CS293 Lab 5
A New Tree

General Instructions

- **There is no in-lab and out-lab component of this assignment.**
- **Your final solution is due by Fri, Sep 9, 2pm on Moodle. Please refer to submission instructions to know what EXACTLY to submit.**
- **You must however submit your partial work by 5.15 pm on Fri, Sep 2 on Moodle.**
- **Those who don't submit their partial work by the above deadline will not have their final submissions evaluated.**
- **Please go through the announcement made on Moodle in this regard.**

Introduction

This lab is all about the **Adelson-Velskii and Landis trees**, aka **AVL Tree** aka **Height Balanced Tree**. Are the AVL trees always better than a *run of the mill* Binary tree, In this lab we will try to answer this question.

Problem Statement

The goal is to implement an AVL tree , and observe the relation it has with a Binary Tree, We will also look into the number of operations(Comparisons and Pointer Swaps) which are required in both building and maintaining an AVL tree

We will use a Journey Code (variable name: **val**) as our basic objects, to mark the entries in the tree. The Journey Code will also be used to sort the data and build the tree.

Tree.h

This File contains the Abstract class that will act as a Super Class for the different types of trees. It also contains the print function that will help you to visualize and check your implementations.
Please Do not change this File

BST.cpp

You have been provided an implementation of the BST (*without recursion*) that we will use to compare the statistics of trees. You need to identify the correct locations and update the comparisons and Pointer Movements functions

AVL.cpp

- **This is the primary file you need to change**
- In this file you need to implement the **insert** and **find** functions for an AVL tree while making sure the tree remains balanced.
- You may implement any other utility functions as you see fit,
- **Do NOT change the name of the given functions**
- You also need to keep track of number of **comparisons**, and the number of **Pointer Movements** that you make and increment the appropriate variable accordingly
- **OPTIONAL: You can also Implement the remove function for an AVL tree**

main.cpp

This is the file that you will run to test your implementation

Use the following commands to run the code

```
g++ main.cpp -o main
./main
```

The Operations that are available in the interactive mode are

- `ADD <journey code>` : ADD a new element to the tree
- `DEL <journey code>` : DELETE the code from the tree
- `FIND <journey code>` : FIND if the code is present in the tree
- `PRINT`: PRINTS a representation of the tree

Testing the Code

- You are provided by a checker.sh file

  The Command to run the file is

  ```
  ./checker.sh <sorted|reverse|random> <numOfNodes> <filename> <AVL|BST>
  ```

- The Program will create TestFiles with *numOfNodes* ADD and *numOfNodes* FIND commands, run your code and then plot graphs comparing the number of Comparisons and Pointer Movements with the number of operations
- Check the relationship between the graphs for the different type of data (Sorted in ascending order and descending order and random permutation of the elements)

Submission Instructions

Make the necessary changes in the files.
Keep all the files in a folder named `<ROLL_NUMBER>_L5` and compress it to a tar file named `<ROLL_NUMBER>_L5.tar.gz` using the command

```
tar -zcvf <ROLL_NUMBER>_L5.tar.gz <ROLL_NUMBER>_L5
```

Submit the tar file on Moodle. The directory structure should be -

```
<ROLL_NUMBER>_L5
|--------------AVL.cpp
|--------------BSTgraph.png
|--------------AVLgraph.png
```

If your Roll number has alphabets, they should be in “small” letters.