CMPSC 130A: Winter 2019

Programming Assignment 2

Assigned: Feb 14 **Due: Feb 28 (11:59 PM)**

This assignment asks you to implement **splay trees** for integer-valued keys. These are balanced search trees in which no explicit balance condition is maintained; instead, a simple restructuring heuristic is used after each access. The following description reviews the main technical steps of splay trees.

Splaying

When a node x is accessed, 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

Splay trees support the following **user operations**, and your program is expected to have these functionalities:

- **find i**: If i is in the tree, print "**item i found**." If i is not found, print error message "**item i not found**".
- **insert i**: If i is already in the tree, do not insert it, and print an error message "**item i not inserted; already present**". Otherwise, insert i and print message "**item i inserted**".
- **delete i**: If i is in the tree, delete it and print "**item i deleted**". Otherwise print an error message "**item i not deleted; not present**".

Implementation

To support these operations, you need to implement the following splay tree functions:

- **access (i)** is the basic routine of splay trees. It searches down from the root looking for i. If the search reaches a node x containing i, then it **splays at** x and return the pointer to x. If search reaches a **null node** (which means the key is not found in the tree), we splay the last non-null node, and return a null pointer.

- **join (t_1, t_2)** combines trees t_1 and t_2 into a single tree and returns the resulting tree, and assumes that all keys in t_1 are less than all those in t_2. To perform **join(t_1, t_2)**, we first access the largest key in t_1. Suppose this key is i. After the access, i is at the root of t_1. Because i is the largest key 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.
- **split (i,t)** splits the tree t at key i. To implement this, first perform "access i" in t. After the access, if the root contains an key 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.

Then, you will be ready to implement the basic functionalities as follows:

- **find (i)**: call "access i" and return the appropriate message.
- **insert (i)** in tree t: perform **split(i,t)**, and 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. Return appropriate message depending on whether i was successfully inserted or not.
- **delete (i)**: perform "access i" in t, and then replace t by the **join(t_1, t_2)** of its left (t_1) and right (t_2) subtrees. Return appropriate message depending on whether i was deleted or not.

How your program will be graded.

We will run your program to test that *find, insert, and delete* operations work properly. (The functions split and join will not be tested explicitly, but they must work correctly for insert and delete to work.) In addition, you are expected to implement the following *print* function:

- **print:** Your program shall print the splay tree in Breadth-First Search (BFS) Order, from top to bottom level by level. The keys at the same level should be printed in one comma separated line. (The last command of the input file is always print, but it can be called at other times during the testing as well.)

Testing

We will test your programs as a black box using autograder, so make sure that your output adheres to the specified format.

You are required to submit a makefile so that the command **make** compiles your code and produces an executable called **prog2**. We should be able to run your program as ./prog2 < input.txt. Therefore make sure you read your input file from standard input (from **std::cin** in C++). The input file will have the following structure:

- The first line will be an integer N, the number of operations (including print) that follow.
- The next N lines will list the operations, one operation per line. The last statement will always be print.

`The following is an example:`

```
10
```

```
insert 1
find 1
insert 3
insert 5
insert 2
insert 6
insert 6
delete 7
delete 1
print
```

The correct output for this input is:

```
item 1 inserted
item 1 found
item 3 inserted
item 5 inserted
item 2 inserted
item 6 inserted
item 6 not inserted; already present
item 7 not deleted; not present
item 1 deleted
5
2, 6
3
```

Finally, the state of the tree after each operation is like this (the left of the arrow is the operation, and the right of the arrow is the BFS of the tree, printed in the same line):

```
insert 1 -> 1
find   1 -> 1
insert 3 -> 3 1
insert 5 -> 5 3 1
insert 2 -> 2 1 3 5
insert 6 -> 6 5 3 2 1
insert 6 -> 6 5 3 2 1
delete 7 -> 6 5 3 2 1
delete 1 -> 5 2 6 3
```

Please implement the splay tree following the instructions in this assignment, so that the results are consistent, and the auto-grader can correctly evaluate your programs.

Submission Instructions

1. Include a **README** file, with the following information in the given format. (1) Please state the programming language you use in your readme file. (2) CSIL login : [Your username here], UCSB Email : [Your ID]@umail.ucsb.edu This is needed to map your CSIL usernames to your records.

2. The recommended programming language is C++, however you may write your program in other languages. Always make sure to include a makefile, so that running make compiles your source files and generates the executable prog2. **Therefore make sure that your code compiles and runs without problems in CSIL computers.**

3. In order to electronically turnin your project files, use the following command: turnin prog2@cs130a LIST where LIST is the list of all your files, i.e., your makefile, all your source and header files. (The turnin program is located in /usr/bin/turnin on csil.cs.ucsb.edu only.) If you turnin more than once then the version that will be graded is the last version you turned in before the deadline.