CSE 373 Spring 2001

Programming Assignment # 1

Comparing Unbalanced Binary Search Trees with Splay Trees

(Due Date: Friday, April 27 at the *beginning* of class)

The goals of this assignment are: (1) to become proficient in programming search trees, and (2) to understand how splaying improves efficiency in the long run for sequences of operations as compared to unbalanced binary search trees. Remember that in splay trees, every Find, Insert, and Delete operation causes rotations along the access path. In the case of a Find or Insert of an item X, we splay along the path from X (or a leaf if X is not found) to the root, ending up with X (or the leaf) as the root. In the case of a Delete for an item X, we splay from X to the root and delete X at the root (see page 129 in the text for more details).

Write two programs, one that implements unbalanced binary search trees and one that implements splay trees. In addition to the standard operations, you will also need to write code for estimating performance based on the number of comparisons (made using operations < or >) that each tree makes in processing the same test data. The code for unbalanced binary search trees is available from the web sites:
http://www.cs.fiu.edu/~weiss/dsaa_c2e/files.html (C code)
http://www.cs.fiu.edu/~weiss/dsaa_c++/code/ (C++ code)
You may also use the code for rotations from the AVL tree implementation in the above web sites for your 6 splay operations (2 zigs, 2 zig-zigs, and 2 zig-zags). Do not use the code for top-down splay trees (we are interested in bottom-up splaying). Use a parent pointer at each node to speed-up the bottom-up splaying along the access path.

The data items will be integers. The input data will be stored in a text file in the following format, one command per line:

```
e          # make the tree empty
i [int]    # insert [int] into the tree; ignore if already there
d [int]    # delete [int] from the tree; print error if not there
f [int]    # find [int] in the tree; print error if not there
p          # print tree using in-order traversal(left subtree, node, right subtree)
t          # print tree using preorder traversal (node, then left and right subtrees)
h          # print height of the tree
c          # print counter of number of comparisons made so far
z          # zero counter of number of comparisons made so far
```

See http://www.cs.washington.edu/373/CurrentQtr/373assignments.html for examples of test data files. For preorder traversal, print the nodes as in Figure 4.7 of the text.

Hand in a printout of your source code and output for the example test files. Also e-mail your program files (with "373 Programming Assignment 1" in the subject) to Charles and Jiwon before class on Friday, April 27.