2014 Spring CS140 Final Exam - James Plank

Question 1 - Big O

Part A: Suppose my friend Fred wants to prove the following statement:

$$5n^3 + 500\ log_2(n) = O(n^3)$$

To do that, Fred is going to have to choose two constants, c and x_0. Which of the following statements must be true about those constants in order for Fred to prove his statement is true:

A. For all $n \geq x_0$, $n^3 \geq c(5n^3 + 500\ log_2(n))$.

B. For all $n \leq x_0$, $n^3 \geq c(5n^3 + 500\ log_2(n))$.

C. For all $n \geq x_0$, $n^3 \leq c(5n^3 + 500\ log_2(n))$.

D. For all $n \leq x_0$, $n^3 \leq c(5n^3 + 500\ log_2(n))$.

E. For all $n \geq x_0$, $x_0(n^3) \geq c(5n^3 + 500\ log_2(n))$.

F. For all $n \leq x_0$, $x_0(n^3) \geq c(5n^3 + 500\ log_2(n))$.

G. For all $n \geq x_0$, $x_0(n^3) \leq c(5n^3 + 500\ log_2(n))$.

H. For all $n \leq x_0$, $x_0(n^3) \leq c(5n^3 + 500\ log_2(n))$.

I. For all $n \geq x_0$, $cn^3 \geq 5n^3 + 500\ log_2(n)$.

J. For all $n \leq x_0$, $cn^3 \geq 5n^3 + 500\ log_2(n)$.

K. For all $n \geq x_0$, $cn^3 \leq 5n^3 + 500\ log_2(n)$.

L. For all $n \leq x_0$, $cn^3 \leq 5n^3 + 500\ log_2(n)$.

M. For all $n \geq x_0$, $cn^3 \geq x_0(5n^3 + 500\ log_2(n))$.

N. For all $n \leq x_0$, $cn^3 \geq x_0(5n^3 + 500\ log_2(n))$.

O. For all $n \geq x_0$, $cn^3 \leq x_0(5n^3 + 500\ log_2(n))$.

P. For all $n \leq x_0$, $cn^3 \leq x_0(5n^3 + 500\ log_2(n))$.

Part B: Which of the following assignments of these constants will suffice to prove the statement (just choose one):

- A. $c = 0, x_0 = 0$
- B. $c = 1, x_0 = 1$
- C. $c = 1, x_0 = 2$
- D. $c = 1, x_0 = 1024$
- E. $c = 5, x_0 = 1$
- F. $c = 5, x_0 = 2$
- G. $c = 5, x_0 = 1024$
- H. $c = 10, x_0 = 1$
- I. $c = 10, x_0 = 2$
- J. $c = 10, x_0 = 1024$

Question 2 - Linked Lists

Behold the following program:

```
#include "dlist.h"
#include <iostream>
using namespace std;

main()
{
  Dlist b;
  Dnode *p, *x, *y;

  b.Push_Back("A");
  b.Push_Back("B");
  b.Push_Back("C");
  b.Push_Back("D");

  p = b.Begin();
  x = b.End();
  y = p->flink;

  cout << y->s << endl;
  cout << p->flink->flink->s << endl;
  cout << x->blink->s << endl;

  // You would never do this, but I'm
  // testing your understanding of dlists

  x->s = "X";
  p->flink = x->flink;

  // More print statements

  cout << p->flink->s << endl;
  cout << y->blink->flink->s << endl;
  cout << y->blink->blink->s << endl;
}
```

Part A: Please tell me the first six lines printed on standard output.

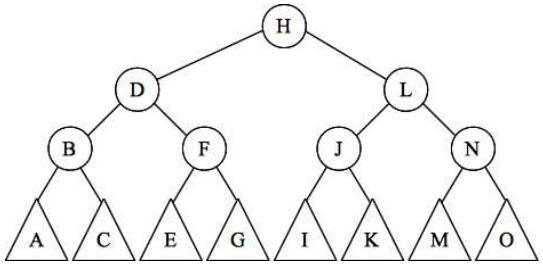
Part B: After the 6th line of output, the program will die on a segmentation violation. To the best of your ability, tell me exactly why.

The cheat sheet includes the header file for Dlists.

2014 CS140 Final Exam

Question 3 - AVL Trees

For this question, you are going to start with the tree to the right. In this tree, the circles are nodes, and the triangles are all valid AVL trees whose keys start with the letter in the tree.



I am going to give you six scenarios. In each, I tell you the heights of the trees in the in triangles, and I tell you what action has just occured. The state of the tree is before the rebalancing. You need to tell me what rebalancing operations need to be performed.

Choose your answers from the multiple choice that follow the scenarios.

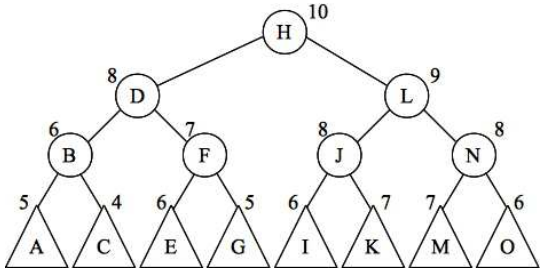
Heights of trees in triangles, and scenario

A	C	E	G	I	K	M	O	Scenario
5	4	6	5	6	7	7	6	1. We have inserted a new node into M
47	46	45	46	44	45	45	45	2. We have deleted a node from O
83	82	83	83	84	85	83	84	3. We have deleted a node from E
17	17	18	19	16	17	16	17	4. We have inserted a new node into G
35	34	33	33	34	33	33	34	5. We have deleted a node from G
60	59	61	60	59	58	59	58	6. We have inserted a new node into E

Multiple Choice Answers:

- *a*: Do nothing
- *b*: Single rotate about B
- *c*: Double rotate about B
- *d*: Single rotate about D
- *e*: Double rotate about D
- *f*: Single rotate about F
- *g*: Double rotate about F
- *h*: Single rotate about H
- *i*: Double rotate about H
- *j*: Single rotate about J
- *k*: Double rotate about J
- *l*: Single rotate about L
- *m*: Double rotate about L
- *n*: Single rotate about N
- *o*: Double rotate about N

Just so we're not confused, I'm going to answer scenario 1 for you. We have just inserted a node into tree M, and its height after the insertion is 7. The resulting tree is drawn below, and I've put all of the heights in all of the nodes. You can see that this tree is a valid AVL tree, so we don't need to do any rebalancing. The answer is ``*a:* Do nothing."



Question 4: Code analysis

There are eight implementations of the procedure **Q4**() below. For each implementation, answer the following question: Suppose I run this with a particular value of *n*, and it takes 10 seconds. Suppose I run it again, but I double the value of *n*. How long should I expect it to run? Choose your answers from the following multiple choice:

10 seconds	11 seconds	20 seconds	22 seconds	40 seconds	44 seconds	> 60 seconds

```
// Implementation 1

void Q4(vector <int> &v, int n)
{
  int i;

  v.clear();
  for (i = 0; i < n; i++) v.push_back(i);
  for (i = 0; i < n; i++) v.push_back(i);
  for (i = 0; i < n; i++) v.push_back(i);
  for (i = 0; i < n; i++) v.push_back(i);
}
```

```
// Implementation 2

void Q4(set <int> &v, int n)
{
  int i;

  v.clear();
  for (i = 0; i < n; i++) v.insert(i%10);
}
```

```
// Implementation 3

void Q4(multiset <int> &v, int n)
{
  int i;

  v.clear();
  for (i = 0; i < n; i++) v.insert(i%10);
}
```

```
// Implementation 4

void Q4(vector <int> &v, int n)
{
  int i;

  v.clear();
  for (i = 0; i < n; i++) v.push_back(i);
}
```

```
// Implementation 5

void Q4(vector <int> &v, int n)
{
  int i, j;

  v.clear();
  for (i = 0; i < n; i++) {
    for (j = 0; j < i; j++) v.push_back(i+j);
  }
}
```

```
// Implementation 6

void Q4(vector <int> &v, int n)
{
  int i, j;

  v.clear();
  for (i = 0; i < n; i++) {
    for (j = 0; j < i; j++) v.push_back(i);
  }
  for (i = 0; i < n; i++) {
    for (j = 1; j < n; j *= 2) v.push_back(i+j);
  }
}
```

```
// Implementation 7

int Q4(int n)
{
  if (n == 0) return 1;
  return 1 + Q4(n-1);
}
```

```
// Implementation 8

int Q4(int n)
{
  if (n == 0) return 1;
  return Q4(n-1) + Q4(n-1);
}
```

Question 5 - Sets and Maps

You are to solve the following problem:

Standard input is going to be composed of lines of text, where each line contains a person's first name and last name. We are going to define the "best" last name to be the last name that has the most different first names. If multiple last names have the same number of first names, then the "best" one is the lexicographically smallest.

Your job is to write a program that prints out, in lexicographic order, the first names of all the people with the "best" last name. I want you to use sets and maps from the C++ Standard Template Library.

Do this in two parts. In the first part, I want to you describe, in words, what data structures you are using to solve the problem, and what each data structure contains.

In the second part, I want to you write the C++ code to solve the problem.

Cheat Sheet

STL Data Structures and Relevant Methods

- **Methods common to all:** size(), begin(), end(), rbegin(), rend(), empty(), clear(), erase(iterator)
- **Strings**: length(), push_back(), find(char), find(string), substr(index, count)
- **Vectors**: push_back(), pop_back(),
- **Deques**: push_back(), push_front(), pop_back(), pop_front()
- **Lists**: push_back(), push_front(), pop_back(), pop_front()
- **Sets**: insert(), find()
- **Maps**: insert(pair), find()

Class Definitions of Data Structures from Class

```
class Dnode {
  public:
    string s;
    Dnode *flink;
    Dnode *blink;
};

class Dlist {
  public:
    Dlist();
    ~Dlist();
    int Empty();
    int Size();
    void Push_Front(string s);
    void Push_Back(string s);
    string Pop_Front();
    string Pop_Back();
    Dnode *Begin();
    Dnode *End();
    Dnode *Rbegin();
    Dnode *Rend();
    void Insert_Before(string s, Dnode *n);
    void Insert_After(string s, Dnode *n);
    void Erase(Dnode *n);
  protected:
    Dnode *sentinel;
    int size;
};
```

```
class Qnode {
  public:
    string s;
    Qnode *ptr;
};

class Queue {
  public:
    Queue();
    ~Queue();
    int Empty();
    int Size();
    void Push(string s);
    string Pop();
  protected:
    Qnode *first;
    Qnode *last;
    int size;
};
```

```
class Stacknode {
  public:
    string s;
    Stacknode *next;
};

class Stack {
  public:
    Stack();
    ~Stack();
    int Empty();
    void Push(string s);
    string Pop();
  protected:
    Stacknode *top;
};
```

Scratch