Introduction to Algorithms October 8, 2003
Massachusetts Institute of Technology 6.046J/18.410J
Professors Shafi Goldwasser and Silvio Micali Handout 15

Problem Set 4

This problem set is due **in lecture** on **Wednesday, October 22.**

Reading: Chapters §12.1-12.3 §18.1-18.2, §13, §14.

There are **four** problems. Each problem is to be handed in separately. 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.

Problem 4-1. Reconstructing Binary Trees Via Traversals

Recall the binary tree data structure; recall three algorithms for traversing the tree: the *inorder* traversal, the *preorder* traversal, and the *postorder* traversal.

(a) Suppose you are given the preorder traversal and the inorder traversal of a binary tree. Can you reconstruct the tree? If so, give an algorithm for doing so and prove its correctness. If not, give a counter example. Note that a binary tree is not necessarily a binary search tree.

(b) Suppose you are given the preorder and postorder traversals of a binary tree. Can you reconstruct the tree? If so, give an algorithm for doing so and prove its correctness. If not, give a counter example.

(c) Suppose you are given the preorder traversal of a binary tree. In addition, you are told that the tree is a binary search tree. Can you reconstruct the tree? If so, give an algorithm for doing so and prove its correctness. If not, give a counter example.

Problem 4-2. Successors In 2-3 Trees

A *2-3 tree* is a special B-tree in which each node which is not a leaf has 2 or 3 children, and every path from the root to a leaf is of the same length. Note that the tree consisting of a single node is a 2-3 tree.

An ordered set can be stored in a 2-3 tree by storing the elements of the set in the leaves of the tree. We assign the elements to leaves in increasing order from left to right. If a is an internal node of the tree, then two pieces of information are stored in it: $L[a]$ is the largest element stored in a leaf of the subtree whose root is the leftmost child of a; $M[a]$ is the largest element stored in the subtree whose root is the middle child of a. The values of L and M assigned to each internal node enable us to start at the root and search for an element in a manner analogous to binary search. Figure 1 provides an illustration.

We wish to add the SUCCESSOR operation to 2-3 trees. Given a leaf x in a 2-3 tree, the SUCCESSOR operation returns the leaf that comes after x in the sorted order that the 2-3 tree determines.

(a) Describe how the SUCCESSOR operation can be implemented on a 2-3 tree in $O(\lg n)$ time, where n is the number of elements stored in the tree.

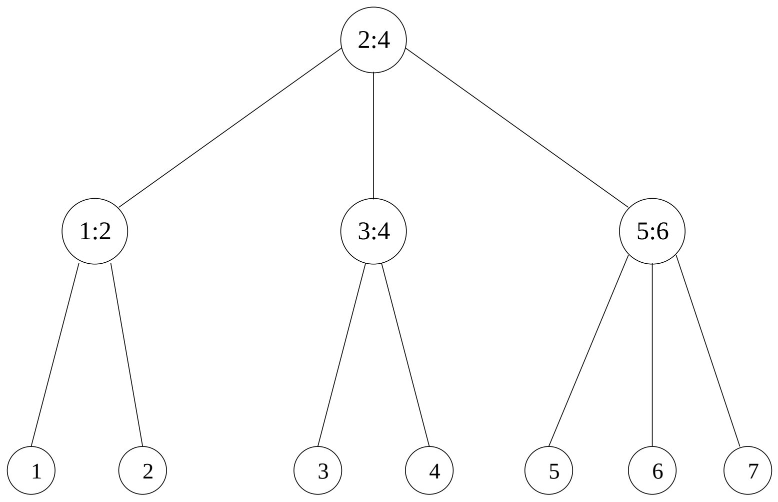

Figure 1: Seven keys stored in a 2-3 tree.

(b) Describe how the 2-3 tree data structure can be modified to support the SUCCESSOR operation in $O(1)$ time. Make sure you can still support INSERT, DELETE, and SEARCH in $O(\lg n)$ time.

(c) Describe how the 2-3 tree data structure can be augmented in $O(n)$ time using $O(1)$ additional space at each node, so that for any $i, 1 \leq i < n$, the i^{th} successor of an element–provided it exists–can be found in $O(\lg n)$ time.

Problem 4-3. For an integer n, a permutation π of size n is a one-to-one function from $\{1, 2, \ldots, n\}$ to $\{1, 2, \ldots, n\}$. Given a sequence of n items $x_1, x_2, \ldots, x_n$, we can *apply* π to that sequence to obtain the permuted sequence $x_{\pi(1)}, x_{\pi(2)}, \ldots, x_{\pi(n)}$. For example, if $\pi(1) = 3$, $\pi(2) = 1$, and $\pi(3) = 2$, then π applied to the sequence M,A,F yields F,M,A.

Let π be a permutation of size n. If $i < j$ and $\pi(i) > \pi(j)$ then the pair $(\pi(i), \pi(j))$ is called an *inversion* of the permutation. A single element out of place may cause many inversions. For example, the permutation

$$\pi(1) = 5, \quad \pi(2) = 1, \quad \pi(3) = 3, \quad \pi(4) = 4, \quad \pi(5) = 2$$

has 6 inversions: $(5, 1)$, $(5, 3)$, $(5, 4)$, $(5, 2)$, $(3, 2)$, and $(4, 2)$.

(a) What permutation of size n has the most inversions? How many does it have?

(b) Give an algorithm that determines the number of inversions in any permutation of size n in $O(n \lg n)$ time. Prove that it is correct and analyze its running time.

Problem 4-4. Skip Lists

A random *skip list* is an alternative data structure to a binary search tree. It has the desirable property that an INSERT or a FIND procedure can be conducted in $O(\lg n)$ time.

A skip list is an ordered linked list of elements: each element is attached to it successor with a pointer. Additionally, some elements have pointers to elements farther down the list, allowing us to *skip* elements when searching for a particular item in the list. An example of a skip list is shown in Figure 2.

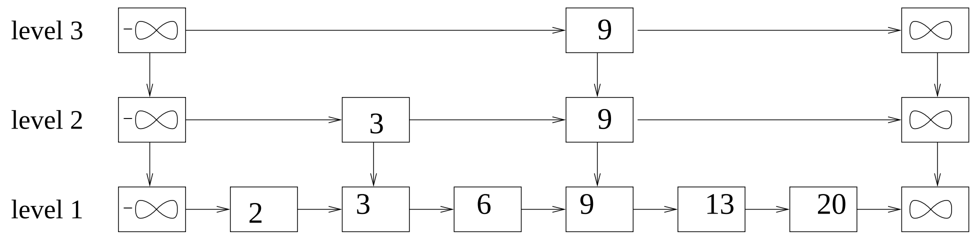

Figure 2: A skip list with height 3.

To create a skip list, we start with an "empty" skip list which contains two elements, ∞ and $-\infty$, with a pointer from ∞ to $-\infty$. Then for each element x that we want to insert into the skip list, we let the height of the element be h with probability $1/2^h$, i.e. we can toss a coin until it comes up heads on trial h. If we determine that the height of x should be k, then we insert x into the skip list and attach it with pointers to its successors at each of the first k levels. (If the height of element x is higher than that of any element already in the skip list, then we increase the height of the ∞ and $-\infty$ elements so that they have the same height as element x.)

(a) Show that if we insert n elements into an initally empty random skip list, the expected space required is $O(n)$. You can assume that an element x uses one unit of space at each level in which it appears and that each pointer uses one unit of space.

(b) Show that if we insert n elements into an initally empty random skip list, the expected number of levels, i.e. the maximum height of an element is at most $\lg n + O(1)$.

(c) Let x_i denote the element in a skip list that has value x at level i. A *parent* of an element x_i in a skip list is the maximum valued element of value at at most x at level $i+1$. For example, in Figure 2, 9_2 is the parent of 9_1, 13_1, and 20_1.

Prove that the expected number of children, i.e. elements on level i with the same parent on level $i+1$, of an element y_{i+1} is $O(1)$.

(d) Suppose we follow the convention that we never insert any element into the skip list with height more than $100\lg n$. In other words, to assign a height to an element x, we flip a coin until it comes up heads on trial h. Then we let the height be $\min\{h, 100\lg n\}$. Use this to determine the expected time for an INSERT procedure.