

CMPSC 130A: Spring 2009

Programming Assignment 2

Assigned: May 14, 2009

Due: May 28, 2009 (Thur midnight)

Your second programming assignment is to implement the **splay trees**. Recall that splay trees are balanced search trees in which no explicit balance condition is maintained; instead, a simple restructuring heuristic is used after each access.

Splaying

Specifically, whenever a node x is accessed (either by search, or insert, or delete), the following **splay operation** is repeated at x until it becomes the root. In the following, $p(x)$ denotes the parent of node x .

- **Case 1:** if $p(x)$ is the root, rotate the edge between x and $p(x)$; and terminate.
- **Case 2:** [Two Single Rotations] if $p(x)$ is not the root and x and $p(x)$ are both left or both right children, first rotate the edge joining $p(x)$ with x 's grandparent $g(x)$, and then rotate the edge joining x with $p(x)$.
- **Case 3:** [Double Rotation] if $p(x)$ is not the root and x is a left child and $p(x)$ is a right child, or vice versa, then rotate the edge joining x with $p(x)$ and then rotate the edge joining x with its new parent.

User Operations

The splay trees support the following **user operations**:

- **access (i,t):** If item i is in tree t , **return the depth at which it was found**; recall that after the splay, i move to the root, so this depth must be calculated before the splay operation. If i is not in the tree, return an error message.
- **insert (i,t):** Insert item i in tree t , assuming it is not there already; if i is already in the tree, return an error message.
- **delete (i,t):** Delete i from t , assuming it is present; if i is not in the tree, return an error message.
- **join (t_1, t_2):** This operation assumes that all items in t_1 are less than all those in t_2 . This combines trees t_1 and t_2 into a single tree and returns the resulting tree.
- **split (i,t):** Construct and return two trees: t_1 , which contains all items less than or equal to i , and t_2 , which contains all items greater than i . This operation destroys t .

Implementation

Internally, you will implement these operations as follows.

- To perform **access(i,t)**, we search down from the root, looking for i . If search reaches a node x containing i , **we splay at** x and return the pointer to x . If search reaches a **null node**, we splay the last non-null node, and return a null pointer.
- To perform **join(t_1, t_2)**, we first access the largest item in t_1 . Suppose this item is i . After the access, i is at the root of t_1 . Because i is the largest item in t_1 , the root must have a null right child. Simply make t_2 's root to be the right child of t_1 . Return the resulting tree.
- To perform **split(i,t)**, first perform **access(i,t)**. If root contains an item greater than i , then break the left child link from the root, and return the two subtrees. Otherwise, break the right child link from the root, and return the two subtrees.
- To do **insert(i,t)**, perform **split(i,t)**. Replace t with a new tree consisting of a new root node containing i , whose left and right subtrees are t_1 and t_2 returned by the split.
- To do **delete(i,t)**, perform **access(i,t)**, and then replace t by the **join** of its left and right subtrees.

How your program will be graded.

First, make sure that the main of your program clearly has your name at the top. **Second**, include a clearly written README file telling the TA how to compile and/or run your program. **Finally**, you must do the following:

1. (60 pts) The TA will run your program to test that access, insert, and delete operations work properly. In order to help him do that, when started, **your program shall initially build a splay tree on keys 1, 2, ..., 89 (inserted in that order)**. It shall then wait for manual commands from the TA, who will try the following operations:
 - (a) **access i** : your program shall determine if i is in the tree, and report the depth of the last node searched.
 - (b) **insert i** : your program shall insert i .
 - (c) **delete i** : your program shall delete i from the tree.

In case of error, output -1. For successful insertion/deletion, output 0.

2. (20 pts) Show a printout of your tree after keys **1, 2, 3, 4, 5, 6, 7, 8, 9** have been inserted in this order You can submit this printout in any readable form: ascii, pdf, doc etc. **Label this file Output1.xxx**, so the TA can easily spot it. (.xxx is the suffix identifying the type of the file: .txt, .doc, .pdf etc.)

3. (20 pts) Run your program and build a splay tree on keys 1, 2, ..., 1000 (inserted in that order). Calculate the **average** depth of a key in your tree, as well as the **maximum** depth of any key in the tree.

Now perform 100 accesses in multiples of 10. That is, perform access 10, access 20, access 30, ..., access 1000. Calculate the **average** and **maximum** depths in the tree again. How different are they? **Print out** both these sets of values in a file **Output2.txt**.