COMP 2012 Object-Oriented Programming and Data Structures

Lab 8 Binary Search Tree (BST)

HKUST Library

Introduction

You are hired to be a librarian in HKUST library. They used to build a binary search tree to manage the books, but the former technician just left. Now it is your responsibility to maintain the BST. In this lab, you will implement some operations to the binary search tree based on what you learned in the class.

Description

In the lecture, you have learned the concept of binary search tree, so we will not repeat it here.

Luckily, the former technician has implement the `BinarySearchTree`, and the correspond basic operation for you, including the constructor, the destructor, the insertion, the searching, the printing function. In the BST, there is only a data member `value` to identify the books. All the rest are necessary for the BST.

To prove your value, you decide to improve the existing BST by adding two more operation:

- Justifying whether two BST are the same.
- Given a BST and a value in this BST, construct a new BST whose root is the given value, and other nodes are the same. To make it easier, we assume that the new BST first sets the given value as the root and then traverses the given BST in an **inorder** way and add nodes one by one.

You believe that with this two operations, you can avoid the worst case of the BST by using random number to construct a different BST and then check whether it is the same as the former.

The example is shown as follows:

Page maintained by

ZHUO Yuxuan

Last Modified:
11/24/2022 17:24:49

Homepage

Course Homepage

```
The first BST:
                10
                        9
                                7
        5
                        4
                3
1
        -2
                        -3
                -5
The second BST:
                10
                        9
                                7
        5
                        4
                3
1
        -2
                        -3
                -5
Whether they are the same:1
Different tree of the first tree with the new root -2:
                                                        10
                                                9
                                        7
                                5
                        4
                3
        1
-2
                -3
        -5
```

Lab tasks

You can download the skeleton code HERE.

Task 1

Implement the `BinarySearchTree::isSame(const BinarySearchTree* other)` in the `BinarySearchTree.cpp`. Hint: use the recersive method!

Task 2

Implement the `BinarySearchTree::traverseAdd(BinarySearchTree* newBST)` in the `BinarySearchTree.cpp`. We have given you the framework of the recursive method. The `BinarySearchTree::differentTree(BinarySearchTree* newBST, int newValue)` is the **entry** of the operation and the `BinarySearchTree::traverseAdd(BinarySearchTree* newBST)` is the recursive function.

Submission

Deadline: 10 minutes after your lab session, Nov 24/25, 2022.

Please submit the following files to ZINC by zipping the following 1 file. ZINC usage instructions can be found here.

- `BinarySearchTree.cpp`

Notes:

- You may submit your file multiple times, but only the last submission will be graded. **You do NOT get to choose which version we grade.** If you submit after the deadline, late penalty will be applied according to the submission time of your last submission.
- Submit early to avoid any last-minute problem. Only ZINC submissions will be accepted.
- The ZINC server will be very busy on the last day especially in the last few hours, so you should expect you would get the grading result report not-very-quickly. However, as long as your submission is successful, we would grade your latest submission with all test cases after the deadline.
- In the grading report, pay attention to various errors reported. For example, **under the "make" section, if you see a red cross, click on the STDERR tab to see the compilation errors.** You must fix those before you can see any program output for the test cases below.
- Make sure you submit the correct file yourself. You can download your own file back from ZINC to verify. Again, **we only grade what you uploaded last to ZINC**.

Compilation Requirement

It is **required** that your submissions can be compiled and run successfully in our online auto-grader ZINC. It won't be graded if we cannot even compile your work. Therefore, for parts that you cannot finish, just put in dummy implementation so that your whole program can be compiled for ZINC to grade the other parts that you have done.