ITU Computer Engineering Department
BLG 223E Data Structures, Fall 23/24
Homework 3

Deadline: 18.12.2023

1 Introduction

In this assignment, similar to previous ones, you will read, write, delete, and update the given CSV files based on the operation files. You will implement these operations on three different data structures: a binary search tree, a hybrid of binary search tree and a vector, and a "map" container from the Standard Template Library (STL).

2 Dataset and Operation Files

You are provided two test folders. In the first folder, we provided a mini-dataset to test your binary search tree implementation to make sure that you implemented the tree correctly. So, this test folder is exclusive to section 3.1. The other folder contains 2 datasets with 100k and 500k employees. We also provided the shuffled versions of the datasets to make the tree more balanced. You will test both ordered and shuffled versions of the datasets and compare them.

For the first test, to diagnose your implementation better, we added two new commands special to the operation files in the first folder. Their names and expected code correspondences are as following:

- "PRINT": `cout<< "P\n"; bt.printToConsole();`
- "HEIGHT": `cout<< "H "<< bt.getHeight()<<"\n";`

assuming "bt" is an instance of the BinaryTree class. (Check out the related section 3.1). For the second test set, the structure of the operations will be the same as the previous homework. If you have any confusion regarding the operations, please refer to the related section of the previous homework.

3 Problem Description

As in previous homeworks, you will need to use the following class to represent the employees:

```
class Employee{
private:
    int id;
    int salary;
    int department;
public:
    Employee(int i, int s, int d);
    // setters and getters...
};
```

As mentioned, you are required to complete the task in three different ways: using a binary search tree, a hybrid of the binary search tree and vector, and a "map" from the standard library. Notice that you will submit each solution separately.

3.1 Binary Tree Solution

For the first solution, you will need to implement a binary search tree to store and operate on the data. Use the following declaration of the Node class to encapsulate the Employee class and make it have right and left pointers:

```
class Node{
    Employee *employee;
    Node *left;
    Node *right;
public:
    Node(Employee *employee);
    // setters and getters...
};
```

Using the Node class, implement the binary search tree as in the following declaration:

```
class BinaryTree{
private:
    Node *root;
public:
    BinaryTree()
    void insert(Employee* employee);
    void remove(int id);
    Employee *search(int id);
    int getHeight();
```

```
    void printToFile(ofstream &output_file);
    void printToConsole();
};
```

Construct the tree based on the ID of the employees, i.e. ID of the left child must be smaller than the ID of the right child.

There are two resembling print functions. The first one, as its name implies, writes the content of the tree into a CSV file in an **in-order** fashion, whereas the second one prints it directly to the console in **pre-order**.

After completing the operations, call the "printToFile" function to save the content of the tree into a file. Do you notice any pattern in the output? Is this a coincidence?

3.2 Hybrid Solution

Rather than keeping all the data in a single large tree, a hybrid approach involving a vector of trees can be adopted. Utilizing the binary search tree model implemented in the previous section, construct a vector where each tree in the vector corresponds to a specific range of employee IDs i.e. the ith tree in the vector will store employees whose IDs fall within the range of [i*5000, (i+1)*5000) as illustrated in the figure 1.

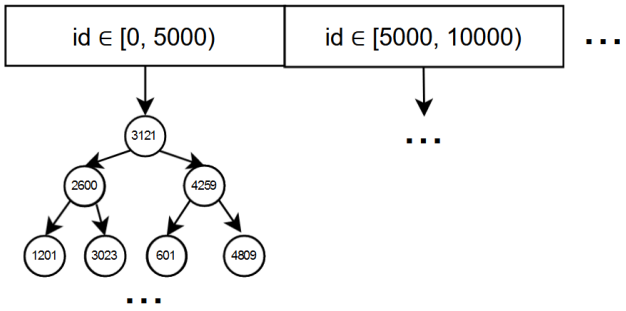

Figure 1: Vector of trees example

Notice that when you add a new employee, you may be required to create a new tree and append it to the vector.

Ponder about the pros and cons of the hybrid solution compared to the plain binary search tree. For instance, you can check the average height of the trees, and compare the execution times.

3.3 STL Map Solution

You will solve the same problem using the Standard Template Library (STL). However, since STL only contains generic containers, there is no "std::tree" or something similar. The closest thing is the "std::map", which stores data as

key-value pairs[1]. In our problem, key-value pair corresponds to id-employee pair respectively. Here is a short code snippet to get you started with std::map.

```
#include <map>
map<int, Employee*> employee_map;
Employee* employee = new Employee(id, salary, department);
// insert
employee_map[employee->get_id()] = employee;
// access
cout << employee_map[employee->get_id()]->get_salary();
```

You are expected to check the documentation[2] by yourself in case you need further details.

4 During the Implementation

Notice that due to the nature of some algorithms, sometimes recursive implementation might be more convenient to write elegant and concise code. However, in practice, each function call brings along an overhead during the execution. Further, it might even cause to program to crash. To prevent this, you may need to find alternative solutions.

You can add new attributes or functions to given class declarations but **do not change the signature** of the current functions or name of the attributes.

Only include related libraries to your code. For example, during the first task do not include vector or map libraries. Also, **never** use "bits/stdc++.h" library.

Your tree implementation will be tested extensively. Make sure you cover all the edge cases as much as possible. It is a good idea to first implement a binary search tree with a toy data type, and make sure every functionality works as expected. Only then fully implement it with the Employee class.

Be careful about the **memory leaks!** Make sure you have a delete call corresponding to each allocation (new call). Do not forget to implement the destructors. Keep in mind that if your code has memory leaks, it might be degraded heavily!

5 Execution Time Measuring

Print and compare the execution times to the terminal to compare each solution. You can use the following code and time.h library:

```
#include <time.h>   // library
```

[1]Typically, it is implemented with a balancing binary tree (usually with Red-Black Tree) under the hood (source)

[2]Documentation

```
clock_t start = clock();   // start to measure
//.... code for measuring
clock_t end = clock();   // finish measure
(double)(end - start) * 1000 / CLOCKS_PER_SEC; // in milliseconds
```

You can examine execution times for addition, deletion and update operations with printing to terminal. For example "ADD: array solution 100 milliseconds". After performing these operations on both solutions, find out which solution is faster for each add, update and delete operations. This will be done just to see comparison of the time it takes to perform different operations on different solutions. Therefore, comment (do not delete) these parts later.

6 Submission Rules

- You cannot use Standard Template Library (STL) except when it is specifically asked. Do not use any other libraries too.
- Make sure you write your name and number in all of the files of your project, in the following format:
  $/* @Author$
  $StudentName :< studentname >$
  $StudentID :< studentid >$
  $Date :< date > */$
- You will submit 3 C++ files.
- Use comments wherever necessary in your code to explain what you did.
- Your program will be checked by using Calico (https://github.com/uyar/calico) automatic checker.
- DO NOT SHARE ANY CODE or text that can be submitted as a part of an assignment.
- Only electronic submissions through Ninova will be accepted no later than deadline.
- You may discuss the problems at an abstract level with your classmates, but you should not share or copy code from your classmates or from the Internet. You should submit your **own, individual homework.**
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- If you have any questions about the homework, you can send an e-mail to Enes Erdoğan (erdogane16@itu.edu.tr).
- Note that **YOUR CODES WILL BE CHECKED WITH THE PLAGIARISM TOOLS!**