Introduction to Algorithms March 19, 2004
Massachusetts Institute of Technology 6.046J/18.410J
Professors Erik Demaine and Shafi Goldwasser Handout 15

Problem Set 5

Reading: Chapters §18.1-18.2, 14.1-14.3, 33.1-33.3, Skip Lists Handout

There are **four** problems. Each problem is to be done on a **separate sheet** (or sheets) of paper. 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. As on previous assignments, "give an algorithm" entails providing a description, proof, and runtime analysis.

Problem 5-1. Joining and Splitting 2-3-4 Trees

The JOIN operator takes as input two 2-3-4 trees, T_1 and T_2, and an element x such that for any $y_1 \in T_1$ and $y_2 \in T_2$, we have $key[y_1] < key[x] < key[y_2]$. As output JOIN returns a 2-3-4 tree T containing the node x and all the elements of T_1 and T_2.

The SPLIT operator is like an "inverse" JOIN: given a 2-3-4 tree T and an element $x \in T$, SPLIT creates a tree T_1 consisting of all elements in $T - \{x\}$ whose keys are less than $key[x]$, and a tree T_2 consisting of all elements in $T - \{x\}$ whose keys are greater than $key[x]$.

In this problem, we will efficiently implement JOIN and SPLIT. For convenience, you may assume that all elements have unique keys.

(a) Suppose that in every node x of the 2-3-4 tree there is a new field $height[x]$ that stores the height of the subtree rooted at x. Show how to modify INSERT and DELETE to maintain the $height$ of each node while still running in $O(\log n)$ time. Remember that all leaves in a 2-3-4 tree have the same depth.

(b) Using part (a), give an $O(1 + |h_1 - h_2|)$-time JOIN algorithm, where h_1 and h_2 are the heights of the two input 2-3-4 trees.

(c) Give an $O(\log n)$-time SPLIT algorithm. Your algorithm will take a 2-3-4 tree T and key k as input. To write your SPLIT algorithm, you should take advantage of the search path from T's root to the node that would contain k. This path will consist of a set of keys $\{k_1, \ldots, k_m\}$. Consider the left and right subtrees of each key k_i and their relationship to k. You may use your JOIN procedure from part (b) in your solution.

Problem 5-2. AVL Trees

An ***AVL tree*** is a binary search tree that is ***height balanced***: for each node x, the heights of the left and right subtrees of x differ by at most 1. Height is defined to be the length of the longest path from a node to any leaf in the tree rooted at that node. To implement an AVL tree, we maintain an extra field in each node: $h[x]$ is the height of node x. As for any other binary search tree T, we assume that *root*$[T]$ points to the root node.

(a) Prove that an AVL tree with n nodes has height $O(\log n)$. (*Hint:* Prove that in an AVL tree of height h, there are at least F_h nodes, where F_h is the hth Fibonacci number.)

(b) To insert into an AVL tree, a node is first placed in the appropriate place in binary search tree order. After this insertion, the tree may no longer be height balanced. Specifically, the heights of the left and right children of some node may differ by 2. Describe a procedure BALANCE(x), which takes a subtree rooted at x whose left and right children are height balanced and have heights that differ by at most 2, i.e., $|h[\textit{right}[x]] - h[\textit{left}[x]]| \leq 2$, and alters the subtree rooted at x to be height balanced. (*Hint:* Use rotations.)

(c) Using part (b), describe a recursive procedure AVL-INSERT(x, z), which takes a node x within an AVL tree and a newly created node z (whose key has already been filled in), and adds z to the subtree rooted at x, maintaining the property that x is the root of an AVL tree. As in TREE-INSERT from Section 12.3 in CLRS, assume that $\textit{key}[z]$ has already been filled in and that $\textit{left}[z] = \text{NIL}$ and $\textit{right}[z] = \text{NIL}$; also assume that $h[z] = 0$. Thus, to insert the node z into the AVL tree T, we call AVL-INSERT$(\textit{root}[T], z)$.

(d) Show that AVL-INSERT, run on an n-node AVL tree, takes $O(\log n)$ time and performs $O(1)$ rotations.

Problem 5-3. Order Statistics in Skip Lists

In this problem we implement the order statistics operations RANK and SEARCH-BY-RANK in a skip list. For a node x in a skip list L, RANK(x, L) gives the rank of x among the elements in the list. SEARCH-BY-RANK(k, L) is the inverse of RANK. It returns the k-th element in the skip list L. If no such node exists, it returns **nil**.

In this problem we assume that all elements in the skip list have distinct key values. Denote the top level of the skip list as level 1. You may assume a skip list L has a variable $L.depth$ that stores the number of levels in L.

(a) Show how you would augment the skip list data structure so that RANK and SEARCH-BY-RANK can be implemented with $O(\log n)$ time complexity.

(b) Modify SEARCH, INSERT and DELETE so that the operations run in $O(\log n)$ time with the augmented data structure.

(c) Give $O(\log n)$ implementations for RANK and SEARCH-BY-RANK.

Problem 5-4. Convex Layers

Given a set Q of points in the plane, we define the ***convex layers*** of Q inductively. The first convex layer of Q consists of those points in Q that are vertices of CH(Q). For $i > 1$, define Q_i to consist of the points of Q with all points in convex layers $1, 2, \ldots, i-1$ removed. Then the ith convex layer of Q is CH(Q_i) if $Q_i \neq \emptyset$ and is undefined otherwise.

(a) Give an $O(n^2)$-time algorithm to find the convex layers of a set on n points. Hint: Refer to CLRS chapter 33.3.

(b) Suppose we are given an unsorted array of n real values A. Let the array B contain the values of A in descending sorted order. Give a linear time algorithm to convert A to a set of points Q, such that each convex layer Q_i can be translated to $B[i]$ in constant time. In other words, give a linear time *reduction* from the sorting problem to the convex layer problem.