15-451 Algorithms, Fall 2000

Homework # 4 **due: Wed-Thurs Oct 25-26, 2000**

Groundrules:

- This is an oral presentation assignment. As stated in the course information handout, you should work in groups of three. At some point before **Monday Oct 23 at midnight** you should sign up for a 1-hour time slot on the signup sheet on the course web page.
- You are not required to hand anything in at your presentation, but you may if you choose. If you do hand something in, it will be taken into consideration (in a non-negative way) in the grading.

Problems:

1. [Graph searching] Let G be a directed graph represented using an adjacency list. So, each node $G[i]$ has a list of all nodes reachable in 1 step from i (all out-neighbors of i). Suppose each node of G also has a value: e.g., node 1 might have value \$100, node 2 might have value \$50, etc.

   Given an $O(|E|+|V|\log|V|)$ time algorithm that computes, for every node, the highest value reachable from that node. For instance, if G is strongly-connected (Manber, section 7.9.2), then for every node this will be the maximum value of all nodes in the entire graph. Hint: one worthwhile preprocessing step is to sort nodes by value.

   Extra credit: Give an $O(|E|+|V|)$ algorithm. In this case, you don't have time to sort by value.

2. [The List-Update Problem] Suppose we have n data items $x_1, x_2, \ldots, x_n$ that we wish to store in a linked list in some order. Let's say the cost for performing a $lookup(x)$ operation is \$1 if x is in the head of the list, \$2 if x is the second element in the list, and so on.

   For instance, say there are 4 items and it turns out that we end up accessing x_1 3 times, x_2 5 times, x_3 once, and x_4 twice. In this case, in hindsight, the best ordering for a linked list would have been (x_2, x_1, x_4, x_3) with a total cost of \$21.

   The *Move-to-Front* (MTF) strategy is the following algorithm for organizing the list if we don't know in advance how many times we will access each element. We begin with the elements in their initial order $(x_1, x_2, \ldots, x_n)$. Then, whenever we perform a $lookup(x)$ operation, we move the item accessed to the front of the list. Let us say that performing the movement is free. For instance, if the first operation was $lookup(x_3)$, then we pay \$3, and afterwards list will look like $(x_3, x_1, x_2, x_4 \ldots)$.

   (a) Suppose $n = 4$ and we perform the following 4 operations:

   $$lookup(x_4), lookup(x_2), lookup(x_4), lookup(x_2).$$

   What does the list look like in the end and what was the total cost?

(b) Your job is to prove that the total cost of the MTF algorithm on a sequence of m operations (think of m as much larger than n) is at most $2C_{static} + n^2$ where C_{static} is the cost of the best static list in hindsight for those m operations (like in our first example). We will prove this in two steps.

i. First prove the somewhat easier statement that the cost of Move-to-Front is at most $2C_{initial}$ where $C_{initial}$ is the cost of the original ordering $(x_1, x_2, \ldots, x_n)$.

Hint: If $i < j$ but x_j is in front of x_i in the MTF list, let's say that x_j has "cut in line" in front of x_i. Now, imagine that each element x_i has a piggy bank with \$1 for everyone that is currently cutting in line in front of it.

ii. Now prove the $2C_{static} + n^2$ bound.

3. [Optimal Search Trees] Consider a binary search tree storing a set of keys $x_1 < x_2 < x_3 < \ldots < x_n$. Let's define the *cost* of handling a request for some key to be the number of comparisions made in searching for it (one plus the distance of the node from the root of the tree). For example, if the root is requested, the cost is 1.

Given a particular sequence of requests, one can calculate the cost that would be incurred on that sequence by different possible binary search trees. The tree that attains the minimum cost is called the *optimal binary search tree* for that sequence.

(a) For a fixed tree, the cost of a given sequence of requests clearly only depends on the number of times each key is requested, not on their order. Suppose that $n = 4$ and that x_1 is accessed once, x_2 is accessed 9 times, x_3 is accessed 5 times, and x_4 is accessed 6 times. Find an optimal binary tree for this set of requests.

(b) Give a general algorithm for constructing the optimal binary tree given a sequence of counts $c_1, c_2, \ldots, c_n$ (c_i is the number of times x_i is accessed). The running time of your algorithm should be $O(n^3)$.

Note: Splay trees perform the role in this problem that the MTF algorithm performed in the previous problem. The access costs in a splay tree are guaranteed to be within a constant factor of the cost of an optimal static tree for the given sequence. This fact can be proven using the access lemma. (But you don't have to do this for your assignment.)