COMP 2012 Object-Oriented Programming and Data Structures

Lab 7 BST

Image by Freepik

Overview

In this lab, you are going to implement a grade recording system based on Binary Search Tree. Imagine you are a TA of COMP2102 (Is this course real?). In this course, each student's score has two parts: the **base score** and the **bonus score**. The following rule would sort each student: **First, they are sorted by base scores. If two students have the same base score, they are sorted by bonus scores.** Assume that all students have different grades, that is, **assume that no students would have identical (baseScore, bonusScore) score pairs** (But of course they can have the same value on either of the score).

You can download the skeleton code in a zip package HERE.

BinarySearchTree

```cpp
template<typename T1, typename T2>
struct ScorePair {
    T1 baseScore;
    T2 bonusScore;
};

template<typename T1, typename T2>
class BinarySearchTree
{
private:
    BinarySearchTree *left_sub_tree;
    BinarySearchTree *right_sub_tree;
    string name;
    ScorePair<T1, T2> score;

public:
    BinarySearchTree() = delete;
    BinarySearchTree(const string&, const T1&, const T2&);

    // TODO
    BinarySearchTree(const BinarySearchTree&);
    ~BinarySearchTree();

    int size() const;
    bool findByScorePair(const T1&, const T2&) const;
    bool findByName(const string&) const;
    void insert(const string&, const T1&, const T2&);
    void inorderTraversal() const;

    //given
    void printName() const;
    void printPoint() const;
    void printAll() const;

};
```

The definition of this class can be found in **BinarySearchTree.h**. Note that this tree implementation is different from the one provided in the lecture notes. Each node in the tree contains two value: **name** and **score**, where **score** is a self-defined template struct. In the given test cases, we will use `ScorePair<int, float>` to save the grade, but we may also test other cases at grading.

Be careful about the sorting rules in the tree. You can refer to the examples below as well as the sample output.

Example 1:
Given two students **A** with score (75, 1.0) and **B** with score (74, 3.0), we will sort B before A.

Example 2:
Given two students **A** with score (75, 1.0) and **B** with score (75, 3.0), we will sort A before B.

Example 3:
Given two students **A** with score (75, 2.0) and **B** with score (76, 0.0), we will sort A before B.

Lab Tasks

Your task is to implement all missing function definition in the **BinarySearchTree** class so that the program can produce the correct results as expected. You are required to add appropriate lines of code in 8 different places in **BinarySearchTree.tpp**. The file with extension `tpp` is just a way to annotate the template implementation in a separate file. If you are not familiar with it, see here.

TODO #1
Implement the copy constructor.

TODO #2

Implement the destructor.

TODO #3
Implement the function `size` which returns the size of the bst.

TODO #4
Implement the function `findByScorePair` which print out the name of the score and return true if the score exists in the tree, otherwise do nothing and return false.

TODO #5
Implement the function `findByName` which print out the score of the name and return true if the name exists in the tree, otherwise do nothing and return false.

TODO #6
Implement the function `insert` which do the insertion if both the name and the score do not exist in the tree.

TODO #7
Implement the function `inorderTraversal` which go over the tree and print the information by inorder.

Sample Sessions

Here is the sample output of the program.

```
Lab 7 Exercise.
Start processing binary search tree 1.
Create the tree with Alice: (60, 2.5)
Insert Steven: (75, 2.0)
Insert Kevin: (74, 3.0)
Insert Tom: (55, 1.0)
Insert Desmond: (90, 4.0)
Insert Cecia: (90, 3.5)
The size of tree 1 is 6.

Check if contains (90, 4.0):
The name is Desmond.
Result: true

Check if contains (77, 2.0):
Result: false

Check if contains Kevin:
The score is (74, 3).
Result: true

Check if contains Carl:
```

Submission & Deadline

The deadline of submission for all lab sections is **20:00:00 on 28 April 2023 (Fri)**.

Please submit your completed work to ZINC by zipping the following file.

> **`BinarySearchTree.tpp`**

ZINC usage instructions can be found here. You can make multiple submissions to ZINC before the deadline. Only the last submission received before the deadline will be graded.

```
Try to insert Grace: (70, 1.5)
```

Menu


```
Name: Cecia, Point: (90, 3.5).
```

Page maintained by

TANG Tianhao

Last Modified: 04/19/2023 07:35:29

After you have properly implemented all missing function definition for the specific operators (**TODO #1** ~ **TODO #7**), you can

Homepage

Course Homepage

as shown in the three sample sessions. In addition to the three public test cases, your program will also be assessed with an unseen test case in ZINC.