Homework #6: Randomized Algorithms and Hashing
Due Date: Tuesday, April 7, 1998 (5:00pm)

Collaboration Policy

Work entirely on your own for all of these problems.

Reading

You should read Ch. 12.1-12.3 of CLR regarding hashing. You may review Chapter 13.4 of CLR, regarding the discussion of randomly built binary search trees.

Practice

Recall, these exercises are purely for your own practice. You need only turn in the official problems.

- Do CLR 12.2-1.
- Do CLR 12.2-6.
- Do CLR 12.3-5.

Problems

1. CLR 12-5 (40 points)
   Let $\mathcal{H} = \{h\}$ be a class of hash functions in which each h maps the universe U of keys to $\{0, 1, \ldots, m-1\}$. We say that $\mathcal{H}$ is *k-universal* if, for every fixed sequence of k distinct keys $\langle x_1, x_2, \ldots, x_k \rangle$ and for any h chosen at random from $\mathcal{H}$, the sequence $\langle h(x_1), h(x_2), \ldots, h(x_k) \rangle$ is equally likely to be any of the m^k sequences of length k with elements drawn from $\{0, 1, \ldots, m-1\}$.

   a. Show that if $\mathcal{H}$ is 2-universal, then it is universal.
   b. Show that the class $\mathcal{H}$ defined in Section 12.3.3 of CLR is not 2-universal.
   c. Show that if we modify the definition of $\mathcal{H}$ in Section 12.3.3 of CLR so that each function also contains a constant term b also chosen uniformly at random from $\{0, 1, \ldots, m-1\}$, that is, if we replace $h(x)$ with
   $$h_{a,b}(x) = \sum_{i=0}^{r} a_i x_i + b \mod m,$$
   then $\mathcal{H}$ is 2-universal.

2. (60 points)
In class last week (and in Chapter 13.4 of the text), we saw that if you take a list of n items with distinct keys and insert them into a standard binary search tree using a *random* order, the expected height of the tree is $O(\lg n)$. We can use this idea to get yet another version of balanced trees. The difficulty, however, is that in a dynamic setting you cannot really assume that the items come in a random order, and you cannot afford to wait until all items arrive before building your data structure. In this problem, we will design a new method, which has expected $O(\lg n)$ time for any operation in a sequence of `Insert`'s, `Delete`'s, and `Search`'s.

Assume that every object x has two associated values, $key[x]$ and $priority[x]$. We define a new data structure called a *treap*, which is a binary tree with objects stored at internal nodes, satisfying the following two properties. (we assume that keys are distinct, and that priorities are distinct)

- If node y is the left (respectively right) child of node x, then $key[y] < key[x]$ (respectively $key[y] > key[x]$).
- If node y is the child of node x, then $priority[y] < priority[x]$.

In other words, this tree happens to be both a binary search tree on the keys, while at the same time a heap on the priorities (and hence the name treap = tree + heap).

a. Assume that we have n objects, and that for each object x, we fix the value of $key[x]$ and $priority[x]$ in some way, so that all values are distinct. Prove that there must exist a *unique* treap containing those objects. (*Hint:* make sure that you prove both the existence and uniqueness.)

We would like to maintain a treap dynamically through a sequence of calls to `Insert`, `Delete`, and `Search`, as defined below. In the following three parts, we will ask you how to implement each of these operations while maintaining the properties of a valid treap. For each operation, you should briefly justify the correctness of your implementation as well as the bound on its running time. In describing your implementations, you may feel free to refer to descriptions of basic operations on binary trees as described in Chapter 13 of the text.

b. Given an existing treap T, explain how to implement $\texttt{Search}(T, k)$, which returns an object with key k if one exists in T. The running time of your operation should be $O(height(T))$.

c. Given an existing treap T, explain how to implement `Insert`(T, x), where the values of $key[x]$ and $priority[x]$ are given as input. Your operation should result in a valid treap T' for the new set of objects, and the running time should be $O(height(T))$.

d. Given an existing treap T, explain how to implement `Delete`(T, x), where you may assume that you are given a pointer to the location of object x in T. Your operation should result in a valid treap T' for the new set of objects, and the running time should be $O(height(T))$.

Up until now, we have assumed that an object given to `Insert` arrives with both a key and a priority. If we want to maintain a set of objects for which only the key is defined, we are free to choose any values we want for priorities, and the above routines will be correct. For the remainder of the problem, we assume that when an object x with $key[x]$ is inserted, we set the priority of that object uniformly at random to a real number in the range $[0, 1]$. (The probability that two different objects choose the identical priority is equivalently zero.)

e. Assume that some sequence of operations has resulted in a treap containing exactly n objects. Argue that no matter what sequence of operations really took place, the current treap must look exactly like the standard binary tree which would have resulted if objects were inserted in order of decreasing priorities.

f. Conclude that the expected running time of any given operation is $O(\lg n)$, where n is the number of objects in the data structure before the operation begins.

Extra Credit

This week, we will save the algebra for those who really want the work. Assuming uniform hashing we were able to bound, without too much trouble, the expected size of a particular bucket in a hashing scheme to be $O(1)$. Of course, there will be some variance in the sizes of various buckets, and so it is not necessarily the case that the maximum sized bucket will be $O(1)$.

Suppose that we have a hash table with n slots, with collisions resolved by chaining, and suppose that n keys are inserted into the table. Each key is equally likely to be hashed to each slot. Let M be the maximum number of keys in any slot after all the keys have been inserted. Your mission is to prove an $O(\lg n / \lg \lg n)$ upper bound on $E[M]$, the expected value of M.

a. Argue that the probability Q_k that k keys hash to a particular slot is given by

$$Q_k = \left(\frac{1}{n}\right)^k \left(1 - \frac{1}{n}\right)^{n-k} \binom{n}{k}.$$

b. Let P_k be the probability that $M = k$, that is, the probability that the slot containing the most keys contains k keys. Show that $P_k \le nQ_k$.

c. Use Stirling's approximation, equation (2.11 of CLR), to show that $Q_k < e^k/k^k$.

d. Show that there exists a constant $c > 1$ such that $Q_{k_0} < 1/n^3$ for $k_0 = c\lg n/\lg\lg n$. Conclude that $P_k < 1/n^2$ for $k \ge k_0 = c\lg n/\lg\lg n$.

e. Argue that

$$E[M] \le Pr\left\{M > \frac{c\lg n}{\lg\lg n}\right\} \cdot n + Pr\left\{M \le \frac{c\lg n}{\lg\lg n}\right\} \cdot \frac{c\lg n}{\lg\lg n}.$$

Conclude that $E[M] = O(\lg n/\lg\lg n)$.