; x < cols; x++) {
            mat[rs2+x] ^= mat[row_start+x];
          }
        } else {
          tmp = mat[k];
          rs2 = cols*j;
          for (x = 0; x < cols; x++) {
            mat[rs2+x] ^= galois_single_multiply(tmp, mat[row_start+x], w);
          }
        }
      }
    }
  }
  return 1;
}

/* Converts a list-style version of the erasures into an array of k+m elements
   where the element = 1 if the index has been erased, and zero otherwise */

int *jerasure_erasures_to_erased(int k, int m, int *erasures)
{
  int td;
  int t_non_erased;
  int *erased;
  int i;

  td = k+m;
  erased = talloc(int, td);
  if (erased == NULL) return NULL;
  t_non_erased = td;

  for (i = 0; i < td; i++) erased[i] = 0;
```

```
  for (i = 0; erasures[i] != -1; i++) {
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      t_non_erased--;
      if (t_non_erased < k) {
        free(erased);
        return NULL;
      }
    }
  }
  return erased;
}

void jerasure_free_schedule(int **schedule)
{
  int i;

  for (i = 0; schedule[i][0] >= 0; i++) free(schedule[i]);
  free(schedule[i]);
  free(schedule);
}

void jerasure_free_schedule_cache(int k, int m, int ***cache)
{
  int e1, e2;

  if (m != 2) {
    fprintf(stderr, "jerasure_free_schedule_cache(): m must equal 2\n");
    exit(1);
  }

  for (e1 = 0; e1 < k+m; e1++) {
    for (e2 = 0; e2 < e1; e2++) {
      jerasure_free_schedule(cache[e1*(k+m)+e2]);
    }
    jerasure_free_schedule(cache[e1*(k+m)+e1]);
  }
  free(cache);
}

void jerasure_matrix_dotprod(int k, int w, int *matrix_row,
                          int *src_ids, int dest_id,
                          char **data_ptrs, char **coding_ptrs, int size)
{
  int init;
  char *dptr, *sptr;
  int i;

  if (w != 1 && w != 8 && w != 16 && w != 32) {
    fprintf(stderr, "ERROR: jerasure_matrix_dotprod() called and w is not 1, 8, 16 or 32\n");
    exit(1);
  }

  init = 0;

  dptr = (dest_id < k) ? data_ptrs[dest_id] : coding_ptrs[dest_id-k];

  /* First copy or xor any data that does not need to be multiplied by a factor */
```

```
  for (i = 0; i < k; i++) {
    if (matrix_row[i] == 1) {
      if (src_ids == NULL) {
        sptr = data_ptrs[i];
      } else if (src_ids[i] < k) {
        sptr = data_ptrs[src_ids[i]];
      } else {
        sptr = coding_ptrs[src_ids[i]-k];
      }
      if (init == 0) {
        memcpy(dptr, sptr, size);
        jerasure_total_memcpy_bytes += size;
        init = 1;
      } else {
        galois_region_xor(sptr, dptr, size);
        jerasure_total_xor_bytes += size;
      }
    }
  }

  /* Now do the data that needs to be multiplied by a factor */

  for (i = 0; i < k; i++) {
    if (matrix_row[i] != 0 && matrix_row[i] != 1) {
      if (src_ids == NULL) {
        sptr = data_ptrs[i];
      } else if (src_ids[i] < k) {
        sptr = data_ptrs[src_ids[i]];
      } else {
        sptr = coding_ptrs[src_ids[i]-k];
      }
      switch (w) {
        case 8:  galois_w08_region_multiply(sptr, matrix_row[i], size, dptr, init); break;
        case 16: galois_w16_region_multiply(sptr, matrix_row[i], size, dptr, init); break;
        case 32: galois_w32_region_multiply(sptr, matrix_row[i], size, dptr, init); break;
      }
      jerasure_total_gf_bytes += size;
      init = 1;
    }
  }
}


int jerasure_bitmatrix_decode(int k, int m, int w, int *bitmatrix, int row_k_ones, int *erasures,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize)
{
  int i;
  int *erased;
  int *decoding_matrix;
  int *dm_ids;
  int edd, *tmpids, lastdrive;

  erased = jerasure_erasures_to_erased(k, m, erasures);
  if (erased == NULL) return -1;

  /* See jerasure_matrix_decode for the logic of this routine.  This one works just like
     it, but calls the bitmatrix ops instead */

  lastdrive = k;
```

src/jerasure.c lines 661 to 720

```
  edd = 0;
  for (i = 0; i < k; i++) {
    if (erased[i]) {
      edd++;
      lastdrive = i;
    }
  }

  if (row_k_ones != 1 || erased[k]) lastdrive = k;

  dm_ids = NULL;
  decoding_matrix = NULL;

  if (edd > 1 || (edd > 0 && (row_k_ones != 1 || erased[k]))) {

    dm_ids = talloc(int, k);
    if (dm_ids == NULL) {
      free(erased);
      return -1;
    }

    decoding_matrix = talloc(int, k*k*w*w);
    if (decoding_matrix == NULL) {
      free(erased);
      free(dm_ids);
      return -1;
    }

    if (jerasure_make_decoding_bitmatrix(k, m, w, bitmatrix, erased, decoding_matrix, dm_ids) < 0) {
      free(erased);
      free(dm_ids);
      free(decoding_matrix);
      return -1;
    }
  }

  for (i = 0; edd > 0 && i < lastdrive; i++) {
    if (erased[i]) {
      jerasure_bitmatrix_dotprod(k, w, decoding_matrix+i*k*w*w, dm_ids, i, data_ptrs, coding_ptrs, size, packetsize);
      edd--;
    }
  }

  if (edd > 0) {
    tmpids = talloc(int, k);
    if (!tmpids) {
      free(erased);
      free(dm_ids);
      free(decoding_matrix);
      return -1;
    }
    for (i = 0; i < k; i++) {
      tmpids[i] = (i < lastdrive) ? i : i+1;
    }
    jerasure_bitmatrix_dotprod(k, w, bitmatrix, tmpids, lastdrive, data_ptrs, coding_ptrs, size, packetsize);
    free(tmpids);
  }

  for (i = 0; i < m; i++) {
    if (erased[k+i]) {
```

```
      jerasure_bitmatrix_dotprod(k, w, bitmatrix+i*k*w*w, NULL, k+i, data_ptrs, coding_ptrs, size, packetsize);
    }
  }

  free(erased);
  if (dm_ids != NULL) free(dm_ids);
  if (decoding_matrix != NULL) free(decoding_matrix);

  return 0;
}

static char **set_up_ptrs_for_scheduled_decoding(int k, int m, int *erasures, char **data_ptrs, char **coding_ptrs)
{
  int ddf, cdf;
  int *erased;
  char **ptrs;
  int i, j, x;

  ddf = 0;
  cdf = 0;
  for (i = 0; erasures[i] != -1; i++) {
    if (erasures[i] < k) ddf++; else cdf++;
  }

  erased = jerasure_erasures_to_erased(k, m, erasures);
  if (erased == NULL) return NULL;

  /* Set up ptrs.  It will be as follows:

       - If data drive i has not failed, then ptrs[i] = data_ptrs[i].
       - If data drive i has failed, then ptrs[i] = coding_ptrs[j], where j is the
            lowest unused non-failed coding drive.
       - Elements k to k+ddf-1 are data_ptrs[] of the failed data drives.
       - Elements k+ddf to k+ddf+cdf-1 are coding_ptrs[] of the failed data drives.

       The array row_ids contains the ids of ptrs.
       The array ind_to_row_ids contains the row_id of drive i.

       However, we're going to set row_ids and ind_to_row in a different procedure.
   */

  ptrs = talloc(char *, k+m);
  if (!ptrs) {
    free(erased);
    return NULL;
  }

  j = k;
  x = k;
  for (i = 0; i < k; i++) {
    if (erased[i] == 0) {
      ptrs[i] = data_ptrs[i];
    } else {
      while (erased[j]) j++;
      ptrs[i] = coding_ptrs[j-k];
      j++;
      ptrs[x] = data_ptrs[i];
      x++;
    }
  }
```

src/jerasure.c lines 781 to 840

```
  for (i = k; i < k+m; i++) {
    if (erased[i]) {
      ptrs[x] = coding_ptrs[i-k];
      x++;
    }
  }
  free(erased);
  return ptrs;
}

static int set_up_ids_for_scheduled_decoding(int k, int m, int *erasures, int *row_ids, int *ind_to_row)
{
  int ddf, cdf;
  int *erased;
  int i, j, x;

  ddf = 0;
  cdf = 0;
  for (i = 0; erasures[i] != -1; i++) {
    if (erasures[i] < k) ddf++; else cdf++;
  }

  erased = jerasure_erasures_to_erased(k, m, erasures);
  if (erased == NULL) return -1;

  /* See set_up_ptrs_for_scheduled_decoding for how these are set */

  j = k;
  x = k;
  for (i = 0; i < k; i++) {
    if (erased[i] == 0) {
      row_ids[i] = i;
      ind_to_row[i] = i;
    } else {
      while (erased[j]) j++;
      row_ids[i] = j;
      ind_to_row[j] = i;
      j++;
      row_ids[x] = i;
      ind_to_row[i] = x;
      x++;
    }
  }
  for (i = k; i < k+m; i++) {
    if (erased[i]) {
      row_ids[x] = i;
      ind_to_row[i] = x;
      x++;
    }
  }
  free(erased);
  return 0;
}

static int **jerasure_generate_decoding_schedule(int k, int m, int w, int *bitmatrix, int *erasures, int smart)
{
  int i, j, x, drive, y, index, z;
  int *decoding_matrix, *inverse, *real_decoding_matrix;
  int *ptr;
  int *row_ids;
```

src/jerasure.c lines 841 to 900

```
  int *ind_to_row;
  int ddf, cdf;
  int **schedule;
  int *b1, *b2;

 /* First, figure out the number of data drives that have failed, and the
    number of coding drives that have failed: ddf and cdf */

  ddf = 0;
  cdf = 0;
  for (i = 0; erasures[i] != -1; i++) {
    if (erasures[i] < k) ddf++; else cdf++;
  }

  row_ids = talloc(int, k+m);
  if (!row_ids) return NULL;
  ind_to_row = talloc(int, k+m);
  if (!ind_to_row) {
    free(row_ids);
    return NULL;
  }

  if (set_up_ids_for_scheduled_decoding(k, m, erasures, row_ids, ind_to_row) < 0) {
    free(row_ids);
    free(ind_to_row);
    return NULL;
  }

  /* Now, we're going to create one decoding matrix which is going to
     decode everything with one call.  The hope is that the scheduler
     will do a good job.    This matrix has w*e rows, where e is the
     number of erasures (ddf+cdf) */

  real_decoding_matrix = talloc(int, k*w*(cdf+ddf)*w);
  if (!real_decoding_matrix) {
    free(row_ids);
    free(ind_to_row);
    return NULL;
  }

  /* First, if any data drives have failed, then initialize the first
     ddf*w rows of the decoding matrix from the standard decoding
     matrix inversion */

  if (ddf > 0) {

    decoding_matrix = talloc(int, k*k*w*w);
    if (!decoding_matrix) {
      free(row_ids);
      free(ind_to_row);
      return NULL;
    }
    ptr = decoding_matrix;
    for (i = 0; i < k; i++) {
      if (row_ids[i] == i) {
        bzero(ptr, k*w*w*sizeof(int));
        for (x = 0; x < w; x++) {
          ptr[x+i*w+x*k*w] = 1;
        }
      } else {
```

```
        memcpy(ptr, bitmatrix+k*w*w*(row_ids[i]-k), k*w*w*sizeof(int));
      }
      ptr += (k*w*w);
    }
    inverse = talloc(int, k*k*w*w);
    if (!inverse) {
      free(row_ids);
      free(ind_to_row);
      free(decoding_matrix);
      return NULL;
    }
    jerasure_invert_bitmatrix(decoding_matrix, inverse, k*w);

/*    printf("\nMatrix to invert\n");
    jerasure_print_bitmatrix(decoding_matrix, k*w, k*w, w);
    printf("\n");
    printf("\nInverse\n");
    jerasure_print_bitmatrix(inverse, k*w, k*w, w);
    printf("\n"); */

    free(decoding_matrix);
    ptr = real_decoding_matrix;
    for (i = 0; i < ddf; i++) {
      memcpy(ptr, inverse+k*w*w*row_ids[k+i], sizeof(int)*k*w*w);
      ptr += (k*w*w);
    }
    free(inverse);
  }

  /* Next, here comes the hard part.  For each coding node that needs
     to be decoded, you start by putting its rows of the distribution
     matrix into the decoding matrix.  If there were no failed data
     nodes, then you're done.  However, if there have been failed
     data nodes, then you need to modify the columns that correspond
     to the data nodes.  You do that by first zeroing them.  Then
     whereever there is a one in the distribution matrix, you XOR
     in the corresponding row from the failed data node's entry in
     the decoding matrix.  The whole process kind of makes my head
     spin, but it works.
   */

  for (x = 0; x < cdf; x++) {
    drive = row_ids[x+ddf+k]-k;
    ptr = real_decoding_matrix + k*w*w*(ddf+x);
    memcpy(ptr, bitmatrix+drive*k*w*w, sizeof(int)*k*w*w);

    for (i = 0; i < k; i++) {
      if (row_ids[i] != i) {
        for (j = 0; j < w; j++) {
          bzero(ptr+j*k*w+i*w, sizeof(int)*w);
        }
      }
    }

    /* There's the yucky part */

    index = drive*k*w*w;
    for (i = 0; i < k; i++) {
      if (row_ids[i] != i) {
        b1 = real_decoding_matrix+(ind_to_row[i]-k)*k*w*w;
```

```
          for (j = 0; j < w; j++) {
            b2 = ptr + j*k*w;
            for (y = 0; y < w; y++) {
              if (bitmatrix[index+j*k*w+i*w+y]) {
                for (z = 0; z < k*w; z++) {
                  b2[z] = b2[z] ^ b1[z+y*k*w];
                }
              }
            }
          }
        }
      }
  }

/*
  printf("\n\nReal Decoding Matrix\n\n");
  jerasure_print_bitmatrix(real_decoding_matrix, (ddf+cdf)*w, k*w, w);
  printf("\n"); */
  if (smart) {
    schedule = jerasure_smart_bitmatrix_to_schedule(k, ddf+cdf, w, real_decoding_matrix);
  } else {
    schedule = jerasure_dumb_bitmatrix_to_schedule(k, ddf+cdf, w, real_decoding_matrix);
  }
  free(row_ids);
  free(ind_to_row);
  free(real_decoding_matrix);
  return schedule;
}

int jerasure_schedule_decode_lazy(int k, int m, int w, int *bitmatrix, int *erasures,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize,
                            int smart)
{
  int i, tdone;
  char **ptrs;
  int **schedule;

  ptrs = set_up_ptrs_for_scheduled_decoding(k, m, erasures, data_ptrs, coding_ptrs);
  if (ptrs == NULL) return -1;

  schedule = jerasure_generate_decoding_schedule(k, m, w, bitmatrix, erasures, smart);
  if (schedule == NULL) {
    free(ptrs);
    return -1;
  }

  for (tdone = 0; tdone < size; tdone += packetsize*w) {
  jerasure_do_scheduled_operations(ptrs, schedule, packetsize);
    for (i = 0; i < k+m; i++) ptrs[i] += (packetsize*w);
  }

  jerasure_free_schedule(schedule);
  free(ptrs);

  return 0;
}

int jerasure_schedule_decode_cache(int k, int m, int w, int ***scache, int *erasures,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize)
{
```

```
  int i, tdone;
  char **ptrs;
  int **schedule;
  int index;

  if (erasures[1] == -1) {
    index = erasures[0]*(k+m) + erasures[0];
  } else if (erasures[2] == -1) {
    index = erasures[0]*(k+m) + erasures[1];
  } else {
    return -1;
  }

  schedule = scache[index];

  ptrs = set_up_ptrs_for_scheduled_decoding(k, m, erasures, data_ptrs, coding_ptrs);
  if (ptrs == NULL) return -1;


  for (tdone = 0; tdone < size; tdone += packetsize*w) {
  jerasure_do_scheduled_operations(ptrs, schedule, packetsize);
    for (i = 0; i < k+m; i++) ptrs[i] += (packetsize*w);
  }

  free(ptrs);

  return 0;
}

/* This only works when m = 2 */

int ***jerasure_generate_schedule_cache(int k, int m, int w, int *bitmatrix, int smart)
{
  int ***scache;
  int erasures[3];
  int e1, e2;

  /* Ok -- this is yucky, but it's how I'm doing it.  You will make an index out
     of erasures, which will be  e1*(k+m)+(e2).  If there is no e2, then e2 = e1.
     Isn't that clever and confusing.  Sorry.

     We're not going to worry about ordering -- in other words, the schedule for
     e1,e2 will be the same as e2,e1.  They will have the same pointer -- the
     schedule will not be duplicated. */

  if (m != 2) return NULL;

  scache = talloc(int **, (k+m)*(k+m+1));
  if (scache == NULL) return NULL;

  for (e1 = 0; e1 < k+m; e1++) {
    erasures[0] = e1;
    for (e2 = 0; e2 < e1; e2++) {
      erasures[1] = e2;
      erasures[2] = -1;
      scache[e1*(k+m)+e2] = jerasure_generate_decoding_schedule(k, m, w, bitmatrix, erasures, smart);
      scache[e2*(k+m)+e1] = scache[e1*(k+m)+e2];
    }
    erasures[1] = -1;
    scache[e1*(k+m)+e1] = jerasure_generate_decoding_schedule(k, m, w, bitmatrix, erasures, smart);
```

```
  }
  return scache;

}

int jerasure_invert_bitmatrix(int *mat, int *inv, int rows)
{
  int cols, i, j, k;
  int tmp;

  cols = rows;

  k = 0;
  for (i = 0; i < rows; i++) {
    for (j = 0; j < cols; j++) {
      inv[k] = (i == j) ? 1 : 0;
      k++;
    }
  }

  /* First -- convert into upper triangular */

  for (i = 0; i < cols; i++) {

    /* Swap rows if we have a zero i,i element.  If we can't swap, then the
       matrix was not invertible */

    if ((mat[i*cols+i]) == 0) {
      for (j = i+1; j < rows && (mat[j*cols+i]) == 0; j++) ;
      if (j == rows) return -1;
      for (k = 0; k < cols; k++) {
        tmp = mat[i*cols+k]; mat[i*cols+k] = mat[j*cols+k]; mat[j*cols+k] = tmp;
        tmp = inv[i*cols+k]; inv[i*cols+k] = inv[j*cols+k]; inv[j*cols+k] = tmp;
      }
    }

    /* Now for each j>i, add A_ji*Ai to Aj */
    for (j = i+1; j != rows; j++) {
      if (mat[j*cols+i] != 0) {
        for (k = 0; k < cols; k++) {
          mat[j*cols+k] ^= mat[i*cols+k];
          inv[j*cols+k] ^= inv[i*cols+k];
        }
      }
    }
  }

  /* Now the matrix is upper triangular.  Start at the top and multiply down */

  for (i = rows-1; i >= 0; i--) {
    for (j = 0; j < i; j++) {
      if (mat[j*cols+i]) {
        for (k = 0; k < cols; k++) {
          mat[j*cols+k] ^= mat[i*cols+k];
          inv[j*cols+k] ^= inv[i*cols+k];
        }
      }
    }
  }
  return 0;
```

```
}

int jerasure_invertible_bitmatrix(int *mat, int rows)
{
  int cols, i, j, k;
  int tmp;

  cols = rows;

  /* First -- convert into upper triangular */

  for (i = 0; i < cols; i++) {

    /* Swap rows if we have a zero i,i element.  If we can't swap, then the
       matrix was not invertible */

    if ((mat[i*cols+i]) == 0) {
      for (j = i+1; j < rows && (mat[j*cols+i]) == 0; j++) ;
      if (j == rows) return 0;
      for (k = 0; k < cols; k++) {
        tmp = mat[i*cols+k]; mat[i*cols+k] = mat[j*cols+k]; mat[j*cols+k] = tmp;
      }
    }

    /* Now for each j>i, add A_ji*Ai to Aj */
    for (j = i+1; j != rows; j++) {
      if (mat[j*cols+i] != 0) {
        for (k = 0; k < cols; k++) {
          mat[j*cols+k] ^= mat[i*cols+k];
        }
      }
    }
  }
  return 1;
}


int *jerasure_matrix_multiply(int *m1, int *m2, int r1, int c1, int r2, int c2, int w)
{
  int *product, i, j, k;

  product = (int *) malloc(sizeof(int)*r1*c2);
  for (i = 0; i < r1*c2; i++) product[i] = 0;

  for (i = 0; i < r1; i++) {
    for (j = 0; j < c2; j++) {
      for (k = 0; k < r2; k++) {
        product[i*c2+j] ^= galois_single_multiply(m1[i*c1+k], m2[k*c2+j], w);
      }
    }
  }
  return product;
}

void jerasure_get_stats(double *fill_in)
{
  fill_in[0] = jerasure_total_xor_bytes;
  fill_in[1] = jerasure_total_gf_bytes;
  fill_in[2] = jerasure_total_memcpy_bytes;
  jerasure_total_xor_bytes = 0;
```

src/jerasure.c lines 1201 to 1260

```
  jerasure_total_gf_bytes = 0;
  jerasure_total_memcpy_bytes = 0;
}

void jerasure_do_scheduled_operations(char **ptrs, int **operations, int packetsize)
{
  char *sptr;
  char *dptr;
  int op;

  for (op = 0; operations[op][0] >= 0; op++) {
    sptr = ptrs[operations[op][0]] + operations[op][1]*packetsize;
    dptr = ptrs[operations[op][2]] + operations[op][3]*packetsize;
    if (operations[op][4]) {
/*      printf("%d,%d %d,%d\n", operations[op][0],
      operations[op][1],
      operations[op][2],
      operations[op][3]);
      printf("xor(0x%x, 0x%x -> 0x%x, %d)\n", sptr, dptr, dptr, packetsize); */
      galois_region_xor(sptr, dptr, packetsize);
      jerasure_total_xor_bytes += packetsize;
    } else {
/*      printf("memcpy(0x%x <- 0x%x)\n", dptr, sptr); */
      memcpy(dptr, sptr, packetsize);
      jerasure_total_memcpy_bytes += packetsize;
    }
  }
}

void jerasure_schedule_encode(int k, int m, int w, int **schedule,
                                   char **data_ptrs, char **coding_ptrs, int size, int packetsize)
{
  char **ptr_copy;
  int i, tdone;

  ptr_copy = talloc(char *, (k+m));
  for (i = 0; i < k; i++) ptr_copy[i] = data_ptrs[i];
  for (i = 0; i < m; i++) ptr_copy[i+k] = coding_ptrs[i];
  for (tdone = 0; tdone < size; tdone += packetsize*w) {
    jerasure_do_scheduled_operations(ptr_copy, schedule, packetsize);
    for (i = 0; i < k+m; i++) ptr_copy[i] += (packetsize*w);
  }
  free(ptr_copy);
}

int **jerasure_dumb_bitmatrix_to_schedule(int k, int m, int w, int *bitmatrix)
{
  int **operations;
  int op;
  int index, optodo, i, j;

  operations = talloc(int *, k*m*w*w+1);
  if (!operations) return NULL;
  op = 0;

  index = 0;
  for (i = 0; i < m*w; i++) {
    optodo = 0;
    for (j = 0; j < k*w; j++) {
      if (bitmatrix[index]) {
```

```
        operations[op] = talloc(int, 5);
        if (!operations[op]) { // -ENOMEM
          goto error;
        }
        operations[op][4] = optodo;
        operations[op][0] = j/w;
        operations[op][1] = j%w;
        operations[op][2] = k+i/w;
        operations[op][3] = i%w;
        optodo = 1;
        op++;

      }
      index++;
    }
  }
  operations[op] = talloc(int, 5);
  if (!operations[op]) {
    // -ENOMEM
    goto error;
  }
  operations[op][0] = -1;
  return operations;

error:
  for (i = 0; i <= op; i++) {
    free(operations[op]);
  }
  free(operations);
  return NULL;
}

int **jerasure_smart_bitmatrix_to_schedule(int k, int m, int w, int *bitmatrix)
{
  int **operations;
  int op;
  int i, j;
  int *diff, *from, *b1, *flink, *blink;
  int *ptr, no, row;
  int optodo;
  int bestrow = 0, bestdiff, top;

/*   printf("Scheduling:\n\n");
  jerasure_print_bitmatrix(bitmatrix, m*w, k*w, w); */

  operations = talloc(int *, k*m*w*w+1);
  if (!operations) return NULL;
  op = 0;

  diff = talloc(int, m*w);
  if (!diff) {
    free(operations);
    return NULL;
  }
  from = talloc(int, m*w);
  if (!from) {
    free(operations);
    free(diff);
    return NULL;
  }
```

```
  flink = talloc(int, m*w);
  if (!flink) {
    free(operations);
    free(diff);
    free(from);
    return NULL;
  }
  blink = talloc(int, m*w);
  if (!blink) {
    free(operations);
    free(diff);
    free(from);
    free(flink);
    return NULL;
  }

  ptr = bitmatrix;

  bestdiff = k*w+1;
  top = 0;
  for (i = 0; i < m*w; i++) {
    no = 0;
    for (j = 0; j < k*w; j++) {
      no += *ptr;
      ptr++;
    }
    diff[i] = no;
    from[i] = -1;
    flink[i] = i+1;
    blink[i] = i-1;
    if (no < bestdiff) {
      bestdiff = no;
      bestrow = i;
    }
  }

  flink[m*w-1] = -1;

  while (top != -1) {
    row = bestrow;
    /* printf("Doing row %d - %d from %d\n", row, diff[row], from[row]);  */

    if (blink[row] == -1) {
      top = flink[row];
      if (top != -1) blink[top] = -1;
    } else {
      flink[blink[row]] = flink[row];
      if (flink[row] != -1) {
        blink[flink[row]] = blink[row];
      }
    }

    ptr = bitmatrix + row*k*w;
    if (from[row] == -1) {
      optodo = 0;
      for (j = 0; j < k*w; j++) {
        if (ptr[j]) {
          operations[op] = talloc(int, 5);
          if (!operations[op]) goto error;
          operations[op][4] = optodo;
```

```
          operations[op][0] = j/w;
          operations[op][1] = j%w;
          operations[op][2] = k+row/w;
          operations[op][3] = row%w;
          optodo = 1;
          op++;
        }
      }
    } else {
      operations[op] = talloc(int, 5);
      if (!operations[op]) goto error;
      operations[op][4] = 0;
      operations[op][0] = k+from[row]/w;
      operations[op][1] = from[row]%w;
      operations[op][2] = k+row/w;
      operations[op][3] = row%w;
      op++;
      b1 = bitmatrix + from[row]*k*w;
      for (j = 0; j < k*w; j++) {
        if (ptr[j] ^ b1[j]) {
          operations[op] = talloc(int, 5);
          if (!operations[op]) goto error;
          operations[op][4] = 1;
          operations[op][0] = j/w;
          operations[op][1] = j%w;
          operations[op][2] = k+row/w;
          operations[op][3] = row%w;
          optodo = 1;
          op++;
        }
      }
    }
    bestdiff = k*w+1;
    for (i = top; i != -1; i = flink[i]) {
      no = 1;
      b1 = bitmatrix + i*k*w;
      for (j = 0; j < k*w; j++) no += (ptr[j] ^ b1[j]);
      if (no < diff[i]) {
        from[i] = row;
        diff[i] = no;
      }
      if (diff[i] < bestdiff) {
        bestdiff = diff[i];
        bestrow = i;
      }
    }
  }

  operations[op] = talloc(int, 5);
  if (!operations[op]) goto error;
  operations[op][0] = -1;
  free(from);
  free(diff);
  free(blink);
  free(flink);

  return operations;

error:
  for (i = 0; i <= op; i++) {
```

src/jerasure.c lines 1441 to 1483

```
    free(operations[op]);
  }
  free(operations);
  free(from);
  free(diff);
  free(blink);
  free(flink);
  return NULL;
}

void jerasure_bitmatrix_encode(int k, int m, int w, int *bitmatrix,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize)
{
  int i;

  if (packetsize%sizeof(long) != 0) {
    fprintf(stderr, "jerasure_bitmatrix_encode - packetsize(%d) %c sizeof(long) != 0\n", packetsize, '%');
    exit(1);
  }
  if (size%(packetsize*w) != 0) {
    fprintf(stderr, "jerasure_bitmatrix_encode - size(%d) %c (packetsize(%d)*w(%d))) != 0\n",
         size, '%', packetsize, w);
    exit(1);
  }

  for (i = 0; i < m; i++) {
    jerasure_bitmatrix_dotprod(k, w, bitmatrix+i*k*w*w, NULL, k+i, data_ptrs, coding_ptrs, size, packetsize);
  }
}

/*
 * Exported function for use by autoconf to perform quick
 * spot-check.
 */
int jerasure_autoconf_test()
{
  int x = galois_single_multiply(1, 2, 8);
  if (x != 2) {
    return -1;
  }
  return 0;
}
```

src/liberation.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "galois.h"
#include "jerasure.h"
#include "liberation.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

int *liberation_coding_bitmatrix(int k, int w)
{
  int *matrix, i, j, index;
```

```
  if (k > w) return NULL;
  matrix = talloc(int, 2*k*w*w);
  if (matrix == NULL) return NULL;
  bzero(matrix, sizeof(int)*2*k*w*w);

  /* Set up identity matrices */

  for(i = 0; i < w; i++) {
    index = i*k*w+i;
    for (j = 0; j < k; j++) {
      matrix[index] = 1;
      index += w;
    }
  }

  /* Set up liberation matrices */

  for (j = 0; j < k; j++) {
    index = k*w*w+j*w;
    for (i = 0; i < w; i++) {
      matrix[index+(j+i)%w] = 1;
      index += (k*w);
    }
    if (j > 0) {
      i = (j*((w-1)/2))%w;
      matrix[k*w*w+j*w+i*k*w+(i+j-1)%w] = 1;
    }
  }
  return matrix;
}


int *liber8tion_coding_bitmatrix(int k)
{
  int *matrix, i, j, index;
  int w;

  w = 8;
  if (k > w) return NULL;
  matrix = talloc(int, 2*k*w*w);
  if (matrix == NULL) return NULL;
  bzero(matrix, sizeof(int)*2*k*w*w);

  /* Set up identity matrices */

  for(i = 0; i < w; i++) {
    index = i*k*w+i;
    for (j = 0; j < k; j++) {
      matrix[index] = 1;
      index += w;
    }
  }

  /* Set up liber8tion matrices */

  index = k*w*w;

  if (k == 0) return matrix;
  matrix[index+0*k*w+0*w+0] = 1;
  matrix[index+1*k*w+0*w+1] = 1;
```

src/liberation.c lines 121 to 180

```
  matrix[index+2*k*w+0*w+2] = 1;
  matrix[index+3*k*w+0*w+3] = 1;
  matrix[index+4*k*w+0*w+4] = 1;
  matrix[index+5*k*w+0*w+5] = 1;
  matrix[index+6*k*w+0*w+6] = 1;
  matrix[index+7*k*w+0*w+7] = 1;

  if (k == 1) return matrix;
  matrix[index+0*k*w+1*w+7] = 1;
  matrix[index+1*k*w+1*w+3] = 1;
  matrix[index+2*k*w+1*w+0] = 1;
  matrix[index+3*k*w+1*w+2] = 1;
  matrix[index+4*k*w+1*w+6] = 1;
  matrix[index+5*k*w+1*w+1] = 1;
  matrix[index+6*k*w+1*w+5] = 1;
  matrix[index+7*k*w+1*w+4] = 1;
  matrix[index+4*k*w+1*w+7] = 1;

  if (k == 2) return matrix;
  matrix[index+0*k*w+2*w+6] = 1;
  matrix[index+1*k*w+2*w+2] = 1;
  matrix[index+2*k*w+2*w+4] = 1;
  matrix[index+3*k*w+2*w+0] = 1;
  matrix[index+4*k*w+2*w+7] = 1;
  matrix[index+5*k*w+2*w+3] = 1;
  matrix[index+6*k*w+2*w+1] = 1;
  matrix[index+7*k*w+2*w+5] = 1;
  matrix[index+1*k*w+2*w+3] = 1;

  if (k == 3) return matrix;
  matrix[index+0*k*w+3*w+2] = 1;
  matrix[index+1*k*w+3*w+5] = 1;
  matrix[index+2*k*w+3*w+7] = 1;
  matrix[index+3*k*w+3*w+6] = 1;
  matrix[index+4*k*w+3*w+0] = 1;
  matrix[index+5*k*w+3*w+3] = 1;
  matrix[index+6*k*w+3*w+4] = 1;
  matrix[index+7*k*w+3*w+1] = 1;
  matrix[index+5*k*w+3*w+4] = 1;

  if (k == 4) return matrix;
  matrix[index+0*k*w+4*w+5] = 1;
  matrix[index+1*k*w+4*w+6] = 1;
  matrix[index+2*k*w+4*w+1] = 1;
  matrix[index+3*k*w+4*w+7] = 1;
  matrix[index+4*k*w+4*w+2] = 1;
  matrix[index+5*k*w+4*w+4] = 1;
  matrix[index+6*k*w+4*w+3] = 1;
  matrix[index+7*k*w+4*w+0] = 1;
  matrix[index+2*k*w+4*w+0] = 1;

  if (k == 5) return matrix;
  matrix[index+0*k*w+5*w+1] = 1;
  matrix[index+1*k*w+5*w+2] = 1;
  matrix[index+2*k*w+5*w+3] = 1;
  matrix[index+3*k*w+5*w+4] = 1;
  matrix[index+4*k*w+5*w+5] = 1;
  matrix[index+5*k*w+5*w+6] = 1;
  matrix[index+6*k*w+5*w+7] = 1;
  matrix[index+7*k*w+5*w+0] = 1;
```

src/liberation.c lines 181 to 240

```
  matrix[index+7*k*w+5*w+2] = 1;

  if (k == 6) return matrix;
  matrix[index+0*k*w+6*w+3] = 1;
  matrix[index+1*k*w+6*w+0] = 1;
  matrix[index+2*k*w+6*w+6] = 1;
  matrix[index+3*k*w+6*w+5] = 1;
  matrix[index+4*k*w+6*w+1] = 1;
  matrix[index+5*k*w+6*w+7] = 1;
  matrix[index+6*k*w+6*w+4] = 1;
  matrix[index+7*k*w+6*w+2] = 1;
  matrix[index+6*k*w+6*w+5] = 1;

  if (k == 7) return matrix;
  matrix[index+0*k*w+7*w+4] = 1;
  matrix[index+1*k*w+7*w+7] = 1;
  matrix[index+2*k*w+7*w+1] = 1;
  matrix[index+3*k*w+7*w+5] = 1;
  matrix[index+4*k*w+7*w+3] = 1;
  matrix[index+5*k*w+7*w+2] = 1;
  matrix[index+6*k*w+7*w+0] = 1;
  matrix[index+7*k*w+7*w+6] = 1;
  matrix[index+3*k*w+7*w+1] = 1;

  return matrix;
}

int *blaum_roth_coding_bitmatrix(int k, int w)
{
  int *matrix, i, j, index, l, m, p;

  if (k > w) return NULL ;

  matrix = talloc(int, 2*k*w*w);
  if (matrix == NULL) return NULL;
  bzero(matrix, sizeof(int)*2*k*w*w);

  /* Set up identity matrices */

  for(i = 0; i < w; i++) {
    index = i*k*w+i;
    for (j = 0; j < k; j++) {
      matrix[index] = 1;
      index += w;
    }
  }

  /* Set up blaum_roth matrices -- Ignore identity */

  p = w+1;
  for (j = 0; j < k; j++) {
    index = k*w*w+j*w;
    if (j == 0) {
      for (l = 0; l < w; l++) {
        matrix[index+l] = 1;
        index += k*w;
      }
    } else {
      i = j;
      for (l = 1; l <= w; l++) {
```

src/liberation.c lines 241 to 262

```
        if (l != p-i) {
          m = l+i;
          if (m >= p) m -= p;
          m--;
          matrix[index+m] = 1;
        } else {
          matrix[index+i-1] = 1;
          if (i%2 == 0) {
            m = i/2;
          } else {
            m = (p/2) + 1 + (i/2);
          }
          m--;
          matrix[index+m] = 1;
        }
        index += k*w;
      }
    }
  }

  return matrix;
}
```

src/reed_sol.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <gf_complete.h>
#include "galois.h"
#include "jerasure.h"
#include "reed_sol.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

int *reed_sol_r6_coding_matrix(int k, int w)
{
  int *matrix;
```

```
  int i, tmp;

  if (w != 8 && w != 16 && w != 32) return NULL;

  matrix = talloc(int, 2*k);
  if (matrix == NULL) return NULL;

  for (i = 0; i < k; i++) matrix[i] = 1;
  matrix[k] = 1;
  tmp = 1;
  for (i = 1; i < k; i++) {
    tmp = galois_single_multiply(tmp, 2, w);
    matrix[k+i] = tmp;
  }
  return matrix;
}

int *reed_sol_vandermonde_coding_matrix(int k, int m, int w)
{
  int i, j;
  int *vdm, *dist;

  vdm = reed_sol_big_vandermonde_distribution_matrix(k+m, k, w);
  if (vdm == NULL) return NULL;
  dist = talloc(int, m*k);
  if (dist == NULL) {
    free(vdm);
    return NULL;
  }

  i = k*k;
  for (j = 0; j < m*k; j++) {
    dist[j] = vdm[i];
    i++;
  }
  free(vdm);
  return dist;
}

static int prim08 = -1;
static gf_t GF08;

void reed_sol_galois_w08_region_multby_2(char *region, int nbytes)
{
  if (prim08 == -1) {
    prim08 = galois_single_multiply((1 << 7), 2, 8);
    if (!gf_init_hard(&GF08, 8, GF_MULT_BYTWO_b, GF_REGION_DEFAULT, GF_DIVIDE_DEFAULT,
                      prim08, 0, 0, NULL, NULL)) {
      fprintf(stderr, "Error: Can't initialize the GF for reed_sol_galois_w08_region_multby_2\n");
      exit(1);
    }
  }
  GF08.multiply_region.w32(&GF08, region, region, 2, nbytes, 0);
}

static int prim16 = -1;
static gf_t GF16;

void reed_sol_galois_w16_region_multby_2(char *region, int nbytes)
{
```

src/reed_sol.c lines 121 to 180

```
  if (prim16 == -1) {
    prim16 = galois_single_multiply((1 << 15), 2, 16);
    if (!gf_init_hard(&GF16, 16, GF_MULT_BYTWO_b, GF_REGION_DEFAULT, GF_DIVIDE_DEFAULT,
                      prim16, 0, 0, NULL, NULL)) {
      fprintf(stderr, "Error: Can't initialize the GF for reed_sol_galois_w16_region_multby_2\n");
      exit(1);
    }
  }
  GF16.multiply_region.w32(&GF16, region, region, 2, nbytes, 0);
}

static int prim32 = -1;
static gf_t GF32;

void reed_sol_galois_w32_region_multby_2(char *region, int nbytes)
{
  if (prim32 == -1) {
    prim32 = galois_single_multiply((1 << 31), 2, 32);
    if (!gf_init_hard(&GF32, 32, GF_MULT_BYTWO_b, GF_REGION_DEFAULT, GF_DIVIDE_DEFAULT,
                      prim32, 0, 0, NULL, NULL)) {
      fprintf(stderr, "Error: Can't initialize the GF for reed_sol_galois_w32_region_multby_2\n");
      exit(1);
    }
  }
  GF32.multiply_region.w32(&GF32, region, region, 2, nbytes, 0);
}

int reed_sol_r6_encode(int k, int w, char **data_ptrs, char **coding_ptrs, int size)
{
  int i;

  /* First, put the XOR into coding region 0 */

  memcpy(coding_ptrs[0], data_ptrs[0], size);

  for (i = 1; i < k; i++) galois_region_xor(data_ptrs[i], coding_ptrs[0], size);

  /* Next, put the sum of (2^j)*Dj into coding region 1 */

  memcpy(coding_ptrs[1], data_ptrs[k-1], size);

  for (i = k-2; i >= 0; i--) {
    switch (w) {
      case 8:  reed_sol_galois_w08_region_multby_2(coding_ptrs[1], size); break;
      case 16: reed_sol_galois_w16_region_multby_2(coding_ptrs[1], size); break;
      case 32: reed_sol_galois_w32_region_multby_2(coding_ptrs[1], size); break;
      default: return 0;
    }

    galois_region_xor(data_ptrs[i], coding_ptrs[1], size);
  }
  return 1;
}

int *reed_sol_extended_vandermonde_matrix(int rows, int cols, int w)
{
  int *vdm;
  int i, j, k;

  if (w < 30 && (1 << w) < rows) return NULL;
```

```
  if (w < 30 && (1 << w) < cols) return NULL;

  vdm = talloc(int, rows*cols);
  if (vdm == NULL) { return NULL; }

  vdm[0] = 1;
  for (j = 1; j < cols; j++) vdm[j] = 0;
  if (rows == 1) return vdm;

  i=(rows-1)*cols;
  for (j = 0; j < cols-1; j++) vdm[i+j] = 0;
  vdm[i+j] = 1;
  if (rows == 2) return vdm;

  for (i = 1; i < rows-1; i++) {
    k = 1;
    for (j = 0; j < cols; j++) {
      vdm[i*cols+j] = k;
      k = galois_single_multiply(k, i, w);
    }
  }
  return vdm;
}

int *reed_sol_big_vandermonde_distribution_matrix(int rows, int cols, int w)
{
  int *dist;
  int i, j, k;
  int sindex, srindex, siindex, tmp;

  if (cols >= rows) return NULL;

  dist = reed_sol_extended_vandermonde_matrix(rows, cols, w);
  if (dist == NULL) return NULL;

  sindex = 0;
  for (i = 1; i < cols; i++) {
    sindex += cols;

    /* Find an appropriate row -- where i,i != 0 */
    srindex = sindex+i;
    for (j = i; j < rows && dist[srindex] == 0; j++) srindex += cols;
    if (j >= rows) {   /* This should never happen if rows/w are correct */
      fprintf(stderr, "reed_sol_big_vandermonde_distribution_matrix(%d,%d,%d) - couldn't make matrix\n",
              rows, cols, w);
      exit(1);
    }

    /* If necessary, swap rows */
    if (j != i) {
      srindex -= i;
      for (k = 0; k < cols; k++) {
        tmp = dist[srindex+k];
        dist[srindex+k] = dist[sindex+k];
        dist[sindex+k] = tmp;
      }
    }

    /* If Element i,i is not equal to 1, multiply the column by 1/i */
```

```
    if (dist[sindex+i] != 1) {
      tmp = galois_single_divide(1, dist[sindex+i], w);
      srindex = i;
      for (j = 0; j < rows; j++) {
        dist[srindex] = galois_single_multiply(tmp, dist[srindex], w);
        srindex += cols;
      }
    }

    /* Now, for each element in row i that is not in column 1, you need
       to make it zero.  Suppose that this is column j, and the element
       at i,j = e.  Then you want to replace all of column j with
       (col-j + col-i*e).   Note, that in row i, col-i = 1 and col-j = e.
       So (e + 1e) = 0, which is indeed what we want. */

    for (j = 0; j < cols; j++) {
      tmp = dist[sindex+j];
      if (j != i && tmp != 0) {
        srindex = j;
        siindex = i;
        for (k = 0; k < rows; k++) {
          dist[srindex] = dist[srindex] ^ galois_single_multiply(tmp, dist[siindex], w);
          srindex += cols;
          siindex += cols;
        }
      }
    }
  }
  /* We desire to have row k be all ones.  To do that, multiply
     the entire column j by 1/dist[k,j].  Then row j by 1/dist[j,j]. */

  sindex = cols*cols;
  for (j = 0; j < cols; j++) {
    tmp = dist[sindex];
    if (tmp != 1) {
      tmp = galois_single_divide(1, tmp, w);
      srindex = sindex;
      for (i = cols; i < rows; i++) {
        dist[srindex] = galois_single_multiply(tmp, dist[srindex], w);
        srindex += cols;
      }
    }
    sindex++;
  }

  /* Finally, we'd like the first column of each row to be all ones.  To
     do that, we multiply the row by the inverse of the first element. */

  sindex = cols*(cols+1);
  for (i = cols+1; i < rows; i++) {
    tmp = dist[sindex];
    if (tmp != 1) {
      tmp = galois_single_divide(1, tmp, w);
      for (j = 0; j < cols; j++) dist[sindex+j] = galois_single_multiply(dist[sindex+j], tmp, w);
    }
    sindex += cols;
  }

  return dist;
}
```

src/timing.c lines 1 to 60

```c
// Timing measurement utilities implementation.

#include "timing.h"
#include <stddef.h>

void
timing_set(
  struct timing * t)
{
#ifdef USE_CLOCK
  t->clock = clock();
#else
  gettimeofday(&t->tv, NULL);
#endif
}

double
timing_get(
  struct timing * t)
{
#ifdef USE_CLOCK
  // The clock_t type is an "arithmetic type", which could be
  // integral, double, long double, or others.
  //
  // Add 0.0 to make it a double or long double, then divide (in
  // double or long double), then convert to double for our purposes.
  return (double) ((t->clock + 0.0) / CLOCKS_PER_SEC);
#else
  return (double) t->tv.tv_sec + ((double) t->tv.tv_usec) / 1000000.0;
#endif
}

double
timing_now()
{
#ifdef USE_CLOCK
  return (double) ((clock() + 0.0) / CLOCKS_PER_SEC);
#else
  struct timeval tv;
  gettimeofday(&tv, NULL);
  return (double) tv.tv_sec + ((double) tv.tv_usec) / 1000000.0;
#endif
}

double
timing_delta(
  struct timing * t1,
  struct timing * t2)
{
#ifdef USE_CLOCK
  // The clock_t type is an "arithmetic type", which could be
  // integral, double, long double, or others.
  //
  // Subtract first, resulting in another clock_t, then add 0.0 to
  // make it a double or long double, then divide (in double or long
  // double), then convert to double for our purposes.
  return (double) (((t2->clock - t1->clock) + 0.0) / CLOCKS_PER_SEC);
#else
  double const d2 = (double) t2->tv.tv_sec + ((double) t2->tv.tv_usec) / 1000000.0;
  double const d1 = (double) t1->tv.tv_sec + ((double) t1->tv.tv_usec) / 1000000.0;
```

src/timing.c lines 61 to 63

```
  return d2 - d1;
#endif
}
```

include/cauchy.h lines 1 to 54

```
/* *
 * Copyright (c) 2013, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#pragma once

#ifdef __cplusplus
extern "C" {
#endif

extern int *cauchy_original_coding_matrix(int k, int m, int w);
extern int *cauchy_xy_coding_matrix(int k, int m, int w, int *x, int *y);
extern void cauchy_improve_coding_matrix(int k, int m, int w, int *matrix);
extern int *cauchy_good_general_coding_matrix(int k, int m, int w);
extern int cauchy_n_ones(int n, int w);

#ifdef __cplusplus
}
#endif
```

include/galois.h lines 1 to 60

```
/* *
 * Copyright (c) 2013, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#pragma once

#include <stdint.h>
#include <gf_complete.h>

#ifdef __cplusplus
extern "C" {
#endif

extern int galois_init_default_field(int w);
extern int galois_uninit_field(int w);
extern void galois_change_technique(gf_t *gf, int w);

extern int galois_single_multiply(int a, int b, int w);
extern int galois_single_divide(int a, int b, int w);
extern int galois_inverse(int x, int w);

void galois_region_xor(           char *src,         /* Source Region */
                                  char *dest,        /* Dest Region (holds result) */
                                  int nbytes);      /* Number of bytes in region */
```

include/galois.h lines 61 to 103

```
/* These multiply regions in w=8, w=16 and w=32.  They are much faster
   than calling galois_single_multiply.  The regions must be long word aligned. */

void galois_w08_region_multiply(char *region,       /* Region to multiply */
                                  int multby,        /* Number to multiply by */
                                  int nbytes,        /* Number of bytes in region */
                                  char *r2,          /* If r2 != NULL, products go here.
                                                        Otherwise region is overwritten */
                                  int add);          /* If (r2 != NULL && add) the produce is XOR'd with r2 */

void galois_w16_region_multiply(char *region,       /* Region to multiply */
                                  int multby,        /* Number to multiply by */
                                  int nbytes,        /* Number of bytes in region */
                                  char *r2,          /* If r2 != NULL, products go here.
                                                        Otherwise region is overwritten */
                                  int add);          /* If (r2 != NULL && add) the produce is XOR'd with r2 */

void galois_w32_region_multiply(char *region,       /* Region to multiply */
                                  int multby,        /* Number to multiply by */
                                  int nbytes,        /* Number of bytes in region */
                                  char *r2,          /* If r2 != NULL, products go here.
                                                        Otherwise region is overwritten */
                                  int add);          /* If (r2 != NULL && add) the produce is XOR'd with r2 */

gf_t* galois_init_field(int w,
                             int mult_type,
                             int region_type,
                             int divide_type,
                             uint64_t prim_poly,
                             int arg1,
                             int arg2);

gf_t* galois_init_composite_field(int w,
                                int region_type,
                                int divide_type,
                                int degree,
                                gf_t* base_gf);

gf_t * galois_get_field_ptr(int w);

#ifdef __cplusplus
}
#endif
```

include/jerasure.h lines 1 to 60

```
/* *
 * Copyright (c) 2013, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#pragma once

#ifndef _JERASURE_H
#define _JERASURE_H

/* This uses procedures from the Galois Field arithmetic library */

#include "galois.h"

#ifdef __cplusplus
extern "C" {
#endif

/* ------------------------------------------------------------ */
/* In all of the routines below:

   k = Number of data devices
   m = Number of coding devices
   w = Word size

   data_ptrs = An array of k pointers to data which is size bytes.
```

```
               Size must be a multiple of sizeof(long).
               Pointers must also be longword aligned.

   coding_ptrs = An array of m pointers to coding data which is size bytes.

   packetsize = The size of a coding block with bitmatrix coding.
                When you code with a bitmatrix, you will use w packets
                of size packetsize.

   matrix = an array of k*m integers.
            It represents an m by k matrix.
            Element i,j is in matrix[i*k+j];

   bitmatrix = an array of k*m*w*w integers.
            It represents an mw by kw matrix.
            Element i,j is in matrix[i*k*w+j];

   erasures = an array of id's of erased devices.
              Id's are integers between 0 and k+m-1.
              Id's 0 to k-1 are id's of data devices.
              Id's k to k+m-1 are id's of coding devices:
                  Coding device id = id-k.
              If there are e erasures, erasures[e] = -1.

   schedule = an array of schedule operations.

              If there are m operations, then schedule[m][0] = -1.

   operation = an array of 5 integers:

          0 = operation: 0 for copy, 1 for xor (-1 for end)
          1 = source device (0 - k+m-1)
          2 = source packet (0 - w-1)
          3 = destination device (0 - k+m-1)
          4 = destination packet (0 - w-1)
 */

/* ---------------------------------------------------------------  */
/* Bitmatrices / schedules ---------------------------------------- */
/*
 - jerasure_matrix_to_bitmatrix turns a m X k matrix in GF(2^w) into a
                              wm X wk bitmatrix (in GF(2)).  This is
                              explained in the Cauchy Reed-Solomon coding
                              paper.

 - jerasure_dumb_bitmatrix_to_schedule turns a bitmatrix into a schedule
                              using the straightforward algorithm -- just
                              schedule the dot products defined by each
                              row of the matrix.

 - jerasure_smart_bitmatrix_to_schedule turns a bitmatrix into a schedule,
                              but tries to use previous dot products to
                              calculate new ones.  This is the optimization
                              explained in the original Liberation code paper.

 - jerasure_generate_schedule_cache precalcalculate all the schedule for the
                              given distribution bitmatrix.  M must equal 2.

 - jerasure_free_schedule frees a schedule that was allocated with
                              jerasure_XXX_bitmatrix_to_schedule.
```

```
 - jerasure_free_schedule_cache frees a schedule cache that was created with
                              jerasure_generate_schedule_cache.
 */

int *jerasure_matrix_to_bitmatrix(int k, int m, int w, int *matrix);
int **jerasure_dumb_bitmatrix_to_schedule(int k, int m, int w, int *bitmatrix);
int **jerasure_smart_bitmatrix_to_schedule(int k, int m, int w, int *bitmatrix);
int ***jerasure_generate_schedule_cache(int k, int m, int w, int *bitmatrix, int smart);

void jerasure_free_schedule(int **schedule);
void jerasure_free_schedule_cache(int k, int m, int ***cache);


/* ------------------------------------------------------------ */
/* Encoding - these are all straightforward.  jerasure_matrix_encode only
   works with w = 8|16|32.  */

void jerasure_do_parity(int k, char **data_ptrs, char *parity_ptr, int size);

void jerasure_matrix_encode(int k, int m, int w, int *matrix,
                          char **data_ptrs, char **coding_ptrs, int size);

void jerasure_bitmatrix_encode(int k, int m, int w, int *bitmatrix,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize);

void jerasure_schedule_encode(int k, int m, int w, int **schedule,
                                  char **data_ptrs, char **coding_ptrs, int size, int packetsize);

/* ------------------------------------------------------------ */
/* Decoding. -------------------------------------------------- */

/* These return integers, because the matrix may not be invertible.

   The parameter row_k_ones should be set to 1 if row k of the matrix
   (or rows kw to (k+1)w+1) of th distribution matrix are all ones
   (or all identity matrices).  Then you can improve the performance
   of decoding when there is more than one failure, and the parity
   device didn't fail.  You do it by decoding all but one of the data
   devices, and then decoding the last data device from the data devices
   and the parity device.

   jerasure_schedule_decode_lazy generates the schedule on the fly.

   jerasure_matrix_decode only works when w = 8|16|32.

   jerasure_make_decoding_matrix/bitmatrix make the k*k decoding matrix
         (or wk*wk bitmatrix) by taking the rows corresponding to k
         non-erased devices of the distribution matrix, and then
         inverting that matrix.

         You should already have allocated the decoding matrix and
         dm_ids, which is a vector of k integers.  These will be
         filled in appropriately.  dm_ids[i] is the id of element
         i of the survivors vector.  I.e. row i of the decoding matrix
         times dm_ids equals data drive i.

         Both of these routines take "erased" instead of "erasures".
         Erased is a vector with k+m elements, which has 0 or 1 for
         each device's id, according to whether the device is erased.
```

include/jerasure.h lines 181 to 240

```
   jerasure_erasures_to_erased allocates and returns erased from erasures.

 */

int jerasure_matrix_decode(int k, int m, int w,
                          int *matrix, int row_k_ones, int *erasures,
                          char **data_ptrs, char **coding_ptrs, int size);

int jerasure_bitmatrix_decode(int k, int m, int w,
                            int *bitmatrix, int row_k_ones, int *erasures,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize);

int jerasure_schedule_decode_lazy(int k, int m, int w, int *bitmatrix, int *erasures,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize,
                            int smart);

int jerasure_schedule_decode_cache(int k, int m, int w, int ***scache, int *erasures,
                            char **data_ptrs, char **coding_ptrs, int size, int packetsize);

int jerasure_make_decoding_matrix(int k, int m, int w, int *matrix, int *erased,
                                  int *decoding_matrix, int *dm_ids);

int jerasure_make_decoding_bitmatrix(int k, int m, int w, int *matrix, int *erased,
                                  int *decoding_matrix, int *dm_ids);

int *jerasure_erasures_to_erased(int k, int m, int *erasures);

/* ------------------------------------------------------------ */
/* These perform dot products and schedules. -------------------*/
/*
   src_ids is a matrix of k id's (0 - k-1 for data devices, k - k+m-1
   for coding devices) that identify the source devices.  Dest_id is
   the id of the destination device.

   jerasure_matrix_dotprod only works when w = 8|16|32.

   jerasure_do_scheduled_operations executes the schedule on w*packetsize worth of
   bytes from each device.  ptrs is an array of pointers which should have as many
   elements as the highest referenced device in the schedule.

 */

void jerasure_matrix_dotprod(int k, int w, int *matrix_row,
                          int *src_ids, int dest_id,
                          char **data_ptrs, char **coding_ptrs, int size);

void jerasure_bitmatrix_dotprod(int k, int w, int *bitmatrix_row,
                             int *src_ids, int dest_id,
                             char **data_ptrs, char **coding_ptrs, int size, int packetsize);

void jerasure_do_scheduled_operations(char **ptrs, int **schedule, int packetsize);

/* ------------------------------------------------------------ */
/* Matrix Inversion ------------------------------------------- */
/*
   The two matrix inversion functions work on rows*rows matrices of
   ints.  If a bitmatrix, then each int will just be zero or one.
   Otherwise, they will be elements of gf(2^w).  Obviously, you can
   do bit matrices with crs_invert_matrix() and set w = 1, but
```

```
   crs_invert_bitmatrix will be more efficient.

   The two invertible functions return whether a matrix is invertible.
   They are more efficient than the inverstion functions.

   Mat will be destroyed when the matrix inversion or invertible
   testing is done.  Sorry.

   Inv must be allocated by the caller.

   The two invert_matrix functions return 0 on success, and -1 if the
   matrix is uninvertible.

   The two invertible function simply return whether the matrix is
   invertible.  (0 or 1). Mat will be destroyed.
 */

int jerasure_invert_matrix(int *mat, int *inv, int rows, int w);
int jerasure_invert_bitmatrix(int *mat, int *inv, int rows);
int jerasure_invertible_matrix(int *mat, int rows, int w);
int jerasure_invertible_bitmatrix(int *mat, int rows);

/* ------------------------------------------------------------ */
/* Basic matrix operations -------------------------------------*/
/*
   Each of the print_matrix routines require a w.  In jerasure_print_matrix,
   this is to calculate the field width.  In jerasure_print_bitmatrix, it is
   to put spaces between the bits.

   jerasure_matrix_multiply is a simple matrix multiplier in GF(2^w).  It returns a r1*c2
   matrix, which is the product of the two input matrices.  It allocates
   the product.  Obviously, c1 should equal r2.  However, this is not
   validated by the procedure.
*/

void jerasure_print_matrix(int *matrix, int rows, int cols, int w);
void jerasure_print_bitmatrix(int *matrix, int rows, int cols, int w);


int *jerasure_matrix_multiply(int *m1, int *m2, int r1, int c1, int r2, int c2, int w);

/* ------------------------------------------------------------ */
/* Stats ------------------------------------------------------ */
/*
  jerasure_get_stats fills in a vector of three doubles:

      fill_in[0] is the number of bytes that have been XOR'd
      fill_in[1] is the number of bytes that have been copied
      fill_in[2] is the number of bytes that have been multiplied
                 by a constant in GF(2^w)

  When jerasure_get_stats() is called, it resets its values.
 */

void jerasure_get_stats(double *fill_in);

int jerasure_autoconf_test();

#ifdef __cplusplus
}
```

include/jerasure.h lines 301 to 302

```
#endif
#endif
```

include/liberation.h lines 1 to 52

```
/* *
 * Copyright (c) 2013, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#pragma once

#ifdef __cplusplus
extern "C" {
#endif

extern int *liberation_coding_bitmatrix(int k, int w);
extern int *liber8tion_coding_bitmatrix(int k);
extern int *blaum_roth_coding_bitmatrix(int k, int w);

#ifdef __cplusplus
}
#endif
```

include/reed_sol.h lines 1 to 59

```
/* *
 * Copyright (c) 2013, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#pragma once

#ifdef __cplusplus
extern "C" {
#endif

extern int *reed_sol_vandermonde_coding_matrix(int k, int m, int w);
extern int *reed_sol_extended_vandermonde_matrix(int rows, int cols, int w);
extern int *reed_sol_big_vandermonde_distribution_matrix(int rows, int cols, int w);

extern int reed_sol_r6_encode(int k, int w, char **data_ptrs, char **coding_ptrs, int size);
extern int *reed_sol_r6_coding_matrix(int k, int w);

extern void reed_sol_galois_w08_region_multby_2(char *region, int nbytes);
extern void reed_sol_galois_w16_region_multby_2(char *region, int nbytes);
extern void reed_sol_galois_w32_region_multby_2(char *region, int nbytes);

#ifdef __cplusplus
}
#endif
```

include/timing.h lines 1 to 43

```
// Timing measurement utilities.

#ifndef JERASURE_INCLUDED__TIMING_H
#define JERASURE_INCLUDED__TIMING_H

// Define USE_CLOCK to use clock(). Otherwise use gettimeofday().
#define USE_CLOCK

#ifdef USE_CLOCK
#include <time.h>
#else
#include <sys/time.h>
#endif

struct timing {
#ifdef USE_CLOCK
  clock_t clock;
#else
  struct timeval tv;
#endif
};

// Get the current time as a double in seconds.
double
timing_now(
  void);

// Set *t to the current time.
void
timing_set(
  struct timing * t);

// Get *t as a double in seconds.
double
timing_get(
  struct timing * t);

// Return *t2 - *t1 as a double in seconds.
double
timing_delta(
  struct timing * t1,
  struct timing * t2);
#endif
```

Examples/cauchy_01.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "cauchy.h"
#include "jerasure.h"
#include "reed_sol.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: cauchy_01 n w - Converts the value n to a bitmatrix using GF(2^w).\n");
  fprintf(stderr, "       \n");
```

```
  fprintf(stderr, "       It prints the bitmatrix, and reports on the numberof ones.\n");
  fprintf(stderr, "       Use 0x to input n in hexadecimal.\n");
  fprintf(stderr, "       W must be <= 32.\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: cauchy_n_ones()\n");
  fprintf(stderr, "                   jerasure_matrix_to_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_print_bitmatrix()\n");
  if (s != NULL) fprintf(stderr, "\n%s\n", s);
  exit(1);
}

int main(int argc, char **argv)
{
  int n;
  int i, no, w;
  int *bitmatrix;

  if (argc != 3) usage(NULL);
  if (sscanf(argv[1], "0x%x", &n) == 0) {
    if (sscanf(argv[1], "%d", &n) == 0) usage("Bad n");
  }
  if (sscanf(argv[2], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (w == 31) {
    if (n & 0x80000000L) usage("Bad n/w combination (n not between 0 and 2^w-1)\n");
  } else if (w < 31) {
    if (n >= (1 << w)) usage("Bad n/w combination (n not between 0 and 2^w-1)\n");
  }

  bitmatrix = jerasure_matrix_to_bitmatrix(1, 1, w, &n);
  printf("<HTML><title>cauchy_01 %u %d</title>\n", w, n);
  printf("<HTML><h3>cauchy_01 %u %d</h3>\n", w, n);
  printf("<pre>\n");
  if (w == 32) {
    printf("Converted the value 0x%x to the following bitmatrix:\n\n", n);
  } else {
    printf("Converted the value %d (0x%x) to the following bitmatrix:\n\n", n, n);
  }
  jerasure_print_bitmatrix(bitmatrix, w, w, w);
  printf("\n");

  no = 0;
  for (i = 0; i < w*w; i++) no += bitmatrix[i];
  if (no != cauchy_n_ones(n, w)) {
    fprintf(stderr, "Jerasure error: # ones in the bitmatrix (%d) doesn't match cauchy_n_ones() (%d).\n",
       no, cauchy_n_ones(n, w));
    exit(1);
  }

  printf("# Ones: %d\n", cauchy_n_ones(n, w));

  return 0;
}
```

Examples/cauchy_02.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "cauchy.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: cauchy_02 k m w seed - CRS coding example using Bloemer's original matrix.\n");
  fprintf(stderr, "       \n");
```

```
  fprintf(stderr, "k+m must be <= 2^w\n");
  fprintf(stderr, "This sets up a generator matrix (G^T) in GF(2^w) whose last m rows are\n");
  fprintf(stderr, "created from a Cauchy matrix, using the original definition from [Bloemer95].\n");
  fprintf(stderr, "It converts this matrix to a bitmatrix, and then it encodes w packets from\n");
  fprintf(stderr, "each of k disks (simulated) onto w packets on each of m disks.  Packets are \n");
  fprintf(stderr, "simply longs.  Then, it deletes m random disks, and decodes.  \n");
  fprintf(stderr, "\n");
  fprintf(stderr, "The encoding and decoding are done twice, first, with jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "and jerasure_bitmatrix_decode(), and second using 'smart' scheduling with\n");
  fprintf(stderr, "jerasure_schedule_encode() and jerasure_schedule_decode_lazy().\n");

  fprintf(stderr, "\n");
  fprintf(stderr, "This demonstrates: cauchy_original_coding_matrix()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_decode()\n");
  fprintf(stderr, "                   cauchy_n_ones()\n");
  fprintf(stderr, "                   jerasure_smart_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_schedule_encode()\n");
  fprintf(stderr, "                   jerasure_schedule_decode_lazy()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   jerasure_print_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_get_stats()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, m;
  int *matrix, *bitmatrix, **schedule;
  char **data, **coding, **dcopy, **ccopy;
  int no;
  int *erasures, *erased;
```

Examples/cauchy_02.c lines 121 to 180

```
  double mstats[3], sstats[3];
  uint32_t seed;

  if (argc != 5) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (sscanf(argv[4], "%d", &seed) == 0) usage("Bad seed");
  if (w < 30 && (k+m) > (1 << w)) usage("k + m is too big");

  matrix = cauchy_original_coding_matrix(k, m, w);
  if (matrix == NULL) {
    usage("couldn't make coding matrix");
  }

  /* Print out header information to the output file. */
  printf("<HTML>\n");
  printf("<TITLE>Jerasure Example Output: cauchy_02 %d %d %d %d</TITLE>\n", k, m, w, seed);
  printf("<h2>Jerasure Example Output: cauchy_02 %d %d %d %d</h3>\n", k, m, w, seed);

  printf("<hr>\n");
  printf("Parameters:\n");
  printf("<UL><LI> Number of data disks <i>(k)</i>: %d\n", k);
  printf("<LI> Number of coding disks <i>(m)</i>: %d\n", m);
  printf("<LI> Word size of the Galois Field: <i>(w)</i>: %d\n", w);
  printf("<LI> Seed for the random number generator: %d\n", seed);
  printf("<LI> Number of bytes stored per disk: %ld\n", sizeof(long)*w);
  printf("<LI> Number of packets stored per disk: %d\n", w);
  printf("<LI> Number of bytes per packet: %ld\n", sizeof(long));
  printf("</UL>\n");

  /* Print out the matrix and the bitmatrix */
  printf("<hr>\n");
  printf("Here is the matrix, which was created with <b>cauchy_original_coding_matrix()</b>.\n");
  printf("This is not the best matrix to use, but we include it to show an example\n");
  printf("of <b>cauchy_original_coding_matrix()</b>.  For the best matrix and encoding/decoding\n");
  printf("methodology, see <b>cauchy_04.</b><p><pre>\n");

  jerasure_print_matrix(matrix, m, k, w);
  printf("</pre>\n");

  bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);

  no = 0;
  for (i = 0; i < k*m; i++) {
    no += cauchy_n_ones(matrix[i], w);
  }

  printf("The bitmatrix, which has %d one%s:<p><pre>\n", no, (no == 1) ? "" : "s");
  jerasure_print_bitmatrix(bitmatrix, m*w, k*w, w);
  printf("</pre>\n");
  printf("<hr>\n");
  MOA_Seed(seed);

  data = talloc(char *, k);
  dcopy = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long)*w);
    dcopy[i] = talloc(char, sizeof(long)*w);
    MOA_Fill_Random_Region(data[i], sizeof(long)*w);
```

Examples/cauchy_02.c lines 181 to 240

```
    memcpy(dcopy[i], data[i], sizeof(long)*w);
  }

  printf("Here are the packets on the data disks:<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");

  coding = talloc(char *, m);
  ccopy = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long)*w);
    ccopy[i] = talloc(char, sizeof(long)*w);
  }

  jerasure_bitmatrix_encode(k, m, w, bitmatrix, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(mstats);

  schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
  jerasure_schedule_encode(k, m, w, schedule, data, ccopy, w*sizeof(long), sizeof(long));
  jerasure_get_stats(sstats);

  printf("<p>Encoding with jerasure_bitmatrix_encode() - Bytes XOR'd: %.0lf.<br>\n", mstats[0]);
  printf("Encoding with jerasure_schedule_encode() - Bytes XOR'd: %.0lf.<br>\n", sstats[0]);

  for (i = 0; i < m; i++) {
    if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
      printf("Problem: the two encodings don't match on disk C%x\n", i);
      exit(0);
    }
  }

  printf("Here are the packets on the coding disks.<br>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(31, 1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
      i++;
    }
  }
  erasures[i] = -1;
  printf("Erasures on the following devices:");
  for (i = 0; erasures[i] != -1; i++) {
    printf(" %c%x", ((erasures[i] < k) ? 'D' : 'C'), (erasures[i] < k ? erasures[i] : erasures[i]-k));
  }
  printf("<br>\nHere is the state of the system:\n<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_bitmatrix_decode(k, m, w, bitmatrix, 0, erasures, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(mstats);

  printf("<p>Decoded with jerasure_bitmatrix_decode - Bytes XOR'd: %.0lf.<br>\n", mstats[0]);
```

Examples/cauchy_02.c lines 241 to 272

```
  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  for (i = 0; erasures[i] != -1; i++) {
    bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
  }

  jerasure_schedule_decode_lazy(k, m, w, bitmatrix, erasures, data, coding, w*sizeof(long), sizeof(long), 1);
  jerasure_get_stats(sstats);

  printf("jerasure_schedule_decode_lazy - Bytes XOR'd: %.0lf.<br>\n", sstats[0]);

  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  printf("Here is the state of the system:\n<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  return 0;
}
```

Examples/cauchy_03.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "cauchy.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: cauchy_03 k m w seed - CRS coding example improving the matrix.\n");
  fprintf(stderr, "       \n");
```

```
  fprintf(stderr, "k+m must be <= 2^w\n");
  fprintf(stderr, "This sets up a generator matrix (G^T) in GF(2^w) whose last m rows are\n");
  fprintf(stderr, "created from a Cauchy matrix, using the original definition from [Bloemer95].\n");
  fprintf(stderr, "This is done with cauchy_xy_coding_matrix().\n");
  fprintf(stderr, "\n");
  fprintf(stderr, "It then it improves the matrix, which yields a different bitmatrix that is\n");
  fprintf(stderr, "MDS like the original bitmatrix, but it will yield a bitmatrix with fewer ones.\n");
  fprintf(stderr, "Then, it encodes w packets from each of k disks (simulated) onto w packets on\n");
  fprintf(stderr, "on each of m disks.  Packets are longs.  Then, it deletes m random disks, and decodes.\n");
  fprintf(stderr, "\n");
  fprintf(stderr, "The encoding and decoding are done twice, first, with jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "and jerasure_bitmatrix_decode(), and second using 'smart' scheduling with\n");
  fprintf(stderr, "jerasure_schedule_encode() and jerasure_schedule_decode_lazy().\n");

  fprintf(stderr, "\n");
  fprintf(stderr, "This demonstrates: cauchy_xy_coding_matrix()\n");
  fprintf(stderr, "                   cauchy_improve_coding_matrix()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_decode()\n");
  fprintf(stderr, "                   cauchy_n_ones()\n");
  fprintf(stderr, "                   jerasure_smart_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_schedule_encode()\n");
  fprintf(stderr, "                   jerasure_schedule_decode_lazy()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   jerasure_print_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_get_stats()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, m;
```

Examples/cauchy_03.c lines 121 to 180

```
  int *matrix, *bitmatrix, **schedule;
  char **data, **coding, **dcopy, **ccopy;
  int no;
  int *erasures, *erased;
  double mstats[3], sstats[3];
  uint32_t seed;
  int *X, *Y;

  if (argc != 5) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (sscanf(argv[4], "%d", &seed) == 0) usage("Bad seed");
  if (w < 30 && (k+m) > (1 << w)) usage("k + m is too big");

  X = talloc(int, m);
  Y = talloc(int, k);
  for (i = 0; i < m; i++) X[i] = i;
  for (i = 0; i < k; i++) Y[i] = m+i;

  matrix = cauchy_xy_coding_matrix(k, m, w, X, Y);
  if (matrix == NULL) {
    usage("couldn't make coding matrix");
  }

  /* Print out header information to the output file. */
  printf("<HTML>\n");
  printf("<TITLE>Jerasure Example Output: cauchy_03 %d %d %d %d</TITLE>\n", k, m, w, seed);
  printf("<h2>Jerasure Example Output: cauchy_03 %d %d %d %d</h3>\n", k, m, w, seed);

  printf("<hr>\n");
  printf("Parameters:\n");
  printf("<UL><LI> Number of data disks <i>(k)</i>: %d\n", k);
  printf("<LI> Number of coding disks <i>(m)</i>: %d\n", m);
  printf("<LI> Word size of the Galois Field: <i>(w)</i>: %d\n", w);
  printf("<LI> Seed for the random number generator: %d\n", seed);
  printf("<LI> Number of bytes stored per disk: %ld\n", sizeof(long)*w);
  printf("<LI> Number of packets stored per disk: %d\n", w);
  printf("<LI> Number of bytes per packet: %ld\n", sizeof(long));
  printf("</UL>\n");

  /* Print out the matrix and the bitmatrix */
  printf("<hr>\n");
  printf("Here is the matrix, which was created with <b>cauchy_xy_coding_matrix()</b>.\n");

  printf("<pre>\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("</pre>\n");

  cauchy_improve_coding_matrix(k, m, w, matrix);

  printf("<hr>\n");
  printf("Here is the matrix improved with <b>cauchy_improve_coding_matrix()</b>.\n");

  printf("<pre>\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("</pre>\n");

  bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);
```

Examples/cauchy_03.c lines 181 to 240

```
  no = 0;
  for (i = 0; i < k*m; i++) {
    no += cauchy_n_ones(matrix[i], w);
  }

  printf("The bitmatrix, which has %d one%s:<p><pre>\n", no, (no == 1) ? "" : "s");
  jerasure_print_bitmatrix(bitmatrix, m*w, k*w, w);
  printf("</pre>\n");
  printf("<hr>\n");

  MOA_Seed(seed);

  data = talloc(char *, k);
  dcopy = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long)*w);
    dcopy[i] = talloc(char, sizeof(long)*w);
    MOA_Fill_Random_Region(data[i], sizeof(long)*w);
    memcpy(dcopy[i], data[i], sizeof(long)*w);
  }

  printf("Here are the packets on the data disks:<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");

  coding = talloc(char *, m);
  ccopy = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long)*w);
    ccopy[i] = talloc(char, sizeof(long)*w);
  }

  jerasure_bitmatrix_encode(k, m, w, bitmatrix, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(mstats);

  schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
  jerasure_schedule_encode(k, m, w, schedule, data, ccopy, w*sizeof(long), sizeof(long));
  jerasure_get_stats(sstats);

  printf("<p>Encoding with jerasure_bitmatrix_encode() - Bytes XOR'd: %.0lf.<br>\n", mstats[0]);
  printf("Encoding with jerasure_schedule_encode() - Bytes XOR'd: %.0lf.<br>\n", sstats[0]);

  for (i = 0; i < m; i++) {
    if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
      printf("Problem: the two encodings don't match on disk C%x\n", i);
      exit(0);
    }
  }

  printf("Here are the packets on the coding disks.<br>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(31, 1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
```

```
      i++;
    }
  }
  erasures[i] = -1;
  printf("Erasures on the following devices:");
  for (i = 0; erasures[i] != -1; i++) {
    printf(" %c%x", ((erasures[i] < k) ? 'D' : 'C'), (erasures[i] < k ? erasures[i] : erasures[i]-k));
  }
  printf("<br>\nHere is the state of the system:\n<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_bitmatrix_decode(k, m, w, bitmatrix, 0, erasures, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(mstats);

  printf("<p>Decoded with jerasure_bitmatrix_decode - Bytes XOR'd: %.0lf.<br>\n", mstats[0]);

  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  for (i = 0; erasures[i] != -1; i++) {
    bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
  }

  jerasure_schedule_decode_lazy(k, m, w, bitmatrix, erasures, data, coding, w*sizeof(long), sizeof(long), 1);
  jerasure_get_stats(sstats);

  printf("jerasure_schedule_decode_lazy - Bytes XOR'd: %.0lf.<br>\n", sstats[0]);

  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  printf("Here is the state of the system:\n<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  return 0;
}
```

Examples/cauchy_04.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "cauchy.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: cauchy_04 k m w seed - CRS coding example improving the matrix.\n");
  fprintf(stderr, "       \n");
```

Examples/cauchy_04.c lines 61 to 120

```
  fprintf(stderr, "k+m must be <= 2^w\n");
  fprintf(stderr, "This sets up a generator matrix (G^T) in GF(2^w) whose last m rows are\n");
  fprintf(stderr, "a 'good' matrix, created with cauchy_good_general_coding_matrix().\n");
  fprintf(stderr, "It converts this matrix to a bitmatrix.\n");
  fprintf(stderr, "\n");
  fprintf(stderr, "Then, it encodes w packets from each of k disks (simulated) onto w packets on\n");
  fprintf(stderr, "on each of m disks.  Packets are longs.  Then, it deletes m random disks, and decodes.\n");
  fprintf(stderr, "\n");
  fprintf(stderr, "The encoding and decoding are done twice, first, with jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "and jerasure_bitmatrix_decode(), and second using 'smart' scheduling with\n");
  fprintf(stderr, "jerasure_schedule_encode() and jerasure_schedule_decode_lazy().\n");

  fprintf(stderr, "\n");
  fprintf(stderr, "This demonstrates: cauchy_good_general_coding_matrix()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_decode()\n");
  fprintf(stderr, "                   cauchy_n_ones()\n");
  fprintf(stderr, "                   jerasure_smart_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_schedule_encode()\n");
  fprintf(stderr, "                   jerasure_schedule_decode_lazy()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   jerasure_print_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_get_stats()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, m;
  int *matrix, *bitmatrix, **schedule;
  char **data, **coding, **dcopy, **ccopy;
  int no;
```

```
  int *erasures, *erased;
  double mstats[3], sstats[3];
  uint32_t seed;

  if (argc != 5) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (sscanf(argv[4], "%d", &seed) == 0) usage("Bad seed");
  if (w < 30 && (k+m) > (1 << w)) usage("k + m is too big");

  matrix = cauchy_good_general_coding_matrix(k, m, w);
  if (matrix == NULL) {
    usage("couldn't make coding matrix");
  }

  /* Print out header information to the output file. */
  printf("<HTML>\n");
  printf("<TITLE>Jerasure Example Output: cauchy_04 %d %d %d %d</TITLE>\n", k, m, w, seed);
  printf("<h2>Jerasure Example Output: cauchy_04 %d %d %d %d</h3>\n", k, m, w, seed);

  printf("<hr>\n");
  printf("Parameters:\n");
  printf("<UL><LI> Number of data disks <i>(k)</i>: %d\n", k);
  printf("<LI> Number of coding disks <i>(m)</i>: %d\n", m);
  printf("<LI> Word size of the Galois Field: <i>(w)</i>: %d\n", w);
  printf("<LI> Seed for the random number generator: %d\n", seed);
  printf("<LI> Number of bytes stored per disk: %ld\n", sizeof(long)*w);
  printf("<LI> Number of packets stored per disk: %d\n", w);
  printf("<LI> Number of bytes per packet: %ld\n", sizeof(long));
  printf("</UL>\n");

  /* Print out the matrix and the bitmatrix */
  printf("<hr>\n");
  printf("Here is the matrix, which was created with <b>cauchy_good_general_coding_matrix()</b>.\n");

  printf("<pre>\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("</pre>\n");

  bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);

  no = 0;
  for (i = 0; i < k*m; i++) {
    no += cauchy_n_ones(matrix[i], w);
  }

  printf("The bitmatrix, which has %d one%s:<p><pre>\n", no, (no == 1) ? "" : "s");
  jerasure_print_bitmatrix(bitmatrix, m*w, k*w, w);
  printf("</pre>\n");
  printf("<hr>\n");

  MOA_Seed(seed);

  data = talloc(char *, k);
  dcopy = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long)*w);
    dcopy[i] = talloc(char, sizeof(long)*w);
    MOA_Fill_Random_Region(data[i], sizeof(long)*w);
```

Examples/cauchy_04.c lines 181 to 240

```
    memcpy(dcopy[i], data[i], sizeof(long)*w);
  }

  printf("Here are the packets on the data disks:<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");

  coding = talloc(char *, m);
  ccopy = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long)*w);
    ccopy[i] = talloc(char, sizeof(long)*w);
  }

  jerasure_bitmatrix_encode(k, m, w, bitmatrix, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(mstats);

  schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
  jerasure_schedule_encode(k, m, w, schedule, data, ccopy, w*sizeof(long), sizeof(long));
  jerasure_get_stats(sstats);

  printf("<p>Encoding with jerasure_bitmatrix_encode() - Bytes XOR'd: %.0lf.<br>\n", mstats[0]);
  printf("Encoding with jerasure_schedule_encode() - Bytes XOR'd: %.0lf.<br>\n", sstats[0]);

  for (i = 0; i < m; i++) {
    if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
      printf("Problem: the two encodings don't match on disk C%x\n", i);
      exit(0);
    }
  }

  printf("Here are the packets on the coding disks.<br>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(31, 1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
      i++;
    }
  }
  erasures[i] = -1;
  printf("Erasures on the following devices:");
  for (i = 0; erasures[i] != -1; i++) {
    printf(" %c%x", ((erasures[i] < k) ? 'D' : 'C'), (erasures[i] < k ? erasures[i] : erasures[i]-k));
  }
  printf("<br>\nHere is the state of the system:\n<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_bitmatrix_decode(k, m, w, bitmatrix, 0, erasures, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(mstats);

  printf("<p>Decoded with jerasure_bitmatrix_decode - Bytes XOR'd: %.0lf.<br>\n", mstats[0]);
```

Examples/cauchy_04.c lines 241 to 272

```
  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  for (i = 0; erasures[i] != -1; i++) {
    bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
  }

  jerasure_schedule_decode_lazy(k, m, w, bitmatrix, erasures, data, coding, w*sizeof(long), sizeof(long), 1);
  jerasure_get_stats(sstats);

  printf("jerasure_schedule_decode_lazy - Bytes XOR'd: %.0lf.<br>\n", sstats[0]);

  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)*w) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  printf("Here is the state of the system:\n<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  return 0;
}
```

Examples/decoder.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

/*
This program takes as input an inputfile, k, m, a coding
technique, w, and packetsize.  It is the companion program
of encoder.c, which creates k+m files.  This program assumes
that up to m erasures have occurred in the k+m files.  It
reads in the k+m files or marks the file as erased. It then
recreates the original file and creates a new file with the
suffix "decoded" with the decoded contents of the file.

This program does not error check command line arguments because
it is assumed that encoder.c has been called previously with the
same arguments, and encoder.c does error check.
*/
```

Examples/decoder.c lines 61 to 120

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/stat.h>
#include <signal.h>
#include <unistd.h>
#include "jerasure.h"
#include "reed_sol.h"
#include "galois.h"
#include "cauchy.h"
#include "liberation.h"
#include "timing.h"

#define N 10

enum Coding_Technique {
    Reed_Sol_Van, Reed_Sol_R6_Op, Cauchy_Orig, Cauchy_Good,
    Liberation, Blaum_Roth, Liber8tion, RDP, EVENODD, No_Coding
};

char           *Methods[N] = {"reed_sol_van", "reed_sol_r6_op", "cauchy_orig", "cauchy_good",
"liberation", "blaum_roth", "liber8tion", "rdp", "evenodd", "no_coding"};

/* Global variables for signal handler */
enum Coding_Technique method;
int             readins, n;

/* Function prototype */
void            ctrl_bs_handler(int dummy);

int
main(int argc, char **argv)
{
    FILE           *fp;
    //File pointer

    /* Jerasure arguments */
        char           **data;
    char          **coding;
    int            *erasures;
    int            *erased;
    int            *matrix;
    int            *bitmatrix;

    /* Parameters */
    int             k, m, w, packetsize, buffersize;
    int             tech;
    char           *c_tech;

    int             i, j;
    //loop control variable, s
        int             blocksize;
    //size of individual files
        int             origsize;
    //size of file before padding
        int             total;
    //used to write data, not padding to file
```

```
        struct stat     status;
    //used to find size of individual files
        int             numerased;
    //number of erased files
        int             dummy;

    /* Used to recreate file names */
    char            *temp;
    char            *cs1, *cs2, *extension;
    char            *fname;
    int              md;
    char            *curdir;

    /* Used to time decoding */
    struct timing    t1, t2, t3, t4;
    double           tsec;
    double           totalsec;


    signal(SIGQUIT, ctrl_bs_handler);

    matrix = NULL;
    bitmatrix = NULL;
    totalsec = 0.0;

    /* Start timing */
    timing_set(&t1);

    /* Error checking parameters */
    if (argc != 2) {
        fprintf(stderr, "usage: inputfile\n");
        exit(0);
    }
    curdir = (char *) malloc(sizeof(char) * 100);
    getcwd(curdir, 100);

    /* Begin recreation of file names */
    cs1 = (char *) malloc(sizeof(char) * strlen(argv[1]));
    cs2 = strrchr(argv[1], '/');
    if (cs2 != NULL) {
        cs2++;
        strcpy(cs1, cs2);
    } else {
        strcpy(cs1, argv[1]);
    }
    cs2 = strchr(cs1, '.');
    if (cs2 != NULL) {
        extension = strdup(cs2);
        *cs2 = '\0';
    } else {
        extension = strdup("");
    }
    fname = (char *) malloc(sizeof(char *) * (100 + strlen(argv[1]) + 20));

    /* Read in parameters from metadata file */
    sprintf(fname, "%s/Coding/%s_meta.txt", curdir, cs1);

    fp = fopen(fname, "rb");
    if (fp == NULL) {
        fprintf(stderr, "Error: no metadata file %s\n", fname);
```

Examples/decoder.c lines 181 to 240

```
        exit(1);
    }
    temp = (char *) malloc(sizeof(char) * (strlen(argv[1]) + 20));
    if (fscanf(fp, "%s", temp) != 1) {
        fprintf(stderr, "Metadata file - bad format\n");
        exit(0);
    }
    if (fscanf(fp, "%d", &origsize) != 1) {
        fprintf(stderr, "Original size is not valid\n");
        exit(0);
    }
    if (fscanf(fp, "%d %d %d %d %d", &k, &m, &w, &packetsize, &buffersize) != 5) {
        fprintf(stderr, "Parameters are not correct\n");
        exit(0);
    }
    c_tech = (char *) malloc(sizeof(char) * (strlen(argv[1]) + 20));
    if (fscanf(fp, "%s", c_tech) != 1) {
        fprintf(stderr, "Metadata file - bad format\n");
        exit(0);
    }
    if (fscanf(fp, "%d", &tech) != 1) {
        fprintf(stderr, "Metadata file - bad format\n");
        exit(0);
    }
    method = tech;
    if (fscanf(fp, "%d", &readins) != 1) {
        fprintf(stderr, "Metadata file - bad format\n");
        exit(0);
    }
    fclose(fp);

    /* Allocate memory */
    erased = (int *) malloc(sizeof(int) * (k + m));
    for (i = 0; i < k + m; i++)
        erased[i] = 0;
    erasures = (int *) malloc(sizeof(int) * (k + m));

    data = (char **) malloc(sizeof(char *) * k);
    coding = (char **) malloc(sizeof(char *) * m);
    if (buffersize != origsize) {
        for (i = 0; i < k; i++) {
            data[i] = (char *) malloc(sizeof(char) * (buffersize / k));
        }
        for (i = 0; i < m; i++) {
            coding[i] = (char *) malloc(sizeof(char) * (buffersize / k));
        }
        blocksize = buffersize / k;
    }
    sprintf(temp, "%d", k);
    md = strlen(temp);
    timing_set(&t3);

    /* Create coding matrix or bitmatrix */
    switch (tech) {
    case No_Coding:
        break;
    case Reed_Sol_Van:
        matrix = reed_sol_vandermonde_coding_matrix(k, m, w);
        break;
    case Reed_Sol_R6_Op:
```

```
        matrix = reed_sol_r6_coding_matrix(k, w);
        break;
    case Cauchy_Orig:
        matrix = cauchy_original_coding_matrix(k, m, w);
        bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);
        break;
    case Cauchy_Good:
        matrix = cauchy_good_general_coding_matrix(k, m, w);
        bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);
        break;
    case Liberation:
        bitmatrix = liberation_coding_bitmatrix(k, w);
        break;
    case Blaum_Roth:
        bitmatrix = blaum_roth_coding_bitmatrix(k, w);
        break;
    case Liber8tion:
        bitmatrix = liber8tion_coding_bitmatrix(k);
    }
    timing_set(&t4);
    totalsec += timing_delta(&t3, &t4);

    /* Begin decoding process */
    total = 0;
    n = 1;
    while (n <= readins) {
        numerased = 0;
        /* Open files, check for erasures, read in data/coding */
        for (i = 1; i <= k; i++) {
            sprintf(fname, "%s/Coding/%s_k%0*d%s", curdir, cs1, md, i, extension);
            fp = fopen(fname, "rb");
            if (fp == NULL) {
                erased[i - 1] = 1;
                erasures[numerased] = i - 1;
                numerased++;
                //printf("%s failed\n", fname);
            } else {
                if (buffersize == origsize) {
                    stat(fname, &status);
                    blocksize = status.st_size;
                    data[i - 1] = (char *) malloc(sizeof(char) * blocksize);
                    dummy = fread(data[i - 1], sizeof(char), blocksize, fp);
                } else {
                    fseek(fp, blocksize * (n - 1), SEEK_SET);
                    dummy = fread(data[i - 1], sizeof(char), buffersize / k, fp);
                }
                fclose(fp);
            }
        }
        for (i = 1; i <= m; i++) {
            sprintf(fname, "%s/Coding/%s_m%0*d%s", curdir, cs1, md, i, extension);
            fp = fopen(fname, "rb");
            if (fp == NULL) {
                erased[k + (i - 1)] = 1;
                erasures[numerased] = k + i - 1;
                numerased++;
                //printf("%s failed\n", fname);
            } else {
                if (buffersize == origsize) {
                    stat(fname, &status);
```

```
                    blocksize = status.st_size;
                    coding[i - 1] = (char *) malloc(sizeof(char) * blocksize);
                    dummy = fread(coding[i - 1], sizeof(char), blocksize, fp);
                } else {
                    fseek(fp, blocksize * (n - 1), SEEK_SET);
                    dummy = fread(coding[i - 1], sizeof(char), blocksize, fp);
                }
                fclose(fp);
            }
        }
        /* Finish allocating data/coding if needed */
        if (n == 1) {
            for (i = 0; i < numerased; i++) {
                if (erasures[i] < k) {
                    data[erasures[i]] = (char *) malloc(sizeof(char) * blocksize);
                } else {
                    coding[erasures[i] - k] = (char *) malloc(sizeof(char) * blocksize);
                }
            }
        }
        erasures[numerased] = -1;
        timing_set(&t3);

        /* Choose proper decoding method */
        if (tech == Reed_Sol_Van || tech == Reed_Sol_R6_Op) {
            i = jerasure_matrix_decode(k, m, w, matrix, 1, erasures, data, coding, blocksize);
        } else if (tech == Cauchy_Orig || tech == Cauchy_Good || tech == Liberation ||
                   tech == Blaum_Roth || tech == Liber8tion) {
            i = jerasure_schedule_decode_lazy(k, m, w, bitmatrix,
                          erasures, data, coding, blocksize, packetsize, 1);
        } else {
            fprintf(stderr, "Not a valid coding technique.\n");
            exit(0);
        }
        timing_set(&t4);

        /* Exit if decoding was unsuccessful */
        if (i == -1) {
            fprintf(stderr, "Unsuccessful!\n");
            exit(0);
        }
        /* Create decoded file */
        sprintf(fname, "%s/Coding/%s_decoded%s", curdir, cs1, extension);
        if (n == 1) {
            fp = fopen(fname, "wb");
        } else {
            fp = fopen(fname, "ab");
        }
        for (i = 0; i < k; i++) {
            if (total + blocksize <= origsize) {
                fwrite(data[i], sizeof(char), blocksize, fp);
                total += blocksize;
            } else {
                for (j = 0; j < blocksize; j++) {
                    if (total < origsize) {
                        fprintf(fp, "%c", data[i][j]);
                        total++;
                    } else {
                        break;
                    }
```

```
                }
            }
        }
        n++;
        fclose(fp);
        totalsec += timing_delta(&t3, &t4);
    }

    /* Free allocated memory */
    free(cs1);
    free(extension);
    free(fname);
    free(data);
    free(coding);
    free(erasures);
    free(erased);

    /* Stop timing and print time */
    timing_set(&t2);
    tsec = timing_delta(&t1, &t2);
    printf("Decoding (MB/sec): %0.10f\n", (((double) origsize) / 1024.0 / 1024.0) / totalsec);
    printf("De_Total (MB/sec): %0.10f\n\n", (((double) origsize) / 1024.0 / 1024.0) / tsec);

    return 0;
}

void
ctrl_bs_handler(int dummy)
{
    time_t          mytime;
    mytime = time(0);
    fprintf(stderr, "\n%s\n", ctime(&mytime));
    fprintf(stderr, "You just typed ctrl-\\ in decoder.c\n");
    fprintf(stderr, "Total number of read ins = %d\n", readins);
    fprintf(stderr, "Current read in: %d\n", n);
    fprintf(stderr, "Method: %s\n\n", Methods[method]);
    signal(SIGQUIT, ctrl_bs_handler);
}
```

Examples/encoder.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

/*

This program takes as input an inputfile, k, m, a coding
technique, w, and packetsize.  It creates k+m files from
the original file so that k of these files are parts of
the original file and m of the files are encoded based on
the given coding technique. The format of the created files
is the file name with "_k#" or "_m#" and then the extension.
(For example, inputfile test.txt would yield file "test_k1.txt".)
*/

#include <assert.h>
#include <time.h>
#include <sys/time.h>
```

Examples/encoder.c lines 61 to 120

```
#include <sys/stat.h>
#include <unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <signal.h>
#include <gf_rand.h>
#include <unistd.h>
#include "jerasure.h"
#include "reed_sol.h"
#include "cauchy.h"
#include "liberation.h"
#include "timing.h"

#define N 10

enum Coding_Technique {
    Reed_Sol_Van, Reed_Sol_R6_Op, Cauchy_Orig, Cauchy_Good, Liberation, Blaum_Roth,
    Liber8tion, RDP, EVENODD, No_Coding
};

char *Methods[N] = {"reed_sol_van", "reed_sol_r6_op", "cauchy_orig", "cauchy_good",
    "liberation", "blaum_roth", "liber8tion", "no_coding"};

/* Global variables for signal handler */
int              readins, n;
enum Coding_Technique method;

/* Function prototypes */
int              is_prime(int w);
void             ctrl_bs_handler(int dummy);

int
jfread(void *ptr, int size, int nmembers, FILE * stream)
{
    if (stream != NULL)
        return fread(ptr, size, nmembers, stream);

    MOA_Fill_Random_Region(ptr, size);
    return size;
}


int
main(int argc, char **argv)
{
    FILE            *fp, *fp2;
    //file pointers
        char            *block;
    //padding file
        int              size, newsize;
    //size of file and temp size
        struct stat      status;
    //finding file size


        enum Coding_Technique tech;
    //coding technique(parameter)
        int              k, m, w, packetsize;
```

Examples/encoder.c lines 121 to 180

```
    //parameters
        int             buffersize;
    //paramter
        int             i;
    //loop control variables
        int             blocksize;
    //size of k + m files
        int             total;
    int             extra;

    /* Jerasure Arguments */
    char            **data;
    char            **coding;
    int              *matrix;
    int              *bitmatrix;
    int             **schedule;

    /* Creation of file name variables */
    char             temp[5];
    char            *s1, *s2, *extension;
    char            *fname;
    int              md;
    char            *curdir;

    /* Timing variables */
    struct timing   t1, t2, t3, t4;
    double          tsec;
    double          totalsec;
    struct timing   start;

    /* Find buffersize */
    int             up, down;


    signal(SIGQUIT, ctrl_bs_handler);

    /* Start timing */
    timing_set(&t1);
    totalsec = 0.0;
    matrix = NULL;
    bitmatrix = NULL;
    schedule = NULL;

    /* Error check Arguments */
    if (argc != 8) {
        fprintf(stderr, "usage: inputfile k m coding_technique w packetsize buffersize\n");
        fprintf(stderr, "\nChoose one of the following coding techniques: \nreed_sol_van,");
        fprintf(stderr, "\nreed_sol_r6_op, \ncauchy_orig, \ncauchy_good, \nliberation, \nblaum_roth, \nliber8tion");
        fprintf(stderr, "\n\nPacketsize is ignored for the reed_sol's");
        fprintf(stderr, "\nBuffersize of 0 means the buffersize is chosen automatically.\n");
        fprintf(stderr, "\nIf you just want to test speed, use an inputfile of \"-number\" ");
        fprintf(stderr, "where number is the size of the fake file you want to test.\n\n");
        exit(0);
    }
    /* Conversion of parameters and error checking */
    if (sscanf(argv[2], "%d", &k) == 0 || k <= 0) {
        fprintf(stderr, "Invalid value for k\n");
        exit(0);
    }
    if (sscanf(argv[3], "%d", &m) == 0 || m < 0) {
```

Examples/encoder.c lines 181 to 240

```
        fprintf(stderr, "Invalid value for m\n");
        exit(0);
    }
    if (sscanf(argv[5], "%d", &w) == 0 || w <= 0) {
        fprintf(stderr, "Invalid value for w.\n");
        exit(0);
    }
    if (argc == 6) {
        packetsize = 0;
    } else {
        if (sscanf(argv[6], "%d", &packetsize) == 0 || packetsize < 0) {
            fprintf(stderr, "Invalid value for packetsize.\n");
            exit(0);
        }
    }
    if (argc != 8) {
        buffersize = 0;
    } else {
        if (sscanf(argv[7], "%d", &buffersize) == 0 || buffersize < 0) {
            fprintf(stderr, "Invalid value for buffersize\n");
            exit(0);
        }
    }

    /*
     * Determine proper buffersize by finding the closest valid buffersize to
     * the input value
     */
    if (buffersize != 0) {
        if (packetsize != 0 && buffersize % (sizeof(long) * w * k * packetsize) != 0) {
            up = buffersize;
            down = buffersize;
            while (up % (sizeof(long) * w * k * packetsize) != 0 && (down % (sizeof(long) * w * k * packetsize) !=0)) {
                up++;
                if (down == 0) {
                    down--;
                }
            }
            if (up % (sizeof(long) * w * k * packetsize) == 0) {
                buffersize = up;
            } else {
                if (down != 0) {
                    buffersize = down;
                }
            }
        } else if (packetsize == 0 && buffersize % (sizeof(long) * w * k) != 0) {
            up = buffersize;
            down = buffersize;
            while (up % (sizeof(long) * w * k) != 0 && down % (sizeof(long) * w * k) != 0) {
                up++;
                down--;
            }
            if (up % (sizeof(long) * w * k) == 0) {
                buffersize = up;
            } else {
                buffersize = down;
            }
        }
    }
    /* Setting of coding technique and error checking */
```

```
    if (strcmp(argv[4], "no_coding") == 0) {
        tech = No_Coding;
    } else if (strcmp(argv[4], "reed_sol_van") == 0) {
        tech = Reed_Sol_Van;
        if (w != 8 && w != 16 && w != 32) {
            fprintf(stderr, "w must be one of {8, 16, 32}\n");
            exit(0);
        }
    } else if (strcmp(argv[4], "reed_sol_r6_op") == 0) {
        if (m != 2) {
            fprintf(stderr, "m must be equal to 2\n");
            exit(0);
        }
        if (w != 8 && w != 16 && w != 32) {
            fprintf(stderr, "w must be one of {8, 16, 32}\n");
            exit(0);
        }
        tech = Reed_Sol_R6_Op;
    } else if (strcmp(argv[4], "cauchy_orig") == 0) {
        tech = Cauchy_Orig;
        if (packetsize == 0) {
            fprintf(stderr, "Must include packetsize.\n");
            exit(0);
        }
    } else if (strcmp(argv[4], "cauchy_good") == 0) {
        tech = Cauchy_Good;
        if (packetsize == 0) {
            fprintf(stderr, "Must include packetsize.\n");
            exit(0);
        }
    } else if (strcmp(argv[4], "liberation") == 0) {
        if (k > w) {
            fprintf(stderr, "k must be less than or equal to w\n");
            exit(0);
        }
        if (w <= 2 || !(w % 2) || !is_prime(w)) {
            fprintf(stderr, "w must be greater than two and w must be prime\n");
            exit(0);
        }
        if (packetsize == 0) {
            fprintf(stderr, "Must include packetsize.\n");
            exit(0);
        }
        if ((packetsize % (sizeof(long))) != 0) {
            fprintf(stderr, "packetsize must be a multiple of sizeof(long)\n");
            exit(0);
        }
        tech = Liberation;
    } else if (strcmp(argv[4], "blaum_roth") == 0) {
        if (k > w) {
            fprintf(stderr, "k must be less than or equal to w\n");
            exit(0);
        }
        if (w <= 2 || !((w + 1) % 2) || !is_prime(w + 1)) {
            fprintf(stderr, "w must be greater than two and w+1 must be prime\n");
            exit(0);
        }
        if (packetsize == 0) {
            fprintf(stderr, "Must include packetsize.\n");
```

```
            exit(0);
        }
        if ((packetsize % (sizeof(long))) != 0) {
            fprintf(stderr, "packetsize must be a multiple of sizeof(long)\n");
            exit(0);
        }
        tech = Blaum_Roth;
    } else if (strcmp(argv[4], "liber8tion") == 0) {
        if (packetsize == 0) {
            fprintf(stderr, "Must include packetsize\n");
            exit(0);
        }
        if (w != 8) {
            fprintf(stderr, "w must equal 8\n");
            exit(0);
        }
        if (m != 2) {
            fprintf(stderr, "m must equal 2\n");
            exit(0);
        }
        if (k > w) {
            fprintf(stderr, "k must be less than or equal to w\n");
            exit(0);
        }
        tech = Liber8tion;
    } else {
        fprintf(stderr, "Not a valid coding technique. Choose one of the following:");
        fprintf(stderr, "reed_sol_van, reed_sol_r6_op, cauchy_orig, cauchy_good\n");
        fprintf(stderr, "liberation, blaum_roth, liber8tion, no_coding\n");
        exit(0);
    }

    /* Set global variable method for signal handler */
    method = tech;

    /* Get current working directory for construction of file names */
    curdir = (char *) malloc(sizeof(char) * 1000);
    getcwd(curdir, 1000);

    if (argv[1][0] != '-') {

        /* Open file and error check */
        fp = fopen(argv[1], "rb");
        if (fp == NULL) {
            fprintf(stderr, "Unable to open file.\n");
            exit(0);
        }
        /* Create Coding directory */
        i = mkdir("Coding", S_IRWXU);
        if (i == -1 && errno != EEXIST) {
            fprintf(stderr, "Unable to create Coding directory.\n");
            exit(0);
        }
        /* Determine original size of file */
        stat(argv[1], &status);
        size = status.st_size;
    } else {
        if (sscanf(argv[1] + 1, "%d", &size) != 1 || size <= 0) {
            fprintf(stderr, "Files starting with '-' should be sizes for randomly created input\n");
            exit(1);
```

```
		}
		fp = NULL;
		MOA_Seed(time(0));
	}

	newsize = size;

	/* Find new size by determining next closest multiple */
	if (packetsize != 0) {
		if (size % (k * w * packetsize * sizeof(long)) != 0) {
			while (newsize % (k * w * packetsize * sizeof(long)) != 0)
				newsize++;
		}
	} else {
		if (size % (k * w * sizeof(long)) != 0) {
			while (newsize % (k * w * sizeof(long)) != 0)
				newsize++;
		}
	}

	if (buffersize != 0) {
		while (newsize % buffersize != 0) {
			newsize++;
		}
	}
	/* Determine size of k+m files */
	blocksize = newsize / k;

	/* Allow for buffersize and determine number of read-ins */
	if (size > buffersize && buffersize != 0) {
		if (newsize % buffersize != 0) {
			readins = newsize / buffersize;
		} else {
			readins = newsize / buffersize;
		}
		block = (char *) malloc(sizeof(char) * buffersize);
		blocksize = buffersize / k;
	} else {
		readins = 1;
		buffersize = size;
		block = (char *) malloc(sizeof(char) * newsize);
	}

	/* Break inputfile name into the filename and extension */
	s1 = (char *) malloc(sizeof(char) * (strlen(argv[1]) + 20));
	s2 = strrchr(argv[1], '/');
	if (s2 != NULL) {
		s2++;
		strcpy(s1, s2);
	} else {
		strcpy(s1, argv[1]);
	}
	s2 = strchr(s1, '.');
	if (s2 != NULL) {
		extension = strdup(s2);
		*s2 = '\0';
	} else {
		extension = strdup("");
	}
```

Examples/encoder.c lines 421 to 480

```
    /* Allocate for full file name */
    fname = (char *) malloc(sizeof(char) * (strlen(argv[1]) + strlen(curdir) + 20));
    sprintf(temp, "%d", k);
    md = strlen(temp);

    /* Allocate data and coding */
    data = (char **) malloc(sizeof(char *) * k);
    coding = (char **) malloc(sizeof(char *) * m);
    for (i = 0; i < m; i++) {
        coding[i] = (char *) malloc(sizeof(char) * blocksize);
        if (coding[i] == NULL) {
            perror("malloc");
            exit(1);
        }
    }



    /* Create coding matrix or bitmatrix and schedule */
    timing_set(&t3);
    switch (tech) {
    case No_Coding:
        break;
    case Reed_Sol_Van:
        matrix = reed_sol_vandermonde_coding_matrix(k, m, w);
        break;
    case Reed_Sol_R6_Op:
        break;
    case Cauchy_Orig:
        matrix = cauchy_original_coding_matrix(k, m, w);
        bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);
        schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
        break;
    case Cauchy_Good:
        matrix = cauchy_good_general_coding_matrix(k, m, w);
        bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);
        schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
        break;
    case Liberation:
        bitmatrix = liberation_coding_bitmatrix(k, w);
        schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
        break;
    case Blaum_Roth:
        bitmatrix = blaum_roth_coding_bitmatrix(k, w);
        schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
        break;
    case Liber8tion:
        bitmatrix = liber8tion_coding_bitmatrix(k);
        schedule = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
        break;
    case RDP:
    case EVENODD:
        assert(0);
    }
    timing_set(&start);
    timing_set(&t4);
    totalsec += timing_delta(&t3, &t4);
```

Examples/encoder.c lines 481 to 540

```
    /* Read in data until finished */
    n = 1;
    total = 0;

    while (n <= readins) {
        /*
         * Check if padding is needed, if so, add appropriate number of zeros
         */
        if (total < size && total + buffersize <= size) {
            total += jfread(block, sizeof(char), buffersize, fp);
        } else if (total < size && total + buffersize > size) {
            extra = jfread(block, sizeof(char), buffersize, fp);
            for (i = extra; i < buffersize; i++) {
                block[i] = '0';
            }
        } else if (total == size) {
            for (i = 0; i < buffersize; i++) {
                block[i] = '0';
            }
        }
        /* Set pointers to point to file data */
        for (i = 0; i < k; i++) {
            data[i] = block + (i * blocksize);
        }

        timing_set(&t3);
        /* Encode according to coding method */
        switch (tech) {
        case No_Coding:
            break;
        case Reed_Sol_Van:
            jerasure_matrix_encode(k, m, w, matrix, data, coding, blocksize);
            break;
        case Reed_Sol_R6_Op:
            reed_sol_r6_encode(k, w, data, coding, blocksize);
            break;
        case Cauchy_Orig:
            jerasure_schedule_encode(k, m, w, schedule, data, coding, blocksize, packetsize);
            break;
        case Cauchy_Good:
            jerasure_schedule_encode(k, m, w, schedule, data, coding, blocksize, packetsize);
            break;
        case Liberation:
            jerasure_schedule_encode(k, m, w, schedule, data, coding, blocksize, packetsize);
            break;
        case Blaum_Roth:
            jerasure_schedule_encode(k, m, w, schedule, data, coding, blocksize, packetsize);
            break;
        case Liber8tion:
            jerasure_schedule_encode(k, m, w, schedule, data, coding, blocksize, packetsize);
            break;
        case RDP:
        case EVENODD:
            assert(0);
        }
        timing_set(&t4);

        /* Write data and encoded data to k+m files */
        for (i = 1; i <= k; i++) {
            if (fp == NULL) {
```

Examples/encoder.c lines 541 to 600

```
                bzero(data[i - 1], blocksize);
            } else {
                sprintf(fname, "%s/Coding/%s_k%0*d%s", curdir, s1, md, i, extension);
                if (n == 1) {
                    fp2 = fopen(fname, "wb");
                } else {
                    fp2 = fopen(fname, "ab");
                }
                fwrite(data[i - 1], sizeof(char), blocksize, fp2);
                fclose(fp2);
            }

        }
        for (i = 1; i <= m; i++) {
            if (fp == NULL) {
                bzero(data[i - 1], blocksize);
            } else {
                sprintf(fname, "%s/Coding/%s_m%0*d%s", curdir, s1, md, i, extension);
                if (n == 1) {
                    fp2 = fopen(fname, "wb");
                } else {
                    fp2 = fopen(fname, "ab");
                }
                fwrite(coding[i - 1], sizeof(char), blocksize, fp2);
                fclose(fp2);
            }
        }
        n++;
        /* Calculate encoding time */
        totalsec += timing_delta(&t3, &t4);
    }

    /* Create metadata file */
    if (fp != NULL) {
        sprintf(fname, "%s/Coding/%s_meta.txt", curdir, s1);
        fp2 = fopen(fname, "wb");
        fprintf(fp2, "%s\n", argv[1]);
        fprintf(fp2, "%d\n", size);
        fprintf(fp2, "%d %d %d %d %d\n", k, m, w, packetsize, buffersize);
        fprintf(fp2, "%s\n", argv[4]);
        fprintf(fp2, "%d\n", tech);
        fprintf(fp2, "%d\n", readins);
        fclose(fp2);
    }
    /* Free allocated memory */
    free(s1);
    free(fname);
    free(block);
    free(curdir);

    /* Calculate rate in MB/sec and print */
    timing_set(&t2);
    tsec = timing_delta(&t1, &t2);
    printf("Encoding (MB/sec): %0.10f\n", (((double) size) / 1024.0 / 1024.0) / totalsec);
    printf("En_Total (MB/sec): %0.10f\n", (((double) size) / 1024.0 / 1024.0) / tsec);

    return 0;
}

/* is_prime returns 1 if number if prime, 0 if not prime */
```

Examples/encoder.c lines 601 to 632

```
int
is_prime(int w)
{
    int             prime55[] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
        73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151, 157, 163, 167, 173, 179,
    181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241, 251, 257};
    int             i;
    for (i = 0; i < 55; i++) {
        if (w % prime55[i] == 0) {
            if (w == prime55[i])
                return 1;
            else {
                return 0;
            }
        }
    }
    assert(0);
}

/* Handles ctrl-\ event */
void
ctrl_bs_handler(int dummy)
{
    time_t          mytime;
    mytime = time(0);
    fprintf(stderr, "\n%s\n", ctime(&mytime));
    fprintf(stderr, "You just typed ctrl-\\ in encoder.c.\n");
    fprintf(stderr, "Total number of read ins = %d\n", readins);
    fprintf(stderr, "Current read in: %d\n", n);
    fprintf(stderr, "Method: %s\n\n", Methods[method]);
    signal(SIGQUIT, ctrl_bs_handler);
}
```

Examples/jerasure_01.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_01 r c w - creates and prints out a matrix in GF(2^w).\n\n");
  fprintf(stderr, "       Element i,j is equal to 2^(i*c+j)\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates jerasure_print_matrix().\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
```

Examples/jerasure_01.c lines 61 to 92

```
}

int main(int argc, char **argv)
{
  int r, c, w, i, n;
  int *matrix;

  if (argc != 4) usage(NULL);
  if (sscanf(argv[1], "%d", &r) == 0 || r <= 0) usage("Bad r");
  if (sscanf(argv[2], "%d", &c) == 0 || c <= 0) usage("Bad c");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0) usage("Bad w");

  matrix = talloc(int, r*c);

  n = 1;
  for (i = 0; i < r*c; i++) {
    matrix[i] = n;
    n = galois_single_multiply(n, 2, w);
  }

  printf("<HTML><TITLE>jerasure_01");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_01");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<pre>\n");

  jerasure_print_matrix(matrix, r, c, w);
  return 0;
}
```

Examples/jerasure_02.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_02 r c w - Converts the matrix of jerasure_01 to a bit matrix.\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates jerasure_print_bitmatrix() and jerasure_matrix_to_bitmatrix().\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}
```

Examples/jerasure_02.c lines 61 to 94

```
int main(int argc, char **argv)
{
  int r, c, w, i, n;
  int *matrix;
  int *bitmatrix;

  if (argc != 4) usage(NULL);
  if (sscanf(argv[1], "%d", &r) == 0 || r <= 0) usage("Bad r");
  if (sscanf(argv[2], "%d", &c) == 0 || c <= 0) usage("Bad c");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0) usage("Bad w");

  matrix = talloc(int, r*c);

  n = 1;
  for (i = 0; i < r*c; i++) {
    matrix[i] = n;
    n = galois_single_multiply(n, 2, w);
  }

  bitmatrix = jerasure_matrix_to_bitmatrix(c, r, w, matrix);

  printf("<HTML><TITLE>jerasure_02");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_02");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<pre>\n");

  jerasure_print_bitmatrix(bitmatrix, r*w, c*w, w);
  return 0;
}
```

Examples/jerasure_03.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_03 k w - Creates a kxk Cauchy matrix in GF(2^w). \n\n");
  fprintf(stderr, "       k must be < 2^w.  Element i,j is 1/(i+(2^w-j-1)).  (If that is\n");
  fprintf(stderr, "       If that is 1/0, then it sets it to zero).  \n");
  fprintf(stderr, "       It then tests whether that matrix is invertible.\n");
  fprintf(stderr, "       If it is invertible, then it prints out the inverse.\n");
```

Examples/jerasure_03.c lines 61 to 119

```
  fprintf(stderr, "       Finally, it prints the product of the matrix and its inverse.\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: jerasure_print_matrix()\n");
  fprintf(stderr, "                   jerasure_invertible_matrix()\n");
  fprintf(stderr, "                   jerasure_invert_matrix()\n");
  fprintf(stderr, "                   jerasure_matrix_multiply().\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

int main(int argc, char **argv)
{
  unsigned int k, w, i, j, n;
  int *matrix;
  int *matrix_copy;
  int *inverse;
  int *identity;

  if (argc != 3) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &w) == 0 || w <= 0 || w > 31) usage("Bad w");
  if (k >= (1 << w)) usage("K too big");

  matrix = talloc(int, k*k);
  matrix_copy = talloc(int, k*k);
  inverse = talloc(int, k*k);

  for (i = 0; i < k; i++) {
    for (j = 0; j < k; j++) {
      n = i ^ ((1 << w)-1-j);
      matrix[i*k+j] = (n == 0) ? 0 : galois_single_divide(1, n, w);
    }
  }

  printf("<HTML><TITLE>jerasure_03");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_03");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<pre>\n");

  printf("The Cauchy Matrix:\n");
  jerasure_print_matrix(matrix, k, k, w);
  memcpy(matrix_copy, matrix, sizeof(int)*k*k);
  i = jerasure_invertible_matrix(matrix_copy, k, w);
  printf("\nInvertible: %s\n", (i == 1) ? "Yes" : "No");
  if (i == 1) {
    printf("\nInverse:\n");
    memcpy(matrix_copy, matrix, sizeof(int)*k*k);
    i = jerasure_invert_matrix(matrix_copy, inverse, k, w);
    jerasure_print_matrix(inverse, k, k, w);
    identity = jerasure_matrix_multiply(inverse, matrix, k, k, k, k, w);
    printf("\nInverse times matrix (should be identity):\n");
    jerasure_print_matrix(identity, k, k, w);
  }
  return 0;
}
```

Examples/jerasure_04.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_04 k w - Performs the analogous bit-matrix operations to jerasure_03.\n\n");
  fprintf(stderr, "       It converts the matrix to a kw*kw bit matrix and does the same operations.\n");
  fprintf(stderr, "       k must be < 2^w.\n");
  fprintf(stderr, "This demonstrates: jerasure_print_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_matrix_to_bitmatrix()\n");
```

```
  fprintf(stderr, "                        jerasure_invertible_bitmatrix()\n");
  fprintf(stderr, "                        jerasure_invert_bitmatrix()\n");
  fprintf(stderr, "                        jerasure_matrix_multiply().\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

int main(int argc, char **argv)
{
  unsigned int k, w, i, j, n;
  int *matrix;
  int *bitmatrix;
  int *bitmatrix_copy;
  int *inverse;
  int *identity;

  if (argc != 3) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &w) == 0 || w <= 0 || w > 31) usage("Bad w");
  if (k >= (1 << w)) usage("K too big");

  matrix = talloc(int, k*k);
  bitmatrix_copy = talloc(int, k*w*k*w);
  inverse = talloc(int, k*w*k*w);

  for (i = 0; i < k; i++) {
    for (j = 0; j < k; j++) {
      n = i ^ ((1 << w)-1-j);
      matrix[i*k+j] = (n == 0) ? 0 : galois_single_divide(1, n, w);
    }
  }
  bitmatrix = jerasure_matrix_to_bitmatrix(k, k, w, matrix);

  printf("<HTML><TITLE>jerasure_04");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_04");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<pre>\n");

  printf("The Cauchy Bit-Matrix:\n");
  jerasure_print_bitmatrix(bitmatrix, k*w, k*w, w);
  memcpy(bitmatrix_copy, bitmatrix, sizeof(int)*k*w*k*w);
  i = jerasure_invertible_bitmatrix(bitmatrix_copy, k*w);
  printf("\nInvertible: %s\n", (i == 1) ? "Yes" : "No");
  if (i == 1) {
    printf("\nInverse:\n");
    memcpy(bitmatrix_copy, bitmatrix, sizeof(int)*k*w*k*w);
    i = jerasure_invert_bitmatrix(bitmatrix_copy, inverse, k*w);
    jerasure_print_bitmatrix(inverse, k*w, k*w, w);
    identity = jerasure_matrix_multiply(inverse, bitmatrix, k*w, k*w, k*w, k*w, 2);
    printf("\nInverse times matrix (should be identity):\n");
    jerasure_print_bitmatrix(identity, k*w, k*w, w);
  }
  return 0;
}
```

Examples/jerasure_05.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_05 k m w size seed - Does a simple Reed-Solomon coding example in GF(2^w).\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       k+m must be <= 2^w.  w can be 8, 16 or 32.\n");
```

Examples/jerasure_05.c lines 61 to 120

```
  fprintf(stderr, "       It sets up a Cauchy generator matrix and encodes\n");
  fprintf(stderr, "       k devices of size bytes with it.  Then it decodes.\n");
  fprintf(stderr, "       After that, it decodes device 0 by using jerasure_make_decoding_matrix()\n");
  fprintf(stderr, "       and jerasure_matrix_dotprod().\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: jerasure_matrix_encode()\n");
  fprintf(stderr, "                   jerasure_matrix_decode()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   jerasure_make_decoding_matrix()\n");
  fprintf(stderr, "                   jerasure_matrix_dotprod()\n");
  if (s != NULL) fprintf(stderr, "\n%s\n\n", s);
  exit(1);
}

static void
print_data_and_coding(int k, int m, int w, int size,
                      char **data, char **coding)
{
    int               i, j, x;
    int               n, sp;

    if (k > m)
        n = k;
    else
        n = m;
    sp = size * 2 + size / (w / 8) + 8;

    printf("%-*sCoding\n", sp, "Data");
    for (i = 0; i < n; i++) {
        if (i < k) {
            printf("D%-2d:", i);
            for (j = 0; j < size; j += (w / 8)) {
                printf(" ");
                for (x = 0; x < w / 8; x++) {
                    printf("%02x", (unsigned char) data[i][j + x]);
                }
            }
            printf("    ");
        } else
            printf("%*s", sp, "");
        if (i < m) {
            printf("C%-2d:", i);
            for (j = 0; j < size; j += (w / 8)) {
                printf(" ");
                for (x = 0; x < w / 8; x++) {
                    printf("%02x", (unsigned char) coding[i][j + x]);
                }
            }
        }
        printf("\n");
    }
    printf("\n");
}

int
main(int argc, char **argv)
{
    int               k, m, w, size;
    int               i, j;
    int              *matrix;
```

Examples/jerasure_05.c lines 121 to 180

```
    char          **data, **coding;
    int            *erasures, *erased;
    int            *decoding_matrix, *dm_ids;
    uint32_t        seed;

    if (argc != 6)
        usage(NULL);
    if (sscanf(argv[1], "%d", &k) == 0 || k <= 0)
        usage("Bad k");
    if (sscanf(argv[2], "%d", &m) == 0 || m <= 0)
        usage("Bad m");
    if (sscanf(argv[3], "%d", &w) == 0 || (w != 8 && w != 16 && w != 32))
        usage("Bad w");
    if (w < 32 && k + m > (1 << w))
        usage("k + m must be <= 2 ^ w");
    if (sscanf(argv[4], "%d", &size) == 0 || size % sizeof(long) != 0)
        usage("size must be multiple of sizeof(long)");
    if (sscanf(argv[5], "%d", &seed) == 0)
        usage("Bad seed");

    matrix = talloc(int, m * k);
    for (i = 0; i < m; i++) {
        for (j = 0; j < k; j++) {
            matrix[i * k + j] = galois_single_divide(1, i ^ (m + j), w);
        }
    }

    printf("<HTML><TITLE>jerasure_05");
    for (i = 1; i < argc; i++)
        printf(" %s", argv[i]);
    printf("</TITLE>\n");
    printf("<h3>jerasure_05");
    for (i = 1; i < argc; i++)
        printf(" %s", argv[i]);
    printf("</h3>\n");
    printf("<pre>\n");

    printf("The Coding Matrix (the last m rows of the Generator Matrix G^T):\n\n");
    jerasure_print_matrix(matrix, m, k, w);
    printf("\n");

    MOA_Seed(seed);
    data = talloc(char *, k);
    for (i = 0; i < k; i++) {
        data[i] = talloc(char, size);
        MOA_Fill_Random_Region(data[i], size);
    }

    coding = talloc(char *, m);
    for (i = 0; i < m; i++) {
        coding[i] = talloc(char, size);
    }

    jerasure_matrix_encode(k, m, w, matrix, data, coding, size);

    printf("Encoding Complete:\n\n");
    print_data_and_coding(k, m, w, size, data, coding);

    erasures = talloc(int, (m + 1));
    erased = talloc(int, (k + m));
```

Examples/jerasure_05.c lines 181 to 232

```
    for (i = 0; i < m + k; i++)
        erased[i] = 0;
    for (i = 0; i < m;) {
        erasures[i] = (MOA_Random_W(w, 1)) % (k + m);
        if (erased[erasures[i]] == 0) {
            erased[erasures[i]] = 1;

            bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i] - k], size);
            i++;
        }
    }
    erasures[i] = -1;

    printf("Erased %d random devices:\n\n", m);
    print_data_and_coding(k, m, w, size, data, coding);

    i = jerasure_matrix_decode(k, m, w, matrix, 0, erasures, data, coding, size);

    printf("State of the system after decoding:\n\n");
    print_data_and_coding(k, m, w, size, data, coding);

    decoding_matrix = talloc(int, k * k);
    dm_ids = talloc(int, k);

    for (i = 0; i < m; i++)
        erased[i] = 1;
    for (; i < k + m; i++)
        erased[i] = 0;

    jerasure_make_decoding_matrix(k, m, w, matrix, erased, decoding_matrix, dm_ids);

    printf("Suppose we erase the first %d devices.  Here is the decoding matrix:\n\n", m);
    jerasure_print_matrix(decoding_matrix, k, k, w);
    printf("\n");
    printf("And dm_ids:\n\n");
    jerasure_print_matrix(dm_ids, 1, k, w);

    bzero(data[0], size);
    jerasure_matrix_dotprod(k, w, decoding_matrix, dm_ids, 0, data, coding, size);

    printf("\nAfter calling jerasure_matrix_dotprod, we calculate the value of device #0 to be:\n\n");
    printf("D0 :");
    for (i = 0; i < size; i += (w / 8)) {
        printf(" ");
        for (j = 0; j < w / 8; j++) {
            printf("%02x", (unsigned char) data[0][i + j]);
        }
    }
    printf("\n\n");

    return 0;
}
```

Examples/jerasure_06.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_06 k m w packetsize seed\n");
  fprintf(stderr, "Does a simple Cauchy Reed-Solomon coding example in GF(2^w).\n");
  fprintf(stderr, "       \n");
```

```
  fprintf(stderr, "       k+m must be < 2^w.  Packetsize must be a multiple of sizeof(long)\n");
  fprintf(stderr, "       It sets up a Cauchy generator matrix and encodes k devices of w*packetsize bytes.\n");
  fprintf(stderr, "       After that, it decodes device 0 by using jerasure_make_decoding_bitmatrix()\n");
  fprintf(stderr, "       and jerasure_bitmatrix_dotprod().\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: jerasure_bitmatrix_encode()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_decode()\n");
  fprintf(stderr, "                   jerasure_print_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_make_decoding_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_bitmatrix_dotprod()\n");
  if (s != NULL) fprintf(stderr, "\n%s\n\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, j, m, psize, x;
  int *matrix, *bitmatrix;
  char **data, **coding;
  int *erasures, *erased;
  int *decoding_matrix, *dm_ids;
  uint32_t seed;

  if (argc != 6) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (w < 30 && (k+m) > (1 << w)) usage("k + m is too big");
  if (sscanf(argv[4], "%d", &psize) == 0 || psize <= 0) usage("Bad packetsize");
  if(psize%sizeof(long) != 0) usage("Packetsize must be multiple of sizeof(long)");
  if (sscanf(argv[5], "%d", &seed) == 0) usage("Bad seed");

  MOA_Seed(seed);
```

Examples/jerasure_06.c lines 121 to 180

```
  matrix = talloc(int, m*k);
  for (i = 0; i < m; i++) {
    for (j = 0; j < k; j++) {
      matrix[i*k+j] = galois_single_divide(1, i ^ (m + j), w);
    }
  }
  bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);

  printf("<HTML><TITLE>jerasure_06");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_06");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");

  printf("<hr>\n");
  printf("Last (m * w) rows of the Generator Matrix: (G^T):\n<pre>\n");
  jerasure_print_bitmatrix(bitmatrix, w*m, w*k, w);
  printf("</pre><hr>\n");

  data = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, psize*w);
    MOA_Fill_Random_Region(data[i], psize*w);
  }

  coding = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, psize*w);
  }

  jerasure_bitmatrix_encode(k, m, w, bitmatrix, data, coding, w*psize, psize);

  printf("Encoding Complete - Here is the state of the system\n\n");
  printf("<p>\n");
  print_array(data, k, psize*w, psize, "D");
  printf("<p>\n");
  print_array(coding, m, psize*w, psize, "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(w, 1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], psize*w);
      i++;
    }
  }
  erasures[i] = -1;

  printf("Erased %d random devices:", m);
  for (i = 0; erasures[i] != -1; i++) {
    printf(" %c%x", ((erasures[i] < k) ? 'D' : 'C'), (erasures[i] < k ? erasures[i] : erasures[i]-k));
  }
  printf(". Here is the state of the system:\n");

  printf("<p>\n");
```

```
  print_array(data, k, psize*w, psize, "D");
  printf("<p>\n");
  print_array(coding, m, psize*w, psize, "C");
  printf("<hr>\n");

  i = jerasure_bitmatrix_decode(k, m, w, bitmatrix, 0, erasures, data, coding,
            w*psize, psize);

  printf("Here is the state of the system after decoding:\n\n");
  printf("<p>\n");
  print_array(data, k, psize*w, psize, "D");
  printf("<p>\n");
  print_array(coding, m, psize*w, psize, "C");
  printf("<hr>\n");

  decoding_matrix = talloc(int, k*k*w*w);
  dm_ids = talloc(int, k);

  x = (m < k) ? m : k;

  for (i = 0; i < x; i++) erased[i] = 1;
  for (; i < k+m; i++) erased[i] = 0;

  jerasure_make_decoding_bitmatrix(k, m, w, bitmatrix, erased, decoding_matrix, dm_ids);

  printf("Suppose we erase the first %d devices.  Here is the decoding matrix:\n<pre>\n", x);
  jerasure_print_bitmatrix(decoding_matrix, k*w, k*w, w);
  printf("</pre>\n");
  printf("And dm_ids:\n<pre>\n");
  jerasure_print_matrix(dm_ids, 1, k, w);
  printf("</pre><hr>\n");

  for (i = 0; i < x; i++) bzero(data[i], w*psize);

  printf("Here is the state of the system after the erasures:\n\n");
  printf("<p>\n");
  print_array(data, k, psize*w, psize, "D");
  printf("<p>\n");
  print_array(coding, m, psize*w, psize, "C");
  printf("<hr>\n");

  for (i = 0; i < x; i++) {
    jerasure_bitmatrix_dotprod(k, w, decoding_matrix+i*(k*w*w), dm_ids, i, data, coding, w*psize, psize);
  }

  printf("Here is the state of the system after calling <b>jerasure_bitmatrix_dotprod()</b>");
  printf(" %d time%s with the decoding matrix:\n\n", x, (x == 1) ? "" : "s");
  printf("<p>\n");
  print_array(data, k, psize*w, psize, "D");
  printf("<p>\n");
  print_array(coding, m, psize*w, psize, "C");
  printf("<hr>\n");
  return 0;
}
```

Examples/jerasure_07.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdint.h>
#include <gf_rand.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_07 k m w seed - Scheduled Cauchy Reed-Solomon coding example in GF(2^w).\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       k+m must be <= 2^w.  It sets up a Cauchy generator matrix and encodes\n");
```

```
  fprintf(stderr, "       k sets of w*%ld bytes. It uses bit-matrix scheduling, both smart and dumb.\n", sizeof(long));
  fprintf(stderr, "       It decodes using bit-matrix scheduling, then shows an example of\n");
  fprintf(stderr, "       using jerasure_do_scheduled_operations().\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: jerasure_dumb_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_smart_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_schedule_encode()\n");
  fprintf(stderr, "                   jerasure_schedule_decode_lazy()\n");
  fprintf(stderr, "                   jerasure_do_scheduled_operations()\n");
  fprintf(stderr, "                   jerasure_get_stats()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, j, m;
  int *matrix, *bitmatrix;
  char **data, **coding, **ptrs;
  int **smart, **dumb;
  int *erasures, *erased;
  double stats[3];
  uint32_t seed;

  if (argc != 5) usage("Wrong number of arguments");
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (sscanf(argv[4], "%d", &seed) == 0) usage("Bad seed");
  if (w < 30 && (k+m) > (1 << w)) usage("k + m is too big");

  matrix = talloc(int, m*k);
  for (i = 0; i < m; i++) {
```

Examples/jerasure_07.c lines 121 to 180

```
    for (j = 0; j < k; j++) {
      matrix[i*k+j] = galois_single_divide(1, i ^ (m + j), w);
    }
  }
  bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);

  printf("<HTML><TITLE>jerasure_07");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_07");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<hr>\n");

  printf("Last m*w rows of the generator matrix (G^T):\n<pre>\n");
  jerasure_print_bitmatrix(bitmatrix, w*m, w*k, w);
  printf("</pre><hr>\n");

  dumb = jerasure_dumb_bitmatrix_to_schedule(k, m, w, bitmatrix);
  smart = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);

  MOA_Seed(seed);
  data = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long)*w);
    MOA_Fill_Random_Region(data[i], sizeof(long)*w);
  }

  coding = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long)*w);
  }

  jerasure_schedule_encode(k, m, w, dumb, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(stats);
  printf("Dumb Encoding Complete: - %.0lf XOR'd bytes.  State of the system:\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_schedule_encode(k, m, w, smart, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(stats);
  printf("Smart Encoding Complete: - %.0lf XOR'd bytes\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(w, 1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
      i++;
```

Examples/jerasure_07.c lines 181 to 223

```
    }
  }
  erasures[i] = -1;

  printf("Erased %d random devices:\n\n", m);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_schedule_decode_lazy(k, m, w, bitmatrix, erasures, data, coding, w*sizeof(long), sizeof(long), 1);
  jerasure_get_stats(stats);

  printf("State of the system after decoding: %.0lf XOR'd bytes\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  ptrs = talloc(char *, (k+m));
  for (i = 0; i < k; i++) ptrs[i] = data[i];
  for (i = 0; i < m; i++) ptrs[k+i] = coding[i];

  for (j = 0; j < m; j++) bzero(coding[j], sizeof(long)*w);
  printf("State of the system after erasing the coding devices:\n");
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_do_scheduled_operations(ptrs, smart, sizeof(long));
  printf("And using <b>jerasure_do_scheduled_operations()</b>: %.0lf XOR'd bytes\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  return 0;
}
```

Examples/jerasure_08.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <stdint.h>
#include <gf_rand.h>
#include "jerasure.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: jerasure_08 k w seed - Example schedule cache usage with RAID-6\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       m=2.  k+m must be <= 2^w.  It sets up a RAID-6 generator matrix and encodes\n");
```

Examples/jerasure_08.c lines 61 to 120

```
  fprintf(stderr, "       k sets of w*%ld bytes. It creates a schedule cache for decoding.\n", sizeof(long));
  fprintf(stderr, "       It demonstrates using the schedule cache for both encoding and decoding.\n");
  fprintf(stderr, "       Then it demonstrates using jerasure_do_parity() to re-encode the first.\n");
  fprintf(stderr, "       coding device\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: jerasure_generate_schedule_cache()\n");
  fprintf(stderr, "                   jerasure_smart_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_schedule_encode()\n");
  fprintf(stderr, "                   jerasure_schedule_decode_cache()\n");
  fprintf(stderr, "                   jerasure_free_schedule()\n");
  fprintf(stderr, "                   jerasure_free_schedule_cache()\n");
  fprintf(stderr, "                   jerasure_get_stats()\n");
  fprintf(stderr, "                   jerasure_do_parity()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, j, m;
  int *matrix, *bitmatrix;
  char **data, **coding;
  int **smart, ***cache;
  int *erasures, *erased;
  double stats[3];
  uint32_t seed;

  if (argc != 4) usage("Wrong number of arguments");
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad m");
  if (sscanf(argv[3], "%d", &seed) == 0) usage("Bad seed");
  m = 2;
  if (w < 30 && (k+m) > (1 << w)) usage("k + m is too big");
```

```
  MOA_Seed(seed);

  matrix = talloc(int, m*k);
  for (j = 0; j < k; j++) matrix[j] = 1;
  i = 1;
  for (j = 0; j < k; j++) {
    matrix[k+j] = i;
    i = galois_single_multiply(i, 2, w);
  }
  bitmatrix = jerasure_matrix_to_bitmatrix(k, m, w, matrix);

  smart = jerasure_smart_bitmatrix_to_schedule(k, m, w, bitmatrix);
  cache = jerasure_generate_schedule_cache(k, m, w, bitmatrix, 1);

  data = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long)*w);
    MOA_Fill_Random_Region(data[i], sizeof(long)*w);
  }

  coding = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long)*w);
  }

  jerasure_schedule_encode(k, m, w, smart, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(stats);

  printf("<HTML><TITLE>jerasure_08");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>jerasure_08");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<hr>\n");

  printf("Encoding Complete: - %.0lf XOR'd bytes.  Here is the state of the system:\n<p>\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erasures[0] = k;
  erasures[1] = k+1;
  erasures[2] = -1;
  for (j = 0; j < m; j++) bzero(coding[j], sizeof(long)*w);

  jerasure_schedule_decode_cache(k, m, w, cache, erasures, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(stats);
  printf("Encoding Using the Schedule Cache: - %.0lf XOR'd bytes\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erased = talloc(int, (k+m));
```

Examples/jerasure_08.c lines 181 to 232

```
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(w, 1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
      i++;
    }
  }
  erasures[i] = -1;

  printf("Erased %d random devices:\n\n", m);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_schedule_decode_cache(k, m, w, cache, erasures, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(stats);

  printf("State of the system after decoding: %.0lf XOR'd bytes\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  bzero(coding[0], sizeof(long)*w);
  printf("Erased the first coding device:\n\n");
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_do_parity(k, data, coding[0], sizeof(long)*w);
  printf("State of the system after using\n");
  printf("<b>jerasure_do_parity()</b> to re-encode it:\n\n");
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  jerasure_free_schedule(smart);
  jerasure_free_schedule_cache(k, m, cache);

  printf("Smart schedule and cache freed.\n\n");

  return 0;
}
```

Examples/liberation_01.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdlib.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "liberation.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: liberation_01 k w seed - Liberation RAID-6 coding/decoding example in GF(2^w).\n");
  fprintf(stderr, "       \n");
```

Examples/liberation_01.c lines 61 to 120

```
  fprintf(stderr, "       w must be prime and k <= w.  It sets up a Liberation bit-matrix\n");
  fprintf(stderr, "       then it encodes k devices of w*%ld bytes using dumb bit-matrix scheduling.\n", sizeof(long));
  fprintf(stderr, "       It decodes using smart bit-matrix scheduling.\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: liberation_coding_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_smart_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_dumb_bitmatrix_to_schedule()\n");
  fprintf(stderr, "                   jerasure_schedule_encode()\n");
  fprintf(stderr, "                   jerasure_schedule_decode_lazy()\n");
  fprintf(stderr, "                   jerasure_print_bitmatrix()\n");
  fprintf(stderr, "                   jerasure_get_stats()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_array(char **ptrs, int ndevices, int size, int packetsize, char *label)
{
  int i, j, x;
  unsigned char *up;

  printf("<center><table border=3 cellpadding=3><tr><td></td>\n");

  for (i = 0; i < ndevices; i++) printf("<td align=center>%s%x</td>\n", label, i);
  printf("</tr>\n");
  printf("<td align=right><pre>");
  for (j = 0; j < size/packetsize; j++) printf("Packet %d\n", j);
  printf("</pre></td>\n");
  for (i = 0; i < ndevices; i++) {
    printf("<td><pre>");
    up = (unsigned char *) ptrs[i];
    for (j = 0; j < size/packetsize; j++) {
      for (x = 0; x < packetsize; x++) {
        if (x > 0 && x%4 == 0) printf(" ");
        printf("%02x", up[j*packetsize+x]);
      }
      printf("\n");
    }
    printf("</td>\n");
  }
  printf("</tr></table></center>\n");
}

int main(int argc, char **argv)
{
  int k, w, i, m;
  int *bitmatrix;
  char **data, **coding;
  int **dumb;
  int *erasures, *erased;
  double stats[3];
  uint32_t seed;

  if (argc != 4) usage("Wrong number of arguments");
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (sscanf(argv[3], "%u", &seed) == 0) usage("Bad seed");
  m = 2;
  if (w < k) usage("k is too big");
  for (i = 2; i*i <= w; i++) if (w%i == 0) usage("w isn't prime");
```

Examples/liberation_01.c lines 121 to 180

```
  bitmatrix = liberation_coding_bitmatrix(k, w);
  if (bitmatrix == NULL) {
    usage("couldn't make coding matrix");
  }

  printf("<HTML><TITLE>liberation_01");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>liberation_01");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<hr>\n");

  printf("Coding Bit-Matrix:\n<pre>\n");
  jerasure_print_bitmatrix(bitmatrix, w*m, w*k, w);
  printf("</pre><hr>\n");

  dumb = jerasure_dumb_bitmatrix_to_schedule(k, m, w, bitmatrix);

  MOA_Seed(seed);
  data = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long)*w);
    MOA_Fill_Random_Region(data[i], sizeof(long)*w);
  }

  coding = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long)*w);
  }

  jerasure_schedule_encode(k, m, w, dumb, data, coding, w*sizeof(long), sizeof(long));
  jerasure_get_stats(stats);
  printf("Smart Encoding Complete: - %.0lf XOR'd bytes.  State of the system:\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(30,1)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], sizeof(long)*w);
      i++;
    }
  }
  erasures[i] = -1;

  printf("Erased %d random devices:\n\n", m);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");
```

Examples/liberation_01.c lines 181 to 192

```
  jerasure_schedule_decode_lazy(k, m, w, bitmatrix, erasures, data, coding, w*sizeof(long), sizeof(long), 1);
  jerasure_get_stats(stats);

  printf("State of the system after decoding: %.0lf XOR'd bytes\n\n", stats[0]);
  printf("<p>\n");
  print_array(data, k, sizeof(long)*w, sizeof(long), "D");
  printf("<p>\n");
  print_array(coding, m, sizeof(long)*w, sizeof(long), "C");
  printf("<hr>\n");

  return 0;
}
```

Examples/reed_sol_01.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "reed_sol.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: reed_sol_01 k m w seed - Does a simple Reed-Solomon coding example in GF(2^w).\n");
  fprintf(stderr, "       \n");
```

```
  fprintf(stderr, "w must be 8, 16 or 32.  k+m must be <= 2^w.  It sets up a classic\n");
  fprintf(stderr, "Vandermonde-based generator matrix and encodes k devices of\n");
  fprintf(stderr, "%ld bytes each with it.  Then it decodes.\n", sizeof(long));
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: jerasure_matrix_encode()\n");
  fprintf(stderr, "                   jerasure_matrix_decode()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   reed_sol_vandermonde_coding_matrix()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

static void print_data_and_coding(int k, int m, int w, int size,
                char **data, char **coding)
{
  int i, j, x;
  int n, sp;

  if(k > m) n = k;
  else n = m;
  sp = size * 2 + size/(w/8) + 8;

  printf("%-*sCoding\n", sp, "Data");
  for(i = 0; i < n; i++) {
          if(i < k) {
                  printf("D%-2d:", i);
                  for(j=0;j< size; j+=(w/8)) {
                          printf(" ");
                          for(x=0;x < w/8;x++){
                                printf("%02x", (unsigned char)data[i][j+x]);
                          }
                  }
                  printf("    ");
          }
          else printf("%*s", sp, "");
          if(i < m) {
                  printf("C%-2d:", i);
                  for(j=0;j< size; j+=(w/8)) {
                          printf(" ");
                          for(x=0;x < w/8;x++){
                                printf("%02x", (unsigned char)coding[i][j+x]);
                          }
                  }
          }
          printf("\n");
  }
        printf("\n");
}

int main(int argc, char **argv)
{
  long l;
  int k, w, i, j, m;
  int *matrix;
  char **data, **coding, **dcopy, **ccopy;
  unsigned char uc;
  int *erasures, *erased;
  uint32_t seed;

  if (argc != 5) usage(NULL);
```

Examples/reed_sol_01.c lines 121 to 180

```
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || (w != 8 && w != 16 && w != 32)) usage("Bad w");
  if (sscanf(argv[4], "%u", &seed) == 0) usage("Bad seed");
  if (w <= 16 && k + m > (1 << w)) usage("k + m is too big");

  matrix = reed_sol_vandermonde_coding_matrix(k, m, w);

  printf("<HTML><TITLE>reed_sol_01 %d %d %d %d</title>\n", k, m, w, seed);
  printf("<h3>reed_sol_01 %d %d %d %d</h3>\n", k, m, w, seed);
  printf("<pre>\n");
  printf("Last m rows of the generator Matrix (G^T):\n\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("\n");

  MOA_Seed(seed);
  data = talloc(char *, k);
  dcopy = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long));
    dcopy[i] = talloc(char, sizeof(long));
    for (j = 0; j < sizeof(long); j++) {
      uc = MOA_Random_W(8, 1);
      data[i][j] = (char) uc;
    }
    memcpy(dcopy[i], data[i], sizeof(long));
  }

  coding = talloc(char *, m);
  ccopy = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long));
    ccopy[i] = talloc(char, sizeof(long));
  }

  jerasure_matrix_encode(k, m, w, matrix, data, coding, sizeof(long));

  for (i = 0; i < m; i++) {
    memcpy(ccopy[i], coding[i], sizeof(long));
  }

  printf("Encoding Complete:\n\n");
  print_data_and_coding(k, m, w, sizeof(long), data, coding);

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  l = 0;
  for (i = 0; i < m; ) {
    erasures[i] = MOA_Random_W(31, 0)%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      memcpy((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], &l, sizeof(long));
      i++;
    }
  }
  erasures[i] = -1;

  printf("Erased %d random devices:\n\n", m);
  print_data_and_coding(k, m, w, sizeof(long), data, coding);
```

Examples/reed_sol_01.c lines 181 to 195

```
  i = jerasure_matrix_decode(k, m, w, matrix, 1, erasures, data, coding, sizeof(long));

  printf("State of the system after decoding:\n\n");
  print_data_and_coding(k, m, w, sizeof(long), data, coding);

  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  return 0;
}
```

Examples/reed_sol_02.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "jerasure.h"
#include "reed_sol.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: reed_sol_02 k m w - Vandermonde matrices in GF(2^w).\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       k+m must be <= 2^w.  This simply prints out the \n");
  fprintf(stderr, "       Vandermonde matrix in GF(2^w), and then the generator\n");
```

Examples/reed_sol_02.c lines 61 to 104

```
  fprintf(stderr, "       matrix that is constructed from it.  See [Plank-Ding-05] for\n");
  fprintf(stderr, "       information on how this construction proceeds\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: reed_sol_extended_vandermonde_matrix()\n");
  fprintf(stderr, "                   reed_sol_big_vandermonde_coding_matrix()\n");
  fprintf(stderr, "                   reed_sol_vandermonde_coding_matrix()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

int main(int argc, char **argv)
{
  int k, w, m;
  int *matrix;

  if (argc != 4) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || w <= 0 || w > 32) usage("Bad w");
  if (w <= 30 && k + m > (1 << w)) usage("k + m is too big");

  matrix = reed_sol_extended_vandermonde_matrix(k+m, k, w);

  printf("<HTML><TITLE>reed_sol_02 %d %d %d</title>\n", k, m, w);
  printf("<h3>reed_sol_02 %d %d %d</h3>\n", k, m, w);
  printf("<pre>\n");
  printf("Extended Vandermonde Matrix:\n\n");
  jerasure_print_matrix(matrix, k+m, k, w);
  printf("\n");

  matrix = reed_sol_big_vandermonde_distribution_matrix(k+m, k, w);
  printf("Vandermonde Generator Matrix (G^T):\n\n");
  jerasure_print_matrix(matrix, k+m, k, w);
  printf("\n");

  matrix = reed_sol_vandermonde_coding_matrix(k, m, w);
  printf("Vandermonde Coding Matrix:\n\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("\n");


  return 0;
}
```

Examples/reed_sol_03.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "reed_sol.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: reed_sol_03 k w seed - Does a simple RAID-6 coding example in GF(2^w).\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       w must be 8, 16 or 32.  k+2 must be <= 2^w.  It sets up a classic\n");
```

```
  fprintf(stderr, "       RAID-6 coding matrix based on Anvin's optimization and encodes\n");
  fprintf(stderr, "       %ld-byte devices with it.  Then it decodes.\n", sizeof(long));
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: reed_sol_r6_encode()\n");
  fprintf(stderr, "                   reed_sol_r6_coding_matrix()\n");
  fprintf(stderr, "                   jerasure_matrix_decode()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}


static void print_data_and_coding(int k, int m, int w, int size,
                char **data, char **coding)
{
  int i, j, x;
  int n, sp;

  if(k > m) n = k;
  else n = m;
  sp = size * 2 + size/(w/8) + 8;

  printf("%-*sCoding\n", sp, "Data");
  for(i = 0; i < n; i++) {
        if(i < k) {
                printf("D%-2d:", i);
                for(j=0;j< size; j+=(w/8)) {
                        printf(" ");
                        for(x=0;x < w/8;x++){
                              printf("%02x", (unsigned char)data[i][j+x]);
                        }
                }
                printf("    ");
        }
        else printf("%*s", sp, "");
        if(i < m) {
                printf("C%-2d:", i);
                for(j=0;j< size; j+=(w/8)) {
                        printf(" ");
                        for(x=0;x < w/8;x++){
                              printf("%02x", (unsigned char)coding[i][j+x]);
                        }
                }
        }
        printf("\n");
  }
      printf("\n");
}

int main(int argc, char **argv)
{
  long l;
  unsigned char uc;
  int k, w, i, j, m;
  int *matrix;
  char **data, **coding, **dcopy, **ccopy;
  int *erasures, *erased;
  uint32_t seed;

  if (argc != 4) usage(NULL);
```

Examples/reed_sol_03.c lines 121 to 180

```
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &w) == 0 || (w != 8 && w != 16 && w != 32)) usage("Bad w");
  if (sscanf(argv[3], "%d", &seed) == 0) usage("Bad seed");
  m = 2;
  if (w <= 16 && k + m > (1 << w)) usage("k + m is too big");

  MOA_Seed(seed);
  matrix = reed_sol_r6_coding_matrix(k, w);

  printf("<HTML><TITLE>reed_sol_03 %d %d %d</title>\n", k, w, seed);
  printf("<h3>reed_sol_03 %d %d %d</h3>\n", k, w, seed);
  printf("<pre>\n");

  printf("Last 2 rows of the Generator Matrix:\n\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("\n");

  data = talloc(char *, k);
  dcopy = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, sizeof(long));
    dcopy[i] = talloc(char, sizeof(long));
    for (j = 0; j < sizeof(long); j++) {
      uc = MOA_Random_W(8, 1) %256;
      data[i][j] = (char) uc;
    }
    memcpy(dcopy[i], data[i], sizeof(long));
  }

  coding = talloc(char *, m);
  ccopy = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, sizeof(long));
    ccopy[i] = talloc(char, sizeof(long));
  }

  reed_sol_r6_encode(k, w, data, coding, sizeof(long));
  for (i = 0; i < m; i++) {
    memcpy(ccopy[i], coding[i], sizeof(long));
  }

  printf("Encoding Complete:\n\n");
  print_data_and_coding(k, m, w, sizeof(long), data, coding);

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  l = 0;
  for (i = 0; i < m; ) {
    erasures[i] = ((unsigned int) MOA_Random_W(w, 1))%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      memcpy((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], &l, sizeof(long));
      i++;
    }
  }
  erasures[i] = -1;

  printf("Erased %d random devices:\n\n", m);
  print_data_and_coding(k, m, w, sizeof(long), data, coding);
```

Examples/reed_sol_03.c lines 181 to 195

```
  i = jerasure_matrix_decode(k, m, w, matrix, 1, erasures, data, coding, sizeof(long));

  printf("State of the system after decoding:\n\n");
  print_data_and_coding(k, m, w, sizeof(long), data, coding);

  for (i = 0; i < k; i++) if (memcmp(data[i], dcopy[i], sizeof(long)) != 0) {
    printf("ERROR: D%x after decoding does not match its state before decoding!<br>\n", i);
  }
  for (i = 0; i < m; i++) if (memcmp(coding[i], ccopy[i], sizeof(long)) != 0) {
    printf("ERROR: C%x after decoding does not match its state before decoding!<br>\n", i);
  }

  return 0;
}
```

Examples/reed_sol_04.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <gf_rand.h>
#include "jerasure.h"
#include "reed_sol.h"

#define talloc(type, num) (type *) malloc(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: reed_sol_04 w seed - Shows reed_sol_galois_wXX_region_multby_2\n");
  fprintf(stderr, "       \n");
```

Examples/reed_sol_04.c lines 61 to 120

```
  fprintf(stderr, "       w must be 8, 16 or 32.  Sets up an array of 4 random words in\n");
  fprintf(stderr, "       GF(2^w) and multiplies them by two.  \n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This demonstrates: reed_sol_galois_w08_region_multby_2()\n");
  fprintf(stderr, "                   reed_sol_galois_w16_region_multby_2()\n");
  fprintf(stderr, "                   reed_sol_galois_w32_region_multby_2()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

int main(int argc, char **argv)
{
  unsigned char *x, *y;
  unsigned short *xs, *ys;
  unsigned int *xi, *yi;
  uint32_t seed;
  int *a32, *copy;
  int i;
  int w;

  if (argc != 3) usage(NULL);
  if (sscanf(argv[1], "%d", &w) == 0 || (w != 8 && w != 16 && w != 32)) usage("Bad w");
  if (sscanf(argv[2], "%d", &seed) == 0) usage("Bad seed");

  printf("<HTML><TITLE>reed_sol_04 %d %d</title>\n", w, seed);
  printf("<h3>reed_sol_04 %d %d</h3>\n", w, seed);
  printf("<pre>\n");

  MOA_Seed(seed);
  a32 = talloc(int, 4);
  copy = talloc(int, 4);
  y = (unsigned char *) a32;
  for (i = 0; i < 4*sizeof(int); i++) y[i] = MOA_Random_W(8, 1);
  memcpy(copy, a32, sizeof(int)*4);

  if (w == 8) {
    x = (unsigned char *) copy;
    y = (unsigned char *) a32;
    reed_sol_galois_w08_region_multby_2((char *) a32, sizeof(int)*4);
    for (i = 0; i < 4*sizeof(int)/sizeof(char); i++) {
       printf("Char %2d: %3u *2 = %3u\n", i, x[i], y[i]);
    }
  } else if (w == 16) {
    xs = (unsigned short *) copy;
    ys = (unsigned short *) a32;
    reed_sol_galois_w16_region_multby_2((char *) a32, sizeof(int)*4);
    for (i = 0; i < 4*sizeof(int)/sizeof(short); i++) {
       printf("Short %2d: %5u *2 = %5u\n", i, xs[i], ys[i]);
    }
  } else if (w == 32) {
    xi = (unsigned int *) copy;
    yi = (unsigned int *) a32;
    reed_sol_galois_w16_region_multby_2((char *) a32, sizeof(int)*4);
    for (i = 0; i < 4*sizeof(int)/sizeof(int); i++) {
       printf("Int %2d: %10u *2 = %10u\n", i, xi[i], yi[i]);
    }
  }

  return 0;
}
```

Examples/reed_sol_test_gf.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gf_complete.h>
#include <gf_method.h>
#include <gf_rand.h>
#include <stdint.h>
#include <sys/time.h>
#include "jerasure.h"
#include "reed_sol.h"

#define BUFSIZE 4096

static void *malloc16(int size) {
```

Examples/reed_sol_test_gf.c lines 61 to 120

```
    void *mem = malloc(size+16+sizeof(void*));
    void **ptr = (void**)((long)(mem+16+sizeof(void*)) & ~(15));
    ptr[-1] = mem;
    return ptr;
}

#if 0
// Unused for now.
static void free16(void *ptr) {
    free(((void**)ptr)[-1]);
}
#endif

#define talloc(type, num) (type *) malloc16(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: reed_sol_test_gf k m w seed (additional GF args) - Tests Reed-Solomon in GF(2^w).\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       w must be 8, 16 or 32.  k+m must be <= 2^w.\n");
  fprintf(stderr, "       See the README for information on the additional GF args.\n");
  fprintf(stderr, "       Set up a Vandermonde-based distribution matrix and encodes k devices of\n");
  fprintf(stderr, "       %d bytes each with it.  Then it decodes.\n", BUFSIZE);
  fprintf(stderr, "       \n");
  fprintf(stderr, "This tests:        jerasure_matrix_encode()\n");
  fprintf(stderr, "                   jerasure_matrix_decode()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   galois_change_technique()\n");
  fprintf(stderr, "                   reed_sol_vandermonde_coding_matrix()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

gf_t* get_gf(int w, int argc, char **argv, int starting)
{
  gf_t *gf = (gf_t*)malloc(sizeof(gf_t));
  if (create_gf_from_argv(gf, w, argc, argv, starting) == 0) {
    free(gf);
    gf = NULL;
  }
  return gf;
}

int main(int argc, char **argv)
{
  int k, w, i, m;
  int *matrix;
  char **data, **coding, **old_values;
  int *erasures, *erased;
  gf_t *gf = NULL;
  uint32_t seed;

  if (argc < 6) usage("Not enough command line arguments");
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || (w != 8 && w != 16 && w != 32)) usage("Bad w");
  if (sscanf(argv[4], "%d", &seed) == 0) usage("Bad seed");
  if (w <= 16 && k + m > (1 << w)) usage("k + m is too big");

  MOA_Seed(seed);
```

Examples/reed_sol_test_gf.c lines 121 to 180

```
  gf = get_gf(w, argc, argv, 5);

  if (gf == NULL) {
    usage("Invalid arguments given for GF!\n");
  }

  galois_change_technique(gf, w);

  matrix = reed_sol_vandermonde_coding_matrix(k, m, w);

  printf("<HTML><TITLE>reed_sol_test_gf");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>reed_sol_test_gf");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<pre>\n");

  printf("Last m rows of the generator matrix (G^T):\n\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("\n");

  data = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, BUFSIZE);
    MOA_Fill_Random_Region(data[i], BUFSIZE);
  }

  coding = talloc(char *, m);
  old_values = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, BUFSIZE);
    old_values[i] = talloc(char, BUFSIZE);
  }

  jerasure_matrix_encode(k, m, w, matrix, data, coding, BUFSIZE);

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = ((unsigned int)MOA_Random_W(w,1))%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      memcpy(old_values[i], (erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], BUFSIZE);
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], BUFSIZE);
      i++;
    }
  }
  erasures[i] = -1;

  i = jerasure_matrix_decode(k, m, w, matrix, 1, erasures, data, coding, BUFSIZE);

  for (i = 0; i < m; i++) {
    if (erasures[i] < k) {
      if (memcmp(data[erasures[i]], old_values[i], BUFSIZE)) {
        fprintf(stderr, "Decoding failed for %d!\n", erasures[i]);
        exit(1);
      }
```

Examples/reed_sol_test_gf.c lines 181 to 191

```
    } else {
      if (memcmp(coding[erasures[i]-k], old_values[i], BUFSIZE)) {
        fprintf(stderr, "Decoding failed for %d!\n", erasures[i]);
        exit(1);
      }
    }
  }

  printf("Encoding and decoding were both successful.\n");
  return 0;
}
```

Examples/reed_sol_time_gf.c lines 1 to 60

```
/* *
 * Copyright (c) 2014, James S. Plank and Kevin Greenan
 * All rights reserved.
 *
 * Jerasure - A C/C++ Library for a Variety of Reed-Solomon and RAID-6 Erasure
 * Coding Techniques
 *
 * Revision 2.0: Galois Field backend now links to GF-Complete
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 *  - Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 *  - Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 *  - Neither the name of the University of Tennessee nor the names of its
 *    contributors may be used to endorse or promote products derived
 *    from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
 * "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
 * LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
 * A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
 * HOLDER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
 * INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING,
 * BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS
 * OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED
 * AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY
 * WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

/* Jerasure's authors:

   Revision 2.x - 2014: James S. Plank and Kevin M. Greenan.
   Revision 1.2 - 2008: James S. Plank, Scott Simmerman and Catherine D. Schuman.
   Revision 1.0 - 2007: James S. Plank.
 */

#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <gf_complete.h>
#include <gf_rand.h>
#include <gf_method.h>
#include <stdint.h>
#include "jerasure.h"
#include "reed_sol.h"
#include "timing.h"

static void *malloc16(int size) {
    void *mem = malloc(size+16+sizeof(void*));
```

Examples/reed_sol_time_gf.c lines 61 to 120

```
    void **ptr = (void**)((long)(mem+16+sizeof(void*)) & ~(15));
    ptr[-1] = mem;
    return ptr;
}

#if 0
// Unused for now.
static void free16(void *ptr) {
    free(((void**)ptr)[-1]);
}
#endif

#define talloc(type, num) (type *) malloc16(sizeof(type)*(num))

static void usage(char *s)
{
  fprintf(stderr, "usage: reed_sol_time_gf k m w seed iterations bufsize (additional GF args) ");
  fprintf(stderr, "- Test and time Reed-Solomon in a particular GF(2^w).\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "       w must be 8, 16 or 32.  k+m must be <= 2^w.\n");
  fprintf(stderr, "       See the README for information on the additional GF args.\n");
  fprintf(stderr, "       Set up a Vandermonde-based distribution matrix and encodes k devices of\n");
  fprintf(stderr, "       bufsize bytes each with it.  Then it decodes.\n");
  fprintf(stderr, "       \n");
  fprintf(stderr, "This tests:        jerasure_matrix_encode()\n");
  fprintf(stderr, "                   jerasure_matrix_decode()\n");
  fprintf(stderr, "                   jerasure_print_matrix()\n");
  fprintf(stderr, "                   galois_change_technique()\n");
  fprintf(stderr, "                   reed_sol_vandermonde_coding_matrix()\n");
  if (s != NULL) fprintf(stderr, "%s\n", s);
  exit(1);
}

gf_t* get_gf(int w, int argc, char **argv, int starting)
{
  gf_t *gf = (gf_t*)malloc(sizeof(gf_t));
  if (create_gf_from_argv(gf, w, argc, argv, starting) == 0) {
    free(gf);
    gf = NULL;
  }
  return gf;
}

int main(int argc, char **argv)
{
  int k, w, i, m, iterations, bufsize;
  int *matrix;
  char **data, **coding, **old_values;
  int *erasures, *erased;
  uint32_t seed;
  double t = 0, total_time = 0;
  gf_t *gf = NULL;

  if (argc < 8) usage(NULL);
  if (sscanf(argv[1], "%d", &k) == 0 || k <= 0) usage("Bad k");
  if (sscanf(argv[2], "%d", &m) == 0 || m <= 0) usage("Bad m");
  if (sscanf(argv[3], "%d", &w) == 0 || (w != 8 && w != 16 && w != 32)) usage("Bad w");
  if (sscanf(argv[4], "%d", &seed) == 0) usage("Bad seed");
  if (sscanf(argv[5], "%d", &iterations) == 0) usage("Bad iterations");
  if (sscanf(argv[6], "%d", &bufsize) == 0) usage("Bad bufsize");
```

Examples/reed_sol_time_gf.c lines 121 to 180

```
  if (w <= 16 && k + m > (1 << w)) usage("k + m is too big");

  MOA_Seed(seed);

  gf = get_gf(w, argc, argv, 7);

  if (gf == NULL) {
    usage("Invalid arguments given for GF!\n");
  }

  galois_change_technique(gf, w);

  matrix = reed_sol_vandermonde_coding_matrix(k, m, w);

  printf("<HTML><TITLE>reed_sol_time_gf");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</TITLE>\n");
  printf("<h3>reed_sol_time_gf");
  for (i = 1; i < argc; i++) printf(" %s", argv[i]);
  printf("</h3>\n");
  printf("<pre>\n");

  printf("Last m rows of the generator matrix (G^T):\n\n");
  jerasure_print_matrix(matrix, m, k, w);
  printf("\n");

  data = talloc(char *, k);
  for (i = 0; i < k; i++) {
    data[i] = talloc(char, bufsize);
    MOA_Fill_Random_Region(data[i], bufsize);
  }

  coding = talloc(char *, m);
  old_values = talloc(char *, m);
  for (i = 0; i < m; i++) {
    coding[i] = talloc(char, bufsize);
    old_values[i] = talloc(char, bufsize);
  }

  for (i = 0; i < iterations; i++) {
    t = timing_now();
    jerasure_matrix_encode(k, m, w, matrix, data, coding, bufsize);
    total_time += timing_now() - t;
  }

  printf("Encode throughput for %d iterations: %.2f MB/s (%.2f sec)\n",
      iterations, (double)(k*iterations*bufsize/1024/1024) / total_time, total_time);

  erasures = talloc(int, (m+1));
  erased = talloc(int, (k+m));
  for (i = 0; i < m+k; i++) erased[i] = 0;
  for (i = 0; i < m; ) {
    erasures[i] = ((unsigned int)MOA_Random_W(w, 1))%(k+m);
    if (erased[erasures[i]] == 0) {
      erased[erasures[i]] = 1;
      memcpy(old_values[i], (erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], bufsize);
      bzero((erasures[i] < k) ? data[erasures[i]] : coding[erasures[i]-k], bufsize);
      i++;
    }
  }
```

Examples/reed_sol_time_gf.c lines 181 to 207

```
  erasures[i] = -1;

  for (i = 0; i < iterations; i++) {
    t = timing_now();
    jerasure_matrix_decode(k, m, w, matrix, 1, erasures, data, coding, bufsize);
    total_time += timing_now() - t;
  }

  printf("Decode throughput for %d iterations: %.2f MB/s (%.2f sec)\n",
     iterations, (double)(k*iterations*bufsize/1024/1024) / total_time, total_time);

  for (i = 0; i < m; i++) {
    if (erasures[i] < k) {
      if (memcmp(data[erasures[i]], old_values[i], bufsize)) {
        fprintf(stderr, "Decoding failed for %d!\n", erasures[i]);
        exit(1);
      }
    } else {
      if (memcmp(coding[erasures[i]-k], old_values[i], bufsize)) {
        fprintf(stderr, "Decoding failed for %d!\n", erasures[i]);
        exit(1);
      }
    }
  }

  return 0;
}
```

Examples/test_galois.c lines 1 to 23

```
#include <assert.h>
#include "galois.h"

int main(int argc, char **argv)
{
  assert(galois_init_default_field(4) == 0);
  assert(galois_uninit_field(4) == 0);
  assert(galois_init_default_field(4) == 0);
  assert(galois_uninit_field(4) == 0);

  assert(galois_init_default_field(8) == 0);
  assert(galois_uninit_field(8) == 0);
  assert(galois_init_default_field(8) == 0);
  assert(galois_uninit_field(8) == 0);

  return 0;
}
/*
 * Local Variables:
 * compile-command: "make test_galois &&
 *     libtool --mode=execute valgrind --tool=memcheck --leak-check=full ./test_galois"
 * End:
 */
```