15-451 Algorithms, Fall 2014

Homework # 3 **Due: September 18, 2014**

Your homework should be submitted as a single PDF file using `autolab`. You are allowed to work in groups of 2-3, but you must not share written work, and must write your solutions yourself. **Please cite your sources, and your collaborators; mention explicitly if you did not discuss problems with others. Else we will deduct 20 points.** Solve problems #1 and #2.

Problem #H is a "honors" problem. See previous writeups for rules about such problems.

(40 pts) 1. **Balanced Splay Trees**

A drawback of the splaying algorithm compared to other balanced BST algorithms is that the tree can be very imbalanced. This is problematic in real-time applications where you want a guaranteed quick response to a query. In this problem we see a way to address this drawback.

First of all, it's possible to maintain a `depth` field in each node of the splay tree. The depth of a null tree is 0. The depth of a node is the maximum of the depths of its two children plus 1. (So, for example, a tree with one node has depth 1.) Similarly, you can keep a `size` field in each node, which stores the number of nodes in the subtree rooted at this node. These allow us to determine the depth and size of the entire tree in $O(1)$ time. The `depth` field also allows us to navigate from the root to a deepest node in the tree (keep following the link to the deeper of the two children) in $O(\text{depth})$ time.

Here is the **deep splaying heuristic**. Whenever we're told to access a node in the tree, we first check if the current tree is "off-balance". What's that? A tree of n nodes with depth d is *on-balance* iff:

$$d < 6 \log_2 n + 3;$$

and a tree that is not on-balance is *off-balance*.

If the current tree is off-balance, we repeatedly splay the deepest node in the tree (this operation is called a **deep splay**) until the tree is on-balance. This ends the deep-splaying part for this access. Finally we splay the requested node to the root, just like in the usual splay tree.

(Q) Starting from a prefectly balanced tree of n nodes, a user of our API does a sequence of s accesses to nodes in our tree. Let t be the number of deep splay operations done. Prove that $t \leq s$.

You will need the following stronger version of the access lemma (which you can use without proving). This version counts rotations rather than splay steps. One small change from the usual analysis: define the rank of node x as $\log_2 s(x)$ instead of $\lfloor \log_2 s(x) \rfloor$. The potential is again the sum of ranks of all nodes.

Strong Access Lemma: The amortized *number of rotations* done when splaying a node x of rank $r(x)$ in a tree rooted at node y of rank $r(y)$ is at most $3(r(y) - r(x)) + 1$.

(60 pts) 2. **Universal Hashing**

In this problem we consider hash functions for strings. Let $M \geq 2$ be the size of the hash table. The universe U is the set of all n character strings from the alphabet $\Sigma = [0, 255]$; assume $n \geq 10$. Hence $|U| = 256^n$. Let $S = s_1 s_2 \ldots s_n$ denote a string in U, so each number s_i is a number between 0 and 255.

(Note: the answers below are the same for all values of $M \geq 2$ and $n \geq 10$. As usual, proofs are required for all parts.)

We say that a hash family $\mathcal{H}$ is *p-universal* if, for every fixed sequence of p distinct keys $\langle x_1, x_2, \ldots, x_p \rangle$ and for any h chosen at random from $\mathcal{H}$, the sequence $\langle h(x_1), h(x_2), \ldots, h(x_p) \rangle$ is equally likely to be any of the M^p sequences of length p with elements drawn from $\{0, 1, \ldots, M-1\}$.

Note that 2-universal implies universal (the converse is not true), and that p-universal implies $(p-1)$-universal. (Check these claims for yourselves!)

(a) Consider the following type of hash function. For a table T of 256 random numbers each in the range $[0, M-1]$, define the hash function is defined as:

$$h_T(S) = (\sum_{i=1}^{n} i * T[s_i]) \bmod M$$

(So there are M^{256} possible tables, each one gives a hash function in this family.)

i. Is this a universal hash family?

ii. Is this family p-universal for any value of $p > 1$?

(b) Now consider a different hash family. This time our table is two dimensional. $T[p][c]$ is a table of random numbers in the range $[0, M-1]$, where $c \in [0, 255]$, and $p \in [1, n]$. (So now there are M^{256n} such tables.) Our hash function is:

$$h'_T(S) = (\sum_{i=1}^{n} T[i][s_i]) \bmod M$$

i. Is this a universal hash family?

ii. What is the maximum value of p for which this hash family is p-universal?

(0 pts) H. **A Zero-counting Data Structure**

We want to represent an array of values $x_0, x_1, \ldots, x_{n-1}$ (all initially zero) under the following operations:

- `inc-range(i,j)`: Increment all the variables $x_i \ldots x_j$ by 1.
- `dec-range(i,j)`: Decrement all the variables $x_i \ldots x_j$ by 1. (It's guaranteed that none of the values will ever go negative.
- `zero-count()`: Count the number of variables whose current value is 0.

All operations should take (possibly amortized) $O(\log n)$ time. Explain how to do this.