

Homework due 10/18. Please write the name of any collaborator you worked with. **This should be more difficult than the previous assignment.**

Since there seemed to be a lot of interest in Splay trees, the homework problems will be geared towards them.

Recall that a binary search tree is a tree of data with a value stored at each node. It has the property that for every node  $i$ , each node in the right subtree has larger value than  $i$  and each node in the left subtree has smaller value than  $i$ . Trees are two types of rotations that may be performed on a binary search tree, right and left rotations. These operations maintain the invariants of a binary search tree. You can find definitions of rotations in the course notes, or at <http://www.cs.mcgill.ca/cs251/OldCourses/1997/topic9/>. Here's a java demo: <http://www.cs.purdue.edu/homes/cmh/AlgAnim/IterardiExample>.

1. Prove that any binary search tree  $T_1$  on  $\{1, 2, \dots, n\}$  can be transformed into any other binary search tree  $T_2$  on  $\{1, 2, \dots, n\}$  in at most  $2n$  rotations.
2. Suppose there is a sequence of requests to items  $\{1, 2, \dots, n\}$ . We would like to find a single (fixed, static) binary search tree  $T$  which minimizes the cost of these lookups. (The cost of a lookup is the depth of the requested item in the fixed tree.) This is called an *optimal binary search tree* for that sequence.
  - (a) Suppose that  $n = 4$  and item 1 is requested once, item 2 is requested 9 times, item 3 is requested 5 times, and item 4 is requested 6 times. Find an optimal binary search tree for these requests.
  - (b) Describe a general algorithm for constructing the optimal binary search tree in hindsight given a sequence of counts  $c_1, c_2, \dots, c_n$ , where  $c_i$  is the number of times item  $i$  is requested. The running time of your algorithm should be at most  $O(n^4)$ , but may be better. Analyze the runtime of your algorithm. (Hint: use dynamic programming.)
3. Randomized frequency count for trees. (Extra credit)

The frequency count algorithm for binary search trees maintains a

count of the number of times each item has been requested so far. It then keeps the tree in the shape of the optimal binary search tree (so far), using an algorithm like your solution to the previous problem. It has a poor competitive ratio.

The randomized frequency count algorithm for trees is similar to the randomized frequency count algorithm that we studied for lists. A counter  $c[i]$ , initially 0, is maintained for each item. After each request, the count of that item is incremented with probability  $1/2$  and kept the same with probability  $1/2$ . Again, the optimal binary search tree (so far) is maintained.

In this problem we want to show that this algorithm has the following guarantee for any sequence of lookups,  $r_1, r_2, \dots, r_T$ :

$$E[\text{cost of randomized freq. count}] \leq \text{min-static-cost} + O(n^3\sqrt{T}).$$

Here, min-static-cost refers to the cost of the best single tree in hindsight, and we are considering only lookup costs (not considering the movement costs of performing rotations). The analysis follows that of lists in Lecture 3.

(a) First consider the regular frequency count algorithm. Argue that:

$$\text{cost of freq. count} \leq \text{min-static-cost} + n(\# \text{ of changes})$$

(b) Then consider randomized frequency count, and the  $m$ th request to any particular item  $i$ . At this point the count of item  $i$  is somewhere between 0 and  $m$ . Show that the probability of randomized frequency count changing trees is at most  $cn/\sqrt{m}$  for some constant  $c$ . (Hint: as you keep everything else fixed and increase the count of item  $i$  from 0 to  $m$ , how many different optimal trees are there?)

This then implies that the expected number of times randomized frequency count changes trees is at most  $O(n^2\sqrt{T})$  because there are  $n$  items and each one causes at most an expected  $O(n\sqrt{T})$  changes. Finally, combining this with (a) and the same analysis of expected costs as in lecture 3, gives the  $O(n^3\sqrt{T})$  regret bound.