

CS 5050 Program 1 (30 points)

Optimal Binary Search Trees

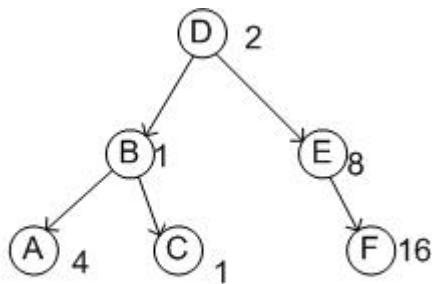
Introduction

A binary search tree is a tree where the key values are stored in the internal nodes, the external nodes (leaves) are null nodes, and the keys are ordered lexicographically, i.e. for each internal node all the keys in the left subtree are less than the keys in the node, and all the keys in the right subtree are greater. **You should have created binary search trees in the prerequisite course.**

When we know the frequency of searching each one of the keys, it is quite easy to compute the expected cost of accessing each node in the tree. An optimal binary search tree is a BST which has minimal expected cost of locating each node. In this version of the problem, we are not concerned with the frequency of searching for a missing node. (If we needed that, we would also be concerned with the height of the null pointers.)

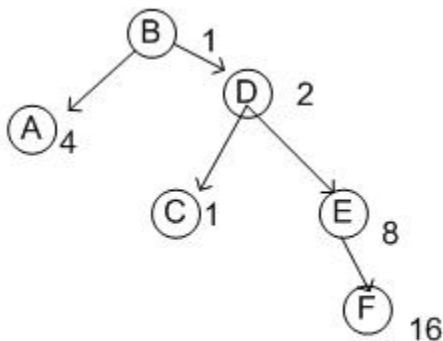
Example:

Node ID	0	1	2	3	4	5
Key	A	B	C	D	E	F
Frequency	4	1	1	2	8	16



$$= [2 \cdot 1 + (1+8) \cdot 2 + (4+1+16) \cdot 3] = 83$$

The expected cost of 83 is computed by multiplying each frequency by its level (starting with the root at 1). In general, let $\text{bestCost}[i][j]$ be the expected cost of the best tree consisting of nodes i through j . A different tree will have a different expected cost:



$$= 1 \cdot 1 + (4+2) \cdot 2 + (1+8) \cdot 3 + 15 \cdot 4 = 104$$

It's clear that this tree is not optimal. - It is easy to see that if F is closer to the root, given its high frequency, the tree will have a lower expected cost.

Part 1: Building a BST (10 points, Due Sept 3)

Using the starter code provided (or some of your own), create the code to create a binary search tree with the data provided in each of the strings below. You are to (1) insert the nodes into a BST in the order given (2) print the tree prettily (3) print the height of the tree (4) compute the expected cost of a search of all the nodes. The format of each node is "nodeID-frequency".

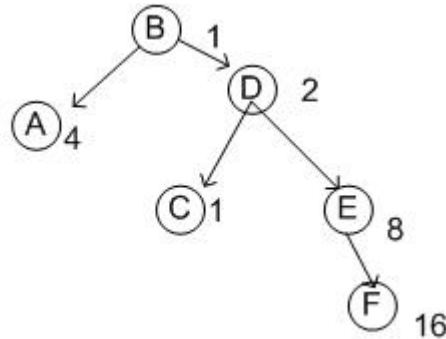
```
String nodeStr = "A-4 B-1 C-1 D-2 E-8 F-16";  
String nodeStr = "A-4 B-1 C-1 D-2 E-8 F-16 G-3 H-1 I-13 J-4 K-11 L-3 M-3";  
String nodeStr = "A-4 B-33 C-3 D-4 E-2 F-5 G-3 H-10 I-13 J-4 K-11 L-3 M-3";
```

Then, jumble the order of the nodes (already done in the code provided). Repeat the test outlined above: (1) insert the nodes into a BST in the order given (2) print the tree prettily (3) print the height of the tree (4) compute the expected cost of a search of all the nodes.

Print Prettily

Print out the tree prettily (on its side), in inorder, with each node (and frequency) on a line by itself and indented to show level. The tree would look like:

```
    F [16]  
   E [8]  
  D [2]  
 C [1]  
B [1]  
A [4]
```



Part 2 Creating an Optimal Binary Search Tree (20 points, Due Sept 10)

Criterion for an optimal tree:

Each optimal binary search tree is composed of a root and (at most) two optimal subtrees, the left and the right.

You are to solve this problem four ways: greedy, recursion, memoizing, and dynamic programming.

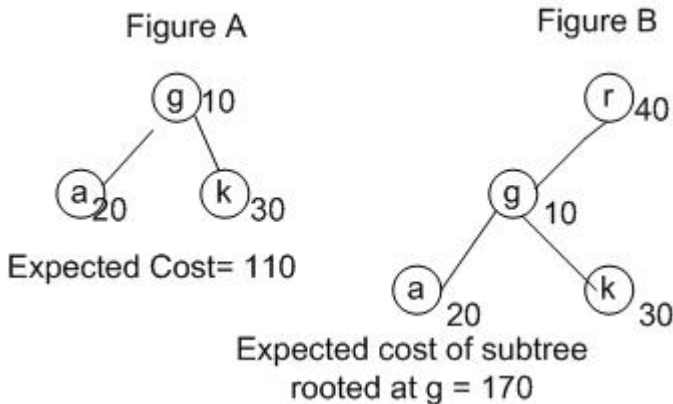
Greedy

With a greedy solution, you pick what appears to be the best node for the root (the one with the highest frequency), and then solve for the subtrees recursively (using a greedy heuristic). Once the root has been determined, the left subtree consists of all nodes with key less than the root. The right subtree consists of all the nodes with the key greater than the root. Note that a greedy solution may or may not be optimal.

(1) Show the tree produced by this method. (2) Print the tree prettily (3) Print the height of the tree (4) Compute the expected cost of a search of all the nodes.

Recursion

With recursion, you can try all solutions. A method which looks at all possibilities to find the best method is called an *exhaustive* method. The term exhaustive doesn't mean you will get tired computing it (although you will), but that you exhaust all possibilities. In using the solution to smaller problems, you need to determine how a solution to a smaller problem is used in determining the expected cost of the tree containing it. For example, if a node *r* has a left subtree with expected cost 110 (when that sub-tree is treated as a separate tree), how does that contribute to the expected cost of the tree rooted at *r*?



Compute the best expected cost of a search of all the nodes. **Note, you do not have to actually build the tree.**

Memoizing

Memoizing refers to writing yourself a memo saying, "Hey, don't compute this again! You've already done it once."

Rewrite the recursion with the following change. Initially, set $\text{bestCost}[i][j]$ to -100 to represent the expected cost of the optimal tree from *i* to *j*. Then as you compute the values recursively, if $\text{bestCost}[][]$ has already been computed, don't recompute it. Keep track of how many times you can look up the results rather than having to recompute it.

Compute the best expected cost of a search of all the nodes. **Print the bestCost matrix. Print the "repeated call count". Note, you do not have to actually build the tree.**

Dynamic Programming (Because of time, We will NOT do the dynamic programming part)

In some recursive solutions (without memoizing), work is increased because you keep solving the same subproblem.

The idea of dynamic programming is to save all the solutions to the subproblems so you can reuse them.

One difference in a dynamic programming solution is that you normally start with the smallest subproblem and determine the solutions for larger and larger subproblems (in terms of the smaller ones). Thus, you build the

value array starting with main diagonal of the bestCost matrix. There is no recursion in dynamic programming, so the method looks quite different than the other two.

Create an option to use this method and display both the matrix and the final expected cost. Actually create the tree with optimal expected cost and print it prettily.

Submission

Create a jar file of your project and submit it via Eagle.

Be sure to read the assignment carefully so you are not docked for easy things you forgot to do. Some students want the grader to continually regrade their assignments as they correct various omitted features. The grader doesn't have time to do this. Get it right the first time!

Report (printed by your program before the program output is printed)

1. How many hours did you spend on this problem?