Homework 3

Aleck Zhao

September 23, 2017

1 Dumbbell Matching Revisited

Recall the dumbbell matching problem from the last homework. You designed a randomized algorithm which took $O(n \log n)$ comparisons. Prove a matching lower bound for deterministic algorithms: prove that any deterministic algorithm which solves the problem must make at least $\Omega(n \log n)$ comparisons.

2 Algorithms for Sorted Arrays

Suppose that you are given a sorted array A of length n and two values x and y with $x < y$. Consider the following problem: determine a) how many elements of A are less than x, b) how many elements of A are at least x and at most y, and c) how many elements of A are larger than y. Find a function $f(n)$ and prove the following about it.

(a) Any algorithm for this problem in the comparison model must have running time $\Omega(f(n))$.

(b) There is an algorithm for the problem which runs in time $O(f(n))$ (give such an algorithm and prove running time and correctness).

3 Range Queries

We saw in class how to use binary search trees as dictionaries, and in particular how to use them to do insert and lookup operations. Some of you might naturally wonder why we bother to do this, when hash tables already allow us to do this. While there are many good reasons to use search trees rather than hash tables, one informal reason is that search trees can in some cases be either used directly or easily extended to allow efficient queries that are difficult or impossible to do efficiently in a hash table.

An important example of this is a range query. Suppose that all keys are distinct. In addition to being able to insert and lookup (and possibly delete), we want to allow a new operation range(x, y) which is suppose to return the number of keys in the tree which are at least x and at most y.

In this problem we will only be concerned with normal binary search trees. Recall that in binary search trees of height h, inserts can be done in $O(h)$ time.

(a) Given a binary search tree of height h, show how to implement range(x, y) in $O(h + k)$ time, where k is the number of elements that are at least x and at most y.

Can we do this operation even faster? It turns out that we can! In particular, for a binary search tree of height h, we can do this in $O(h)$ time.

(b) Describe an extra piece of information that you will store at each node of the tree, and describe how to use this extra information to do the above range query in $O(h)$ time. (Hint: think of keeping track of a size.)

(c) Describe how to maintain this information in $O(h)$ time when a new node is inserted (note that there are no rotations on an insert - it's just the regular binary search tree insert, but you need to update information appropriately).