Data Structures and Algorithms

Assignment #4

Due: Paper Assignment Wednesday May 3rd in class
Due: Program Assignment Monday May 8th 11:00 am

This assignment will deal mostly with balanced BST's. In the programming part you will implement methods for splay trees. In the paper part, you will have problems on heaps (not BSTs, I know!), AVL trees, and splay trees. It is recommended that you do the splay tree paper assignment (problem 7) before the programming part.

Paper Assignment

(Heaps)
1. Problems R-8.16 and R-8.17. A 3-node heap should be sufficient for both cases.

2. Problem C-8.14 (in pseudo-code)

(AVL Trees)
3. Problems R-10.6 and R-10.7. Two sequences of the same 4 keys should be sufficient.

4. Given an AVL tree, prove that at most one rotation (either single or double) is required to balance the tree for the Insert operation.

5. Given an AVL tree, show by example that more than one rotation might be necessary to keep the tree balanced after the Delete operation.

(BST, AVL and Splay trees)
6. Problem R-10.20 (a), (d) and (e)

(Splay trees)
7. Problem R-10.21 (The final answer to each part will be posted soon; you are required to show all steps leading to the final answers).

Programming Assignment

In this assignment you have to implement some basic Java methods for *splay trees*. The application that you have to program is a little contrived. Splay trees are more advantageous when there are a large number of nodes and the 80-20 rule (80% of the accesses are for 20% of the entries) is followed. However, so that you don't have to deal with huge data sets, we are going to limit the test file sizes.

For this application, each node in the tree has:

- A *key* (in this assignment the key will be a string of 7 characters; each character is a digit but the point is that the key is "comparable to" other keys)
- A *value* (this will be a small positive integer as defined below)
- Three links, *left*, *right* and *parent*

In the first part of the assignment, a splay tree will be build from a file **enroll.txt** that contains student numbers. This file is the concatenation of enrollments in several courses (this is taken from some old

course lists of mine, somewhat changed and all anonymity has been preserved). Each student number is a string of 7 digits (e.g., 9703456), one per line of the file. The same student number can appear several times in the file if s/he has been enrolled in several courses. The file terminates with a student number of 0000000 that should not be entered in the splay tree.

Besides the *size()* and *isEmpty()* methods, in order to build the splay tree you should implement:

- *find(k)* which looks for the node in the tree with key k and if found returns the value x found in the node. If *find(k)* fails, it should return the value 0. In the case of a successful *find*, splaying should occur from the found node up to the root. If *find* fails, splaying should start at the last node visited before the failure is detected.
- *Insert(k)* If there is no node in the tree with the key k, this node should be inserted and the *value* field should be set to 1. Splaying should occur starting at the inserted node. If there is already a node in the tree with the key k, the *value* field of that node is incremented by 1 and splaying occurs as for a successful *find*.
- *delete(k)* (this will be useful for the second part of the assignment). If there is no node with key k, the value 0 is returned and splaying proceeds like in the case of a failed *find*. If the node is in the tree, its value is returned, it is deleted and splaying starts from the parent of the "deleted" node. The meaning of "deleted" node is unambiguous when the node with key k has no children or only 1 child. If it has 2 children, the "deleted" node is its inorder successor (see slide 18 in Lecture on Splay trees).
- *print()* prints the contents of the tree, i.e., pairs (key,value) in *inorder* based on the keys. You should print 10 pairs/line.

When you program the above methods, draw diagrams to be sure that you are writing the code you intend to write. Use helper methods, for example for "zig-zig" and "zig-zag".

In summary: build the tree according to the above specifications using **enroll.txt** as your input file and output using *print* as indicated above. Note that you are allowed (encouraged) to use code from Assignment #3, suitably modified to take into account the parent link.

For part two of this assignment you are given a second file, **query.txt** with the same format as **enroll.txt**. You can assume that the number of entries in the **query.txt** file is small, say less than 15.

Your task is to build a list (in the ADT sense, the implementation is your choice) of the student numbers in the **query.txt** file sorted by the values (number of courses) they have taken. Of course this information is in the splay tree. If a student number exists in the **query.txt** file but not in the splay tree (i.e., not in the **enroll.txt** file), you should still enter it in the list but with the value 0. When a student number is found in the splay tree, it should be deleted from the tree following the splaying instructions given above.

When all entries in **query.txt** have been processed you should print the sorted list of (student number, courses) in descending order of the number of courses. In case of ties, the student with the smaller student number should be listed first. (Here, code from Assignment #1 suitably modified could be useful).

A template for this assignment, as well as the **enroll.txt** and *query.txt* files will be posted before the week-end.