Programming Assignment #4 - Binary Search Trees

February 12, 2019

1 Introduction

Assignment 4 deals with binary search trees (BSTs). In this assignment, you'll create a program that lets a user enter data, and builds a BST from that data. Your program will also let the user interact with that BST, using a series of commands:

- s number – this will search for "number" in the BST. Tell the user whether that number is found or not.
- d number – search for "number" and, if found, delete it. If it is not found, tell the user.
- X – this command is used to display information about the BST. The following information should be displayed:
  - pre-, in- and post-order traversal of the tree. For each one, output the string "NLR" "LNR:" or "LRN:" followed by a comma-separated list of the nodes. Output one line for each traversal.
  - output data from a breatdh-first traversal of the tree: Output the string "BFS:" followed by a comma-separated list of the nodes.
  - output the height of the tree (an empty tree has height -1; a tree with a single node has height 0; etc.) Your output should say "HEIGHT:" followed by the height.

  - Indicate whether or not the tree is balanced. Your output should say "BALANCED:" followed by "YES" or "NO"

- ? – display a help message giving the user a list of valid commands.
- Q – exit the program

If only a number is entered (positive, negative, or 0), insert it into the tree. **Only allow a number to be entered once**. If the user tries to insert a number that's already in the tree, you should print an error message and not insert the second copy of the number.

Typing anything else should print an error message, and tell the user how to get help.

2 Details

Do not balance the tree! Insert nodes only below leafs, in the unique location that preserves the property of being a BST. When deleting a node N, select the largest node from N's **left** sub-tree to replace N, unless there is no left sub-tree; in that case, move the root to point to N's right sub-tree.

Be sure to free all memory that you allocate. Use valgrind on your code to ensure there are no memory leaks.

Use your queue from PA3 for implementing breadth-first search (BFS).

You should code your ~~insert, search,~~ delete, height, balance-check and depth-first-search routines recursively, as discussed in class. Your breadth-first search should be implemented using a queue.

Create a makefile to build your executable image (which should be called "pa4"). Bundle all your source files and your makefile into a single .tar and upload via Canvas by the due date.

3 Suggestions

I suggest writing your insert and in-order traversal code first. This lets you make a main program that simply calls your insert code with fixed values,

which should build a tree. Then you can call your traversal code to confirm that your insertion works correctly. Once that is working, deletion is the next major hurdle.

Make your code modular, and use simpler pieces to help you in writing the more complex parts of your code.