CS130N Problem set 5: Dictionaries

March 3, 2001

1. A certain Professor Amongus claims that the order in which a a fixed set of elements is inserted in to a binary search tree does not matter - the same tree results every time. Give a small counter example to prove Professor Amongus wrong.

2. Professor Amongus now claims that he has a patch to his claim from the previous exercise, namely, that the order in which a fixed set of elements is inserted in to a AVL tree does not matter - the same AVL tree results every time. Give a small counter example to prove that Professor Amongus is still wrong.

3. Explain why performing a rotation in an n-node binary tree represented as a sequence takes $\Omega(n)$ time.

4. Write pseudo-codes for deletion of an item from a Dictionary when the Dictionary is implemented as

   (a) Binary search tree

   (b) AVL tree

   (c) Red-Black tree

   (d) 2-3-4 tree

   (e) skip list

   (f) Open address hashing with probing sequence computed by double hashing.

5. Draw the 11 item hash table that results from using the hash function

   $$h(i) = (2i + 5) mod 11$$

   to hash the keys 12,44,13,88,23,94,11,39,20,16 and 5, assuming that collisions are handled by chaining.

6. What is the result of the previous exercise, assuming collisions are handled by linear probing? What if we use quadratic probing? What about double hashing?

7. Suppose we are given two ordered dictionaries S and T, each with n items, and that S and T are implemented by array based ordered sequences. Describe an $O(\log^2 n)$ time algorithm for finding the $k-th$ smallest key in the union of the keys from S and T (assuming no duplicates).

8. Give an $O(\log n)$ solution to the previous problem.

9. Draw an example of an AVL tree such that a single remove operation could require $\Theta(\log n)$ rotations from a leaf to the root.

10. Show that any n node binary tree can be converted to any other n node binary tree using $O(n)$ rotations.

11. Let D be an ordered dictionary with n items implemented by means of an AVL tree. Show how to implement

    (a) the operation $findAllInRange(k_1, k_2)$ (which returns an enumeration of all the elements in the D with key k such that $k_1 \leq k \leq k_2$) on D in time $O(\log n + s)$, where s is the size of the enumeration returned.

    (b) the operation $countAllInRange(k_1, k_2)$ (which returns the number of items in D with key k such that$k_1 \leq k \leq k_2$). Note that this returns a single integer. You will need to extend the AVL tree data structure.

12. The universal hashing implementation requires that we find a prime number between M and $2M$. Give a method for finding such a prime using the **sieve algorithm**. In this algorithm we allocate a $2M$ cell boolean array A such that the cell i is associated with the integer i. We then initialize the array cells to be *true* and we *mark off* all the cells that are multiples of 2,3,5,7, and so on. This process can stop after it reaches a number larger that $\sqrt{2M}$. (Hint: Consider a bootstrapping method for finding the primes up to $\sqrt{2M}$.)

13. Discuss the following statement with your tutor: "In universal hashing the average is taken over the functions in the class H, i.e., the randomization is done by the algorithm itself and not by the user. For

every fixed set $S \subset U$, a random function $h \in H$ will distribute S fairly evenly over the hash table, i.e., almost all function $h \in H$ work well for S and only very few will give us bad performance on any fixed set S".

14. Discuss with your tutor the structural correspondence between red-black trees and 2-3-4 trees.

15. Let T and U be 2-3-4 trees storing n and m items, respectively, such that all items in T have keys less than the keys of all items in U. Describe an $O(\log n + \log m)$ algorithm for joining T and U into a single tree that stores all items in T and U (destroying the old versions of T and U).

16. Let T be a red-black tree storing n items, and let k be the key of n item in T. SHow how to construct from T in $O(\log n)$ time two red-black trees T' and T'' such that T' contains all the keys of T less than k, and T'' contains all the keys of T greater than k. This operation destroys T.

17. Study Sandeep Sen's analysis of skip-lists from the document http://www.cse.iitd.ernet.in/∼ssen/papers/ddss.ps.gz (page 10).