# MA/CSSE 473 – Design and Analysis of Algorithms

## Homework 13 (90 points total)

When a problem is given by number, it is from the textbook. 1.1.2 means "problem 2 from section 1.1" .

## Problems for enlightenment/practice/review (not to turn in, but you should think about them):

How many of them you need to do serious work on depends on you and your background. I do not want to make everyone do one of them for the sake of the (possibly) few who need it. You can hopefully figure out which ones you need to do.

8.3.5 (Root of Optimal tree)
8.3.2 (Time and space efficiency of optimal BST calculation)
8.3.8 ($n^2$ algorithm for optimalBST. Not for the faint of heart!)
------ For the frequencies of the Day 33 class example (AEIOU), find he optimal tree if we consider only successful searches (set all qi to 0)
------ For the frequencies of the Day 33 class example (AEIOU), find he optimal tree if we consider only unsuccessful searches (set all pi to 0)
9.1.1 (Greedy change-making not optimal)
9.1.5 (greedy bridge crossing)

## Problems to write up and turn in:

1. (50) Optimal BST problem: described below.
2. (10) 8.3.3 (Optimal BST from root table)
3. ( 5) 8.3.10a (Matrix chain multiplication) Also think about parts (b) and (c), which may appear on a later assignment or exam.
4. ( 5) 8.3.4 (Sum for optimalBST in constant time).
5. (10) 8.3.6 (optimalBST--successful search only--if all probabilities equal)
6. (10) 9.1.3 (Greedy job scheduling)

**Optimal BST Dynamic Programming Problem ( problem #1)**

In a binary search tree, the key of each node in the left subtree is smaller than the key of the root, and the key of each node in the right subtree is larger than the root. The same property holds for each subtree. Section 8.3 discusses a dynamic programming algorithm to find an optimal static tree if only successful searches are taken into account. In class (Days 30-31) we discussed (using a slightly different notation) a modified algorithm that takes unsuccessful searches into account.

Suppose that we have a static set of N keys $K_1, K_2, \dots , K_n$ (in increasing order), and that we have collected statistics about the frequency of searches for each key and for items in each "gap" between keys (i.e. each place that an unsuccessful search can end up).

For i = 1 ... n, let $a_i$ be the frequency of searches that end successfully at $K_i$. (I called these $p_i$ in class)

For i = 1 ... n-1, let $b_i$ be the frequency of unsuccessful searches for all "missing keys" that are between $K_i$ and $K_{i+1}$ (also, $b_0$ is the frequency of searches for keys smaller than $K_1$, and $b_n$ is the frequency for keys that are larger than $K_n$). (I called these $q_i$ in class)

We build an extended BST T (see Figure 4.5 for a diagram of an extended tree) whose internal nodes contain the N keys, $K_1, \ldots, K_n$. Let $x_i$ be the depth of the node containing $K_i$, and let $y_i$ be the depth of the "external node" that represents the gap between $K_i$ and $K_{i+1}$ (where $y_0$ and $y_n$ are the depths of the leftmost and rightmost external nodes of T). Recall that the depth of a tree's root is 0. The optimal tree for the given keys and search frequencies is one that minimizes the *weighted path length* C(T), where C is defined by

$$C = \sum_{i=1}^{N} a_i [1 + x_i] + \sum_{i=0}^{N} b_i y_i$$

For example, in class we considered the following data:

| i | $K_i$ | $a_i$ | $b_i$ |
|---|---|---|---|
| 0 | | | 0 |
| 1 | A | 32 | 34 |
| 2 | E | 42 | 38 |
| 3 | I | 26 | 58 |
| 4 | O | 32 | 95 |
| 5 | U | 12 | 21 |

If we choose to build the BST with I as the root, E and O as I's children, A as E's child, and U as O's child, then C = 948, and the average search length is 948/390 = 2.43 (you should verify this, to check your understanding of the formula for C). It turns out that this tree is not optimal.

In class we discussed (with a slightly different notation) a dynamic programming algorithm that finds a tree that minimizes C for any set of n keys and 2n+1 associated frequencies. It uses an alternate, recursive, formulation of C:

(a) (10) Let T be a non-trivial BST. If T is empty, then C(T) = 0. Otherwise T has a root and two subtrees $T_L$ and $T_R$. Then $C(T) = C(T_L) + C(T_R) + \text{sum}(a_i, i=1..n) + \text{sum}(b_i, i=0..n)$. Show by induction that this recursive definition of C(T) is equivalent to the summation definition given above. [The recursive definition is used in the provided algorithm].

(b) (5) The algorithm and a Python implementation are provided, along with a table of final values for the above inputs. Use the information that table to draw the optimal tree.

(c) (10) What is the big-theta running time for the optimal-tree-generating algorithm? Show your computations that led you to this conclusion

(d) (25) (This is a challenging problem) Find a way to improve the given algorithm so that it has a smaller big-theta running time, and show that it really is smaller.