CSE 214 - Data Structures
Fall 1997
Program #5

Due: Friday, December 12, 1997

In many if not most information retrieval problems, the probability of performing a query is not uniform over all search keys. In this assignment, we will consider both random search trees and splay trees as data structures for maintaining password files.

Each time you log on the computer and enter a password, the operating system has to look you up in a password file to check whether your password is correct. Since new users are always being added to the system, we want our password file to support both insertion and query operations. A random or balanced search tree would work, but it would fail to exploit either locality or frequency of reference. Certain people log on far more often than others, and we would want them up near the top of the search tree. Most people use computers in spurts, ie. right before an assignment is due, so we would expect logins to also occur in clusters. Thus we want people who logged on recently to also, appear high in the tree. Splay trees can take advantage of both these phenomena.

The file ~cse214/program5/logons contains the names of well over 10,000 consecutive logins onto sbstaff2, one of the Department of Computer Science's research machines. These logins were made by approximately 250 different users, so the average user logged on over forty times in this interval. However, the login frequency is very non-uniform.

Starting from an empty random search tree and an empty splay tree, your program will read user ids one at a time, and proceed to search for each user in both of the two trees. If the user is found, you should add one to the count of how often the user has signed on. If the user is not found, you should insert him/her in the tree, and then continue. At the end, for each of the fifty most frequently accessed names in the data file, you should print out their depth in both trees along with their frequency of access. You should also print the total number of element comparisons used by each data structure in the course of the program, and the total number of rotations performed by the splay tree.

A few hints are in order concerning splay trees. First, observe that the type declaration for nodes of both random and splay trees are identical, so you can reuse data types and procedures for printing and debugging. On an insertion into a splay tree, the new key is inserted into its proper position and then immediately splayed to the root. On a query in a splay tree, after finding the element it is splayed to the root. This ensures that we avoid building very skewed trees.

When counting comparisons, observe that no key comparisons are made during the splay step, since we are only comparing against the search path down the root, which are left and right turns, instead of against any actual elements. To decide which series of rotations to apply, I recommend trying to develop a recursive algorithm. Realize that in a splay tree, double rotations happen every second level, meaning that the height of the key decreases by two each step except possibly the last one. Comparing a splay tree implementation to red-black or AVL trees, we see that instead of passing up the change in height after insertion at a leaf, we would be interested in knowing whether we must traverse LL, LR, RL, or RR to get to the search key.

Challenge Problems

Each assignment will contain one or more extra credit problems for students who want a challenge, or to pursue the subject in greater depth. NOTE THAT THESE PROBLEMS ARE NOT REQUIRED.

(1) Once you have implemented insertion, you should consider deletion. The algorithm for deletion from a random tree was discussed in class. If you are interested in a splay tree deletion algorithm, come talk to me and I will provide a reference.

Rules of the Game

(1) This assignment is intended to compare the performance of random binary search trees and splay trees. The actual program should not be very long, but implementing splay trees will be quite tricky, especially if you are uncomfortable with pointers and recursion.

(2) Splay trees are no covered in this edition of Sedgewick's book, but copies of the relevant pages from his new *Algorithms in C* will be handed out. Another accessible description of the splay operation appears in Kingston's *Algorithms and Data Structures*.

(3) I warn in advance that it is hopeless to start writing the splay tree section of this assignment without a good understanding of the algorithm, and how rotations propagate up the tree in red-black or AVL trees. Start early because you cannot get that understanding overnight. I recommend doing the random tree implementation first since it is essentially given in the text.

(4) Debugging a tree program is difficult without proper tools, because it is hard to see exactly what the tree looks like at a given stage. I *STRONGLY* recommend that before you even write the actual splay routines, you write a debug function which prints out the state of the tree. My recommendation is that this function should do an in-order traversal of the tree to print the keys in sorted order, and for each key print the search path from the root to the node. By search path, I mean the order of left and right steps to get to the node, for example LLRRL.

(5) You will want to write separate procedures which do left and right rotation, as well as the left and right double rotation specified for splay trees. Note that the double rotation is significantly different than for red-black / AVL trees, and depends upon the search path to the node.

(6) To compute which are the fifty most frequently occurring users, I would recommend sorting the users by frequency using the Quicksort you wrote for the fourth assignment. How easy do you think it will be to reuse your previously written code?

(7) This assignment is an excellent opportunity to experiment with object-oriented features of Modula-3 such as inheritance, since splay trees can be derived from random trees by overloading insert and search.

(8) This particular application of splay trees to login access is a little contrived, since login checking represents such a small fraction of what an operating system must do that it is not necessary to have such a sophisticated data structure. However, the data does illustrate the locality of reference behavior which is typical of many different applications.

(9) Don't forget to write out and sign the pledge - "On my honor as a student I have neither given nor received aid on this programrq. Your assignment will not be graded without it.