CS 124 Project 3

Due on Gradescope by Wednesday, October 21st

For this project, you will store your 1000+ objects in a Binary Search Tree, an AVL Tree, and a Splay Tree. You will search for objects in the tree and record how many objects you had to visit to complete the search. You will analyze your results from the different data structures.

Setup

- You will want to start at the end of Project 1. You should have a vector of 1000+ objects in your main function.
- You will need to overload the <, >, <=, >=, and == operators of your class from Project 1. Use a unique field to compare objects of your class.

Binary Search Tree

- Copy the BinarySearchTree.h file from Blackboard into your CLion project.
- Modify the BinarySearchTree's find methods so that a search of the tree also stores the depth of the last node visited. To do this without losing information, pass an integer by reference into both methods and modify it inside the methods.
  - Note: the root node is at depth 0.
  - Note: even if the search fails, you should still record the depth of the last node visited.
- Create an integer Binary Search Tree. Insert the numbers from 1 to 100 in that order. Then search for all 100 numbers in the tree and record their depths in a file.
  - Search for 0, 101, and 102. What return values do you get from the find method? What depths do you get? Why?
- Create another integer Binary Search Tree. Insert the numbers from 1 to 100 in a random order. Then search for all 100 numbers in the tree and record their depths in a file.
  - Note: you can get a random sequence either from random.org or by shuffling a vector of numbers (C++ has a shuffle function).
  - How do these depths compare to the first file of depths? Why?
- Create a BinarySearchTree of your custom data type from Project 1. Insert all 1000+ objects from the vector into the tree, then search for all the objects and write their depths to a file.
  - How do these depths compare to the integer BST depths? Why?
  - Graph these depths to compare with the other trees (do not use C++ to graph, you can use whatever spreadsheet application or graphical programming language you prefer).

AVL Tree

- Copy the AVLTree.h file from Blackboard into your CLion project.
- Modify the AVLTree's find methods to store the depth of the last node visited, as you did in the BinarySearchTree class.
- Create two integer AVL Trees. Insert the numbers from 1 to 100 in order into one tree and in a random sequence in the other tree (in similar fashion to the BST). Search for all the integers in both trees and record their depths in one or more files.
  - How do these depths compare to the BST depths? Why?
- Create an AVL Tree of your custom data type from Project 1. Insert all 1000+ objects from the vector into the tree, then search for all the objects and write their depths to a file.

- How do these depths compare to the BST? Why?
- Graph these depths to compare with the other trees.

Splay Tree

- Copy the SplayTree.h file from Blackboard into your CLion project.
- The Splay Tree constructor takes a bool parameter to decide if you want the tree to splay at the end of the add method. We added this option because the online animations from USFCA splay when adding, and we are curious if it improves performance. You'll find out for us.
- Modify the SplayTree's find methods to store the depth of the last node visited, as you did in the BinarySearchTree and AVLTree classes.
- Create two integer Splay Trees and set splayOnAdd to false. Insert the numbers from 1 to 100 in order into both trees.
  - In the first tree, search for the integers in order and record the depths to a file. Look at the file. Why do these depths make sense?
  - In the second tree, search for the integers in a random order and record the depths to a file. Look at the file. Why are these depths different from the first file?
- Create four Splay Trees of your custom data type from Project 1, where two have splayOnAdd set to true and the other two have it set to false. Insert all 1000+ objects from the vector into the trees.
  - In two of the trees (one with splayOnAdd and one without), search for the objects in the order you inserted them and record the depths to a file.
    - Graph these depths to compare with the other trees.
  - In the other two trees, search for the objects in a random order, and repeat each search five times (i.e. search for the first object five times in a row before searching for the second object, etc.). Record the depths to a file. Why do these depths make sense? Did splaying-on-add make the tree more balanced?

Report and Submission

- Answer all of the above questions in your report.
- Compare and contrast the graphs and explain the differences based on what you know about the structure and behavior of the trees. Justify the complexity of searching the trees based on the results.
  - You can use whatever graph is most readable to you (e.g. scatter plot, histogram, etc.).
- Any code that is not authored by you or the instructor **must be cited in your report**.
- You must submit your source files (including the three modified tree header files, your class header file, and the main program file), your data files (including your .csv data file and all of the output files), and your report (saved as a PDF).

Extra Credit

To earn up to 10 extra credit points (at the grader's discretion), you can get more thorough results. This can include:

- Setting timers to record how long it takes you to search for the objects in each data structure
- Performing more experiments with the order of insertions/searches and analyzing the results
- Performing the same experiments on 100, 200, 300, …N objects and graphing the results

Grading

The project is out of 80 points.

5 pts	Program compiles and runs.
5 pts	Code style. Readable, naming style is consistent, comments where appropriate.
15 pts	BST prompts are complete and correct
15 pts	AVL Tree prompts are complete and correct
20 pts	Splay Tree prompts are complete and correct
20 pts	Report: analysis of results, professional, grammatically correct, cites sources.