This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

Semester Project Part 5: Reading and Writing The AVL Tree

Data Structures and Analysis of Algorithms, akk5

Objectives

- To strengthen student's knowledge of C++ programming
- To give the student experience in writing Data Structures for data types
- To give the student experience writing a non-linear data structure
- To give the student experience implementing a basic load and save feature
- To give the student experience using multiple data structures together to achieve a common goal.

Instructions

Attention!!! You will be using your AVL tree or the AVL code base I have provided for this project. You will also need to use the stack and vector STL's.

For this assignment you will be extending the functionality of your AVL program to including reading from and writing to a data file. This data file will store the post order traversal of the tree – I have found that writing the string stored by each node on separate lines makes for an easy to read and write format.

You should then write a program that allows a user to interact with an instance of the AVL you have implemented. This program should implement a text-based interface that allows the user to:

1. Create an empty AVL. This should warn the user they are deleting the existing AVL and ask them if they wish to proceed. Remind the user they can save the contents of their AVL to a file.
2. Insert a string into the current AVL.
3. Search for a string in the current AVL.
4. Remove a string from the current AVL.
5. Output the in-order traversal of the current AVL.
6. Output the pre-order traversal of the current AVL.
7. Output the post-order traversal of the current AVL.
8. Save the post-order traversal of the current AVL to a user specified filename.
9. Read the post-order traversal from a user specified filename and reconstruct a functional AVL duplicate of the saved AVL
10. Exit.

Make certain you inform the user of the available commands and any information pertaining to the state of the system such as whether an AVL has been created, the number of nodes in the AVL, etc.

This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

This project shares lots of code with the AVL. It would be wise to use your AVL as a starting point for implementing the AVL. I will provide you with a working AVL code base, but not a working user interface to make this easier.

So, how do we convert a post order traversal back into the original tree? We use the following approach:

1. Destroy the existing tree
2. Read the strings from the save file into an array (use a vector for ease of use)
3. Create the root node and assign it the value of the last element of the array
4. Push the root onto an empty stack
5. Iterate I backwards from length(array)-2 to 0
    a. Declare pointer to a node current and assign it the value of the array[I]
    b. Declare pointers to a node temp and top
    c. Loop while the stack isn't empty
        i. Assign top the value of stack.peek()
        ii. If (current's value >= top's value) exit the while loop
        iii. Assign temp the value of top
        iv. Pop an element from the stack
    d. If temp has a value assigned to it
        i. Set temp's left to current
    e. Otherwise
        i. Assign top the value of stack.peek()
        ii. Set top's right to current
    f. Push current onto the stack
6. Finally traverse the newly create tree and fix the heights

This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

Grading Breakdown

Point Breakdown	
Structure	12 pts
The program has a header comment with the required information.	3 pts
The overall readability of the program.	3 pts
Program uses separate files for main and class definitions	3 pts
Program includes meaningful comments	3 pts
Syntax	18 pts
Implements Class AVL correctly	3 pts
Implements Class Node correctly	3 pts
Implements the loadAVL method correctly	12 pts
Behavior	70 pts
Program handles the following correctly	
• Handles all the interface options from the previous program	6 pts
• Read the traversal file and successfully recreate the related AVL	8 pts
• Insert a string after loading an AVL	8 pts
• Remove a string after loading an AVL	8 pts
• Find a string after loading an AVL	8 pts
• Output the pre-order traversal after loading an AVL	8 pts
• Output the post-order traversal after loading an AVL	8 pts
• Output the in-order traversal after loading an AVL	8 pts
• Save the post-order traversal to file after loading an AVL	8 pts
Total Possible Points	**100pts**
Penalties	
Program does NOT compile	-100
Late up to 24 hrs	-30
Late more than 24hrs	-100

This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

This is an individual assignment. Seeking direct help from students, tutors, and websites such as chegg or stack overflow will be construed as a violation of the honor code.

Header Comment

At the top of each program, type in the following comment:

/*

Student Name: <student name>

Student NetID: <student NetID>

Compiler Used: <Visual Studio, GCC, etc.>

Program Description:

<Write a short description of the program.>

*/

Example:

/*

Student Name: John Smith

Student NetID: jjjs123

Compiler Used: Eclipse using MinGW

Program Description:

This program prints lots and lots of strings!!

*/

Assignment Information

Due Date: 11/4/2019 (Section 1), 11/3/2019 (Section 3)

Files Expected:

1. Main.cpp – File containing function main
2. AVL.h - File containing class Node and class AVL.
3. AVL.cpp - File containing the implementation for the AVL