A B-TREE ASSIGNMENT THAT IS REALISTIC ENOUGH THAT STUDENTS CAN LIST IT ON THEIR RESUMES

James S. Plank
EECS Department
University of Tennessee
Knoxville, TN 37996
865-974-4397
plank@cs.utk.edu

Citation information for this paper:

The Journal of Computing Sciences in Colleges.
Volume 31, #2, December, 2015, pages 278-282.

MOTIVATION

High level languages like Python and Java, and even C++, abstract away so many details of what goes on "beneath the hood'" when a computer program is running, that it becomes a challenge to open students' eyes to the fascinating interplay between algorithms and computer systems. In particular, the C++ Standard Template Library does such a good job with vectors, sets and maps, that it is hard to impart to students the subtle impact of issues such as memory management, that can profoundly affect the performance of a program. Most Computer Science curricula focus on the theory and abstract mechanics of algorithms, and then move onto the design principles of operating systems. In my experience, students do not get enough hands-on experience with the bits and bytes of their computer systems, and how the algorithms that they learn in the abstract interact with these bits and bytes.

One data structure that we teach our sophomores/juniors, almost as an aside, is the B-Tree [1]. This is a balanced tree data structure, which like other balanced trees (for example AVL trees, red-black trees, splay trees) has logarithmic time queries, insertions and deletions. However, the B-Tree is designed to work in an out-of-core environment, where the relevant data is stored on disk, and therefore the most expensive operations are reads and writes of disk sectors, rather than CPU cycles.

From my experience both learning B-Trees as an undergraduate in the 1980's and teaching them for the past 20 years, I believe that when we teach B-Tree to students, they do not intuitively understand how brilliantly B-Trees achieve their goals. The reason is that when we teach them, we limit our examples to toy scenarios applicable to pencil and paper.

OVERVIEW

To address this motivation, I have developed a programming assignment that revolves around B-Trees. In this assignment, I give the students a library that presents them with a simulated raw disk interface. This is implemented on top of

standard Unix files. The interface is realistic, allowing the students only to read and write 1K disk sectors in their entirety.

On top of this interface, the students write a second library that implements B-Trees. The B-Trees are stored on the disks in a specific, and very realistic format:

- The first block of the disk is a ``boot block'' which contains information about the keys that the B-Tree stores, and how many disk blocks the B-Tree currently consumes.
- The B-Trees store keys, which are a fixed number of bytes, and data, which are whole disk sectors.
- The nodes of the B-Trees hold keys, and disk block addresses, which are pointers either to other B-Tree nodes (if the node is an internal node), or to data (if the node is an external node).
- The number of keys that a node may hold is fixed, and it determined by the size of the keys. In the assignment, keys may range from four bytes to 254 bytes.

I provide multiple programs that use the procedures of the students' libraries to build and test a variety of B-Trees. Some of these are designed to help the students debug, and others are designed to test their implementation. Because the B-Trees adhere to a specific format, the trees themselves must be interoperable between the students' implementations, and mine.

EXAMPLE

To help illustrate, I draw an example B-Tree in Figure 1. In this example, B-Tree nodes hold up to four keys each. The root of the tree is held in disk block 8 (identified in the boot block of the disk), and holds one key. Therefore, it is an internal node with two pointers to other nodes, which are in disk blocks 1 and 7 respectively. Those two nodes are external nodes, which also hold up to four keys, and they store pointers to the data. For example, the external node that holds keys 1 through 4 points to data in disk blocks, 4, 11, 9, 2, and 6.

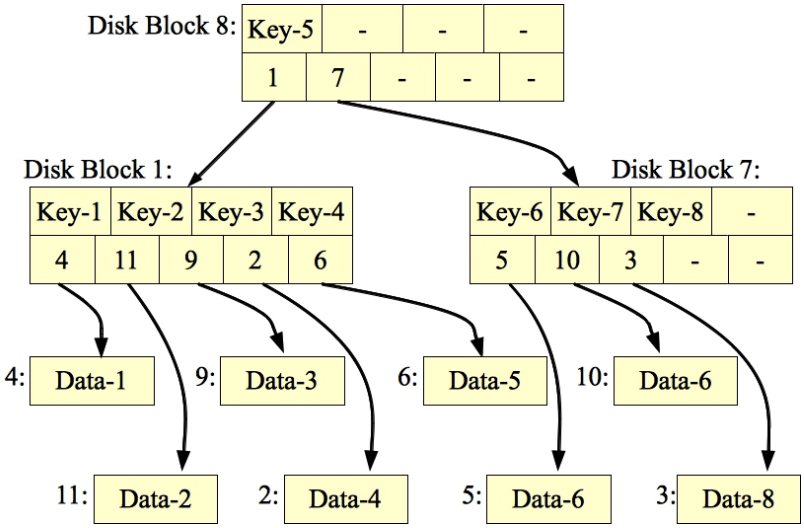

Figure 1: Example B-Tree structure: Internal nodes, external nodes and data each fit into disk sectors, which must be read and written in their entirety.

The B-Tree structure should be apparent from the figure; however all of the components are stored on disk sectors that must be read and written in their entirety. It is the job of the students to implement B-Tree operations that work within this very realistic structure.

ALGORITHMS AND SYSTEMS

A strength of this assignment is that is stresses both algorithms and systems. Consider the steps that the student's library must perform to find a key and read its data:

- It must read the boot block and interpret it to figure out the key size and the location of the root node.
- It must, for every internal node from the root down to the appropriate leaf node, read the node's block from disk, and convert it to an internal format that represents the B-Tree node, and then use that node to find the next appropriate node.
- When it gets to a leaf node, it must read the disk block corresponding to the data.

Of course, the write path is more challenging, as it can involve ``splitting'' both external and internal nodes when they accumulate too many keys.

Therefore, the students need to understand not only the theoretical mechanics of B-Trees, but they must address the challenge of converting 1K blocks of bytes into B-Tree structure, perhaps modify that structure, and then convert it back to 1K blocks.

EXPERIENCE

I developed this assignment for a junior/senior elective course entitled ``Advanced Programming and Algorithms.'' The prerequisite course is our third semester programming course, ``Data Structures / Algorithms II.'' The students programmed this assignment in C, although C++ (with no STL) would work, too. No linguistic data structure support is required, because the main data structure (the B-Tree) is held on disk.

In the spring of 2015, I allotted the students three weeks to do this assignment. Were I teaching it in an Operating Systems or Systems Programming course, I would probably give them two weeks, and perhaps partition it into two week-long assignments to help them budget their time.

The assignment was a challenge to the students. The major stumbling blocks were the conversion from 1024-byte chunks on disk to in-memory data structures and back again, and doing the actual splitting of external and internal nodes. These issues were challenges for our students because they have limited experience with

the bits and bytes of their computers, having cut their teeth on general-purpose data structures like the STL.

The students reported a *great* deal of satisfaction on completing this assignment. Several of them put the assignment as a line-item on their resume --- ``Implemented B-Trees on a Raw Disk Interface'' --- which communicates that they have some of the low-level experience that many of our local employers (for example Cisco, and Cadre5) crave in their hires.

AVAILABILITY AND APPLICABILITY

The assignment, plus all supporting software, is available at http://web.eecs.utk.edu/~plank/plank/classes/cs494/494/labs/Lab-1-Btree. As with all of my programming courses, I developed automatic grading software that applies 100 test cases to the students' libraries (both alone and in conjunction with my programs). This helps them develop their code, debug and test. All of the code and scripts may be ported to other systems with minimal effort.

This assignment would be appropriate in Computer Science and Computer Engineering curricula in the following places:

- **Systems Programming**: At the University of Tennessee, we require our CS majors to take a Systems Programming course whose focus is on low-level programming above the Unix operating system. This course dovetails beautifully with standard operating systems courses, and would be an appropriate place for this assignment.
- **Operating Systems**: One of the strengths of this assignment is that it delivers the “flavor” of an Operating Systems assignment without all of the software (and sometimes hardware) scaffolding that is required for a real OS assignment.
- **Group Project for an Algorithms Class**: Although the ``systems'' flavor of this assignment make it difficult for an Algorithms class, it would make a good group project.

REFERENCES

[1] Weiss, M. A., *Data Structures and Algorithm Analysis in C++*, 3rd Edition, Boston, MA: Addison Wesley, 1007.