Introduction to Algorithms October 17, 2005
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik D. Demaine and Charles E. Leiserson Handout 15

Problem Set 4

MIT students: This problem set is due in lecture on **Monday, October 24, 2005.** The homework lab for this problem set will be held 2–4 P.M. on Sunday, October 23, 2005.

Reading: Chapters 12–13, 18

Both exercises and problems should be solved, but *only the problems* should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated. **Please staple and turn in your solutions on 3-hole punched paper**.

You will often be called upon to "give an algorithm" to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

Remember, your goal is to communicate. Full credit will be given only to correct solutions *which are described clearly*. Convoluted and obtuse descriptions will receive low marks.

Exercise 4-1. Do Exercise 12.1-5 on page 256 of CLRS.

Exercise 4-2. Do Exercise 12.2-4 on page 260 of CLRS.

Exercise 4-3. Do Exercise 12.4-3 on page 268 of CLRS.

Exercise 4-4. Do Exercise 13.1-6 on page 277 of CLRS.

Exercise 4-5. Do Exercise 13.3-1 on page 287 of CLRS.

Exercise 4-6. Do Exercise 18.2-6 on page 449 of CLRS.

Problem 4-1. Treaps

If we insert a set of n items into a binary search tree using TREE-INSERT, the resulting tree may be horribly unbalanced. As we saw in class, however, we expect randomly built binary search trees to be balanced. (Precisely, a randomly built binary search tree has expected height $O(\lg n)$.) Therefore, if we want to build an expected balanced tree for a fixed set of items, we could randomly permute the items and then insert them in that order into the tree.

What if we do not have all the items at once? If we receive the items one at a time, can we still randomly build a binary search tree out of them?

We will examine a data structure that answers this question in the affirmative. A ***treap*** is a binary search tree with a modified way of ordering the nodes. Figure 1 shows an example of a treap. As usual, each item x in the tree has a key *key*$[x]$. In addition, we assign *priority*$[x]$, which is a random number chosen independently for each x. We assume that all priorities are distinct and also that all keys are distinct. The nodes of the treap are ordered so that (1) the keys obey the binary-search-tree property and (2) the priorities obey the min-heap order property. In other words,

- if v is a left child of u, then *key*$[v] <$ *key*$[u]$;
- if v is a right child of u, then *key*$[v] >$ *key*$[u]$; and
- if v is a child of u, then *priority*$(v) >$ *priority*(u).

(This combination of properties is why the tree is called a "treap": it has features of both a binary search tree and a heap.)

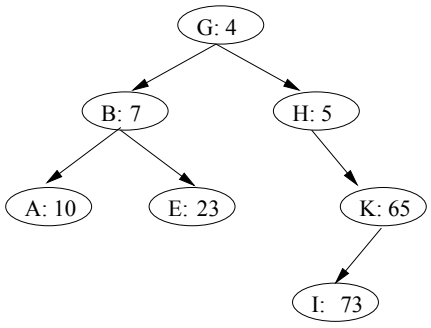

Figure 1: A treap. Each node x is labeled with *key*$[x]$: *priority*$[x]$. For example, the root has key G and priority 4.

It helps to think of treaps in the following way. Suppose that we insert nodes $x_1, x_2, \ldots, x_n$, each with an associated key, into a treap in arbitrary order. Then the resulting treap is the tree that would have been formed if the nodes had been inserted into a normal binary search tree in the order given by their (randomly chosen) priorities. In other words, *priority*$[x_i] <$ *priority*$[x_j]$ means that x_i is effectively inserted before x_j.

(a) Given a set of nodes $x_1, x_2, \ldots, x_n$ with keys and priorities all distinct, show that there is a unique treap with these nodes.

(b) Show that the expected height of a treap is $O(\lg n)$, and hence the expected time to search for a value in the treap is $O(\lg n)$.

Let us see how to insert a new node x into an existing treap. The first thing we do is assign x a random priority *priority*$[x]$. Then we call the insertion algorithm, which we call TREAP-INSERT, whose operation is illustrated in Figure 2.

(c) Explain how TREAP-INSERT works. Explain the idea in English and give pseudocode. (*Hint:* Execute the usual binary search tree insert and then perform rotations to restore the min-heap order property.)

(d) Show that the expected running time of TREAP-INSERT is $O(\lg n)$.

TREAP-INSERT performs a search and then a sequence of rotations. Although searching and rotating have the same asymptotic running time, they have different costs in practice. A search reads information from the treap without modifying it, while a rotation changes parent and child pointers within the treap. On most computers, read operations are much faster than write operations. Thus we would like TREAP-INSERT to perform few rotations. We will show that the expected number of rotations performed is bounded by a constant (in fact, less than 2)!

In order to show this property, we need some definitions, illustrated in Figure 3. The ***left spine*** of a binary search tree T is the path which runs from the root to the item with the smallest key. In other words, the left spine is the maximal path from the root that consists only of left edges. Symmetrically, the ***right spine*** of T is the maximal path from the root consisting only of right edges. The ***length*** of a spine is the number of nodes it contains.

(e) Consider the treap T immediately after x is inserted using TREAP-INSERT. Let C be the length of the right spine of the left subtree of x. Let D be the length of the left spine of the right subtree of x. Prove that the total number of rotations that were performed during the insertion of x is equal to $C + D$.

We will now calculate the expected values of C and D. For simplicity, we assume that the keys are $1, 2, \ldots, n$. This assumption is without loss of generality because we only compare keys.

For two distinct nodes x and y, let $k = \textit{key}[x]$ and $i = \textit{key}[y]$, and define the indicator random variable

$$X_{i,k} = \begin{cases} 1 & \text{if } y \text{ is a node on the right spine of the left subtree of } x \text{ (in } T\text{)}, \\ 0 & \text{otherwise.} \end{cases}$$

(f) Show that $X_{i,k} = 1$ if and only if (1) $\textit{priority}[y] > \textit{priority}[x]$, (2) $\textit{key}[y] < \textit{key}[x]$, and (3) for every z such that $\textit{key}[y] < \textit{key}[z] < \textit{key}[x]$, we have $\textit{priority}[y] < \textit{priority}[z]$.

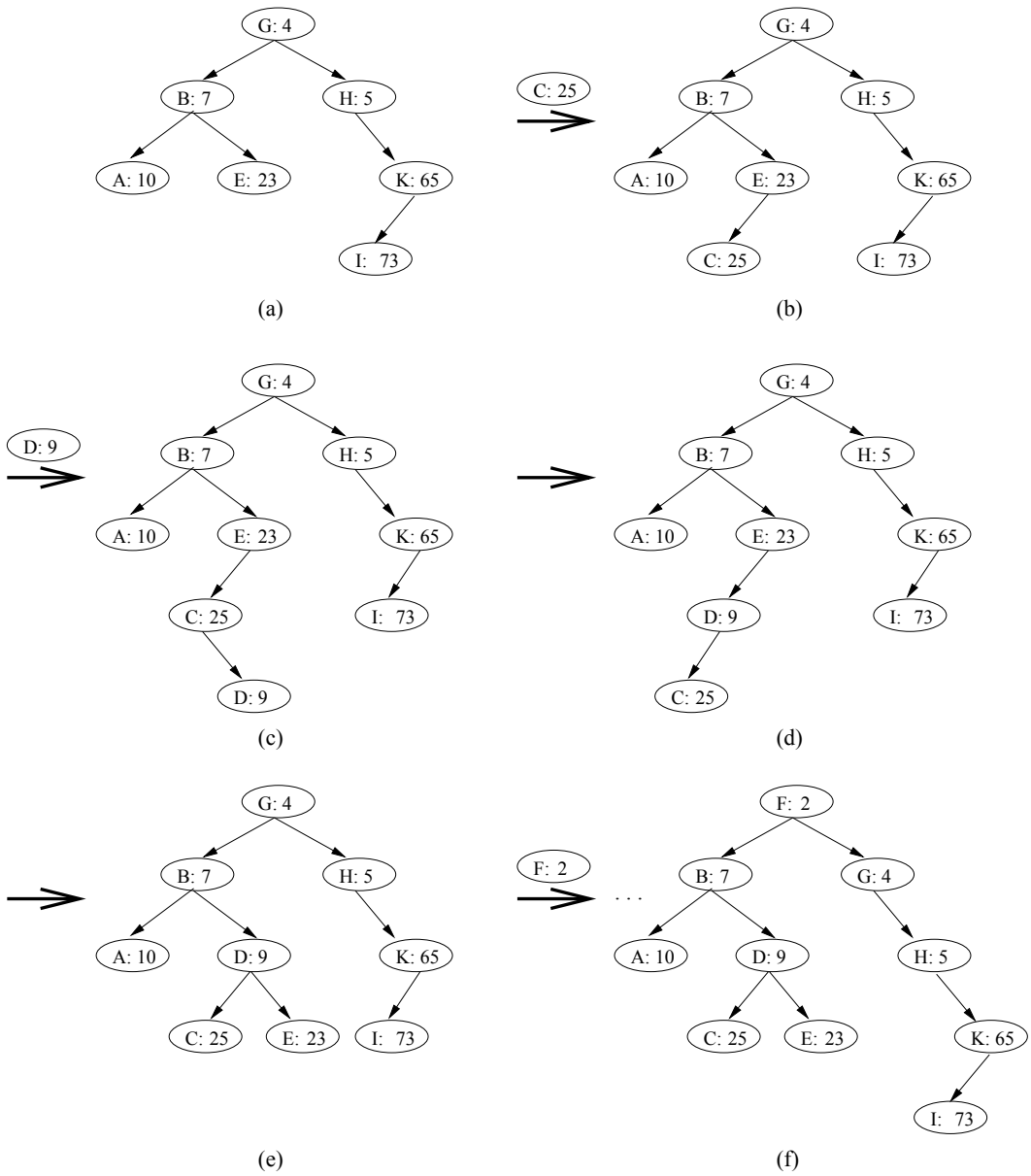

Figure 2: Operation of TREAP-INSERT. As in Figure 1, each node x is labeled with *key*$[x]$: *priority*$[x]$. **(a)** Original treap prior to insertion. **(b)** The treap after inserting a node with key C and priority 25. **(c)–(d)** Intermediate stages when inserting a node with key D and priority 9. **(e)** The treap after insertion of parts (c) and (d) is done. **(f)** The treap after inserting a node with key F and priority 2.

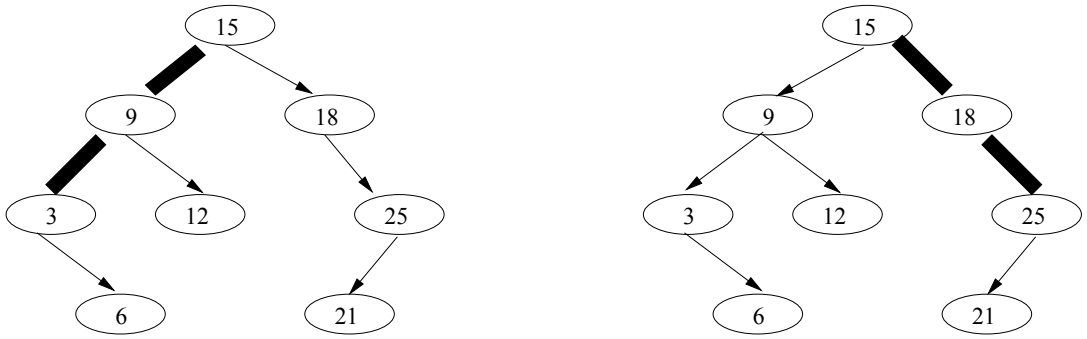

Figure 3: Spines of a binary search tree. The left spine is shaded in **(a)**, and the right spine is shaded in **(b)**.

(g) Show that

$$\Pr\{X_{i,k} = 1\} = \frac{(k-i-1)!}{(k-i+1)!} = \frac{1}{(k-i+1)(k-i)}.$$

(h) Show that

$$\mathrm{E}[C] = \sum_{j=1}^{k-1} \frac{1}{j(j+1)} = 1 - \frac{1}{k}.$$

(i) Use a symmetry argument to show that

$$\mathrm{E}[D] = 1 - \frac{1}{n-k+1}.$$

(j) Conclude that the expected number of rotations performed when inserting a node into a treap is less than 2.

Problem 4-2. Being balanced

Call a family of trees ***balanced*** if every tree in the family has height $O(\lg n)$, where n is the number of nodes in the tree. (Recall that the ***height*** of a tree is the maximum number of edges along any path from the root of the tree to a leaf of the tree. In particular, the height of a tree with just one node is 0.)

For each property below, determine whether the family of binary trees satisfying that property is balanced. If you answer is "no", provide a counterexample. If your answer is "yes", give a proof (hint: it should be a proof by induction). Remember that being balanced is an *asymptotic* property, so your counterexamples must specify an infinite set of trees in the family, not just one tree.

(a) Every node of the tree is either a leaf or it has two children.

(b) The size of each subtree can be written as $2^k - 1$, where k is an integer (k is *not* the same for each subtree).

(c) There is a constant $c > 0$ such that, for each node of the tree, the size of the smaller child subtree of this node is at least c times the size of the larger child subtree.

(d) There is a constant c such that, for each node of the tree, the heights of its children subtrees differ by at most c.

(e) The average depth of a node is $O(\lg n)$. (Recall that the ***depth*** of a node x is the number of edges along the path from the root of the tree to x.)