CS3610 Project 3

Dr. Jundong Liu

Part A: due – March 9, Thursday, 11:59 pm
Part B: due – March 30, Thursday, 11:59 pm

Introduction

A binary search tree is a data structure designed for efficient item insertion, deletion, and retrieval. The worst-case time complexities of these three operations are all determined by the height of the tree, which could be as bad as n, where n is the number of the nodes. Balanced binary trees, on the other hand, can guarantee O(log(n)) for these operations. AVL tree, named after two mathematicians Georgy Adelson-Velsky and Evgenii Landis, is the first such data structure to be invented.

Code and implementation

In this project, you are asked to implement several functions of AVL trees, including tree traversal, node insertion and deletion. An implementation of the node insertion function, as well as the associated rotation functions, is available in the textbook, which has also been included in this assignment as the start code. The assignment consists of two parts.

0.1 Part A assignment

In Part A, you are asked to extend the start code to implement a "print" function to visualize an AVL tree. More specifically, this function should output the heights and key values of the nodes in an AVL tree, through an inorder traversal. The height of each node is defined as the height of the subtree rooted at that node, while leaf nodes have a height of 1.

You may find it helpful to compare your results with those obtained from an interactive, AVL tree visualization demo found at `https://www.cs.usfca.edu/~galles/visualization/AVLtree.html`.

0.2 Part B assignment

You will implement an AVL node deletion function in Part B. As you may remember, there are four different replacement cases in AVL node deletion. The first three cases are relatively easy to handle. For case 4, we can choose either the *inorder* predecessor or the successor as the replacement. The textbook uses the **predecessor** (on page 652) as the replacement in case 4. Therefore, for consistency, we will use the same setup in this project.

For node insertion, the textbook uses a bool parameter *IsTaller* in insertIntoAVL() function to communicate between a parent node and the affected subtree. The node

deletion procedure takes a similar approach (outlined on page 652), where a bool variable **shorter** (*IsShorter* in the start code) is used to indicate whether the height of the affected subtree has been reduced. In this project, **you are required to use the textbook approach, as well as the associated functions** (rotateToLeft(), rotateToRight(), balanceFromLeft(), balanceFromRight(), etc) provided in the start code, to implement the AVL deletion function. A 80% penalty will be applied if you choose to use a different design or function set. The "print" functions implemented in Part A will be useful in verifying whether your code is functioning correctly or not.

Again, you can use the visualization demo `https://www.cs.usfca.edu/~galles/visualization/AVLtree.html` to check your results.

Input

Input commands are read from the keyboard. Your program must continue to check for input until the quit command is issued. The accepted input commands are listed as follows:

`i k` : Insert node with key value k into AVL tree.

`h` : Print the height of each node using an inorder traversal.

`p` : Print the key value of each node using an inorder traversal.

`r k` : Remove node with key value k from AVL tree.
(When removing a node with two children, take the *inorder predecessor* in left subtree as the replacement.)

`q` : Quit the program.

Please note that the node insertion function ("i k") and quit ("q") are already provided in the start code. **Your task for Part A is to implement the print function ("h" and "p"). In Part B, you will be required to add the deletion function ("r k") to your submission**.

Output

Print the results of the `p` and `h` commands on one line using space as a delimiter. If the tree is empty when issuing the commands `p` or `h`, output the message `Empty`. If an attempt is made to remove a node not present in the tree, print `No node`. If an attempt is made to insert a node with a duplicate key value, print `Duplicate` without adding the new node to the tree. Do not worry about verifying the input format.

Sample Test Case

Use input redirection to redirect commands written in a file to the standard input, e.g. `$ ./a.out < input1.dat`.

Input 1

```
i 100
i 200
i 300
h
p
q
```

Output 1

```
1 2 1
100 200 300
```

Turn In

Submit your source code through blackboard. If you have multiple files, package them into a zip file.

Grading

Total: 100 pts.

- **10**/100 - Code style, commenting, general readability.
- **10**/100 - Compiles.
- **10**/100 - Follows provided input and output format.
- **70**/100 - Successful implementation of the AVL tree.