Course: CpSc 212: Algorithms and Data Structures Spring 2014
Lab 06: Splay Trees

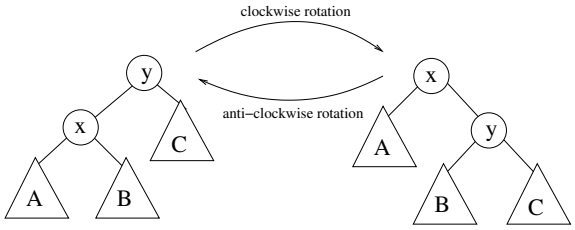
1 The Splay Tree

Splay trees provide a very elegant way to balance a binary search tree. In this lab, you will be continuing your binary search tree from lab 05 and balance it as a splay tree.

All nodes should still satisfy the binary search tree property: An element with key k is at the root. All elements lesser and greater than k are in the left and right subtrees of the root respectively. The splay tree will continue to represent a set data structure. Therefore, no duplicate elements will be present in the splay tree.

1.1 The Splay Operation

All balancing mechanisms in a binary search tree are based on edge rotations. A clockwise rotation of a node x with its parent y is shown in the diagram below.

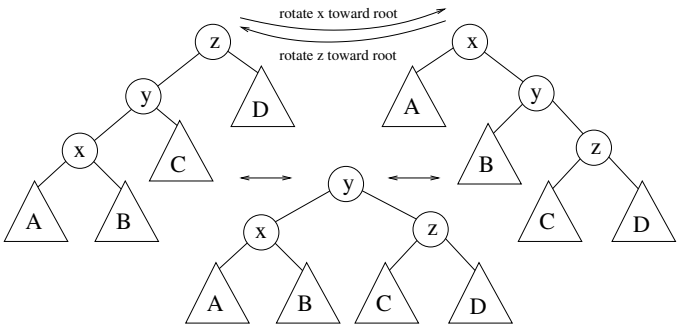


All operations in a splay tree are based on the splay operation. This operation accepts a node (say x) as input and splays it all the way up to the root based on double rotations. If x is one level below the root, then x is rotated with its root. Otherwise, if x is in-line with its parent and grand parent, then the parent node is rotated with the grad parent first, and then x is rotated with its parent. Else, x is rotated with its parent twice. These double rotations continue until x is the root of the tree. The following picture summarizes the in-line double rotations. Out of line double rotations are just two single rotations.

The splay operation takes $O(\log n)$ amortized time. This means a sequence of k splay operations take $O(k \log n)$ time in the worst case.

1.2 Insert and Find

In a splay tree, every time an element is accessed, it is splayed to the root. Therefore, after an element is found or inserted during operations *find* and *insert* respectively, it is splayed to the root. Both the find and insert operations take $O(\log n)$ amortized time.



1.3 Downloading and Extraction

Please first download the *lab06.tar.gz* file from the following webpage:

`http://people.cs.clemson.edu/~rmohan/course/cpsc2120-sp14/`

Next, please extract the files using the following command.

```
tar -zxvf lab06.tar.gz
```

This command will extract all the files in the directory named *lab06*. Please change your directory using the command:

```
cd lab06
```

In this lab, we will be working in two folders *bst* and *list*. You can change to any of these directories using the *cd* command. For instance,

```
cd bst
```

Whenever you want to move to your parent directory, please type the command:

```
cd ..
```

1.4 Program specifications

1.4.1 Rotate and Splay

Please open *intset.h* to see the specifications of the IntSet class. This is the same class as lab 05 with two added functions *rotate* and *splay*. The rotate function takes as input a node x and rotates it with its parent. The splay function takes as input a node x and splays it all the way up to the root based on the double rotations explained above. Please use *rotate* to implement *splay*.

Please copy and paste your *intset.cpp* file from lab 05. You will be modifying this file to add the two functions *rotate* and *splay*.

In your *intset.cpp*, declare the functions as,

```
void IntSet::rotate (Node *x) {
  // Fill your code here
}

void IntSet::splay (Node *x) {
  // Fill your code here
}
```

Please implement the rotate and splay functions as described above.

1.4.2 Find and Insert

After implementing the rotate and splay operations, change the insert and find operations so that an element is splayed to the root after it is accessed.

1.5 Compiling and Testing

You are to write your own code to test the IntSet class. No additional code is provided for this task. If you named your test file *testIntSet.cpp*, then you can compile using the following command.

g++ testIntSet.cpp -o testIntSet intset.cpp

You can enter your input in a file named *testIntSetInput* and redirect your input while executing.

./testIntSet < testIntSetInput

You have to thoroughly test the IntSet class. Make sure your program works for all boundary cases. You will receive full credit only if your code works for all cases. You can use the test code you wrote for lab 05.

2 Performance Testing

We will be testing the performance of *find* of the two implementations of the integer set data structure. In particular, we will be testing the performance of *find* in a splay tree and a sorted linked list. Remember that, in lab 03, an IntSet data structure was developed using sorted linked list.

We wil be measuring the actual time to run a series of *find* operations in a splay tree and sorted linked list and compare the results. The timing for various values of input size (number of *find* operations) can be plotted as a graph and the two functions can be easily compared. This kind of testing is also called as *emperical testing*.

A directory called *list* is already created for you. Please copy the files *intset.h* and *intset.cpp* from lab 03 into the folder named list. Notice the *main.cpp* file in both folders *bst* abd *list*. This *main.cpp*

file makes use of the *IntSet* class which we have now implemented two ways - using a sorted and linked list and as a splay tree. We can clealry see how two different implementations are used without changing *main.cpp*.

The *main.cpp* file inserts 10000 keys into an integer set data structure and performs a bunch of *find* operations. For each sequence of *find* operations, it prints the time it takes to perform those operations. Please run *main.cpp* and note down the time it took for all the operations.

2.1 Compiling

Please compile your *main.cpp* program as:

```
g++ main.cpp -o main -lrt intset.cpp
```

2.2 Plotting

We would like to plot the running times of the *find* operation in a splay tree and a sorted linked list. Please open the *plot.py* file in the parent directory. Please fill the lists *ylistbst* and *ylistlist*. These lists report the running times as noted down from running *main.cpp* for splay tree and sorted linked list respectively. Each element in the list should be separated by a comma. For instance:

```
ylistbst = [12, 13, 14, 16]
```

After filling in these lists, please execute the python code using:

```
python plot.py
```

You should be able to see a plot. Please save the plot as a *png* file and submit.

3 Submission

Please submit your solutions through handin.cs.clemson.edu. In particular, please submit your *intset.cpp* file for problem #1 and the plot for problem #2. The due date for this assignment is Friday February 28th at 11:59pm.

4 Grading

The first section is graded out of 12 points and the plot is graded out of 8 points. No points will be awarded to a program that does not compile. You will recieve 1 point for successful compilation and 1 point for successful execution (no seg faults and infinite loops). 5 points is awarded for correctness and 3 points for efficiency each and 2 points awarded for good code: readability, comments, simplicity and elegance, etc.