COMP2113 Programming technologies [2019]
Final Programming Project

General Instructions

In this project, you will solve **THREE** dependent programming tasks and write a short README document which briefly introduces your code and logic in each task. We will grade your submission manually with reference to your program logic, so there is generally no input/output requirement. You should implement and test the program by following questions in each task. You can use whatever programming language you like in this project, but we encourage you to use C/C++ or Python, as we had multiple practices over them.

Total: 100 points
30 points for the first program
30 points for the second program
30 points for the third program
10 points for your document

Submission

1. The deadline will be announced in Moodle. Late submission will not be accepted.
2. You will receive 0 marks if you submit after the deadline.
3. You will receive 0 marks if you submit incorrect files.
4. You should compress your code (3) and document (1) into a ZIP/TAR file, and submit only **ONE** compressed file.

Evaluation

We will review your work individually to ensure that you receive due credit for your work. Please note that both your program output and logic will be considered for marking.

Academic Dishonesty

We will check your code against other submissions in the class and from the Internet for logical redundancy. If you copy someone else's code and submit it with minor changes, we will know. We trust you all to submit your own work only; please don't let us down. If you do, we will pursue the strongest consequences available to us.

About the project

As we have introduced several programming technologies, in this final project, you are going to learn, develop and optimize an advanced and useful data structure – Binary Search Tree (BST). BST stores "items" (such as numbers, names etc.) in computer memory. It allows fast lookup, addition and removal of items. You will need to understand the operations of BST through on-line sources (e.g., Google, Github) and implement them according to our requirements.

Here are some links for useful information in this project:
1. https://en.wikipedia.org/wiki/Binary_search_tree
2. https://en.wikipedia.org/wiki/AVL_tree
3. https://www.cnblogs.com/lordage/p/6031616.html?from=timeline&isappinstalled=0

4. **https://hackernoon.com/learn-c-multi-threading-in-5-minutes-8b881c92941f**
5. https://www.tutorialspoint.com/cplusplus/cpp_multithreading.htm

---

Task 1 BST(30 points)

Hospital A's database records patients' identities and their ages. Here is a sequence of <id, age> pairs: <3, 10>, <1, 20>, <4, 17>, <6, 31>, <7, 17>, <13, 9>, <8, 25>, <10, 11>, <14, 28>. A binary search tree has organized them into:

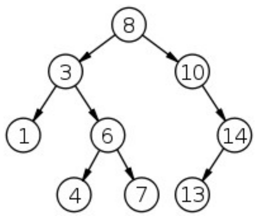


Figure 1. Sample Binary Search Tree

The id of each pair acts as the key in each node. For example, the root node of this tree is 8, which represents the pair <8, 25>. Now you have to code for three operations of BST: addition, removal and searching for a node. Please make sure you have understood how BST works and then start your coding.

Q1. Implement the BST data structure and write three functions named insert(), remove() and search() which correspond to the three functions.

Q2. Initialize the BST as showed in Figure 1. Then, call function insert() to add two nodes whose <id, age> pair are: <2, 49>, <0, 33> to this tree, respectively.

Q2. Call function remove() to delete node 7 in the tree while keeping the nodes in sorted order.

Q3. Call function search() to query the age of the patient whose identity is 13.

Task 2 AVL Tree(30 points)

Binary Search Tree greatly decreases the searching time in large database since it leverages the idea of binary search (refer to https://en.wikipedia.org/wiki/Binary_search_algorithm for details), which is useful in many cases. For example, if you play a game to quickly guess a specific number ranging from 1 ~ 100, it is best to first guess number 50, and then decide whether to guess 25 or 75 based on whether 50 is smaller or larger, and so on. However, binary search tree is not always efficient. Consider the unbalanced binary search tree below, the height of the left subtree is 3 while the right one is 1.

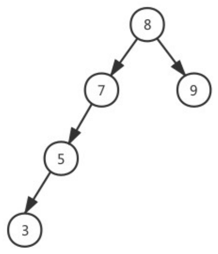
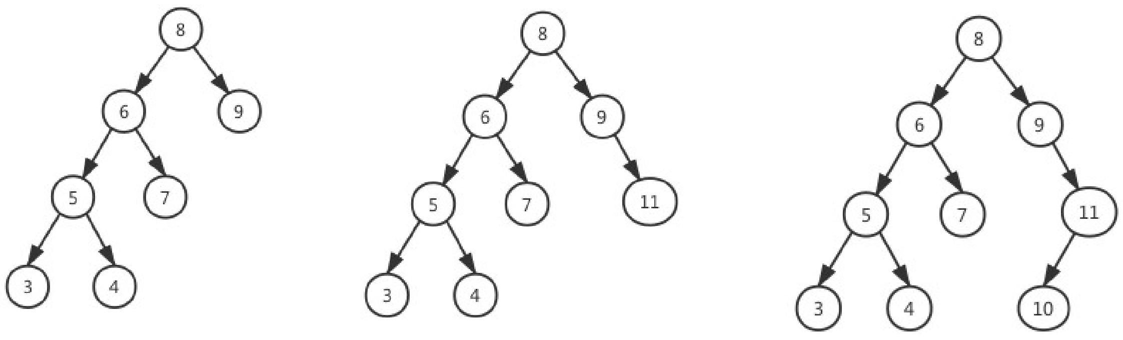


Figure 2. unbalanced binary search tree

If we search for the value whose key is 3, then the binary search algorithm retrogrades to linear search in all data. To be more specific, in the worst case of a search in a binary search tree, the performance could degrade greatly. The AVL tree figures out this problem by balancing the height of subtrees for each node. Details of AVL tree can be found in above link **2** and **3**.

Q1. Point out all the unbalanced trees below. (Write your answer to README document)


(a) (b) (c)

Q2. After executing all the questions in task 1 (Q1 ~ Q3), is the binary search tree balanced or not? (Write your answer to README document)

Q3. If the binary search tree in Q2 is unbalanced, write a function rebalance () to rebalance the tree after each operation by using the AVL algorithm. As there are many approaches to make it balanced (e.g., RR, LL), please indicate your chosen approach clearly in the README document.

Task 3 Concurrent Search(30 points)

As a balanced binary search tree, AVL tree has better performance in extreme cases (e.g., Figure 2). In other word, AVL is an optimized version of BST which accelerates the searching process. In this task, you are going to write a multi-thread version of BST, which also accelerates searching.

Multithreading is the ability of a CPU (or a single core in a multi-core processor) to provide multiple threads of execution concurrently, supported by the operating system. All the programs you write in this class are the single-thread program, which means that they use only limited computation power of a computer. Thus, by using multi-thread search in a binary search tree, the efficiency gets better.

Before programming, you will need to learn the multithreading technique by yourself (e.g., you may refer to link **4** and **5** for some help).

Q1 (the only question in this task): Write a program that conducts a concurrent preorder traversal of the BST of task 1 (after executed Q1 ~ Q3), and outputs

1. the result of preorder traversal (i.e., a list of node identities).

2. the age of a patient whose identity is 6.

About preorder traversal: https://en.wikipedia.org/wiki/Tree_traversal

Hint:
1. You may use a thread pool. In each program run, assign the left subtree to a thread, and assign the right subtree to a free thread in pool (if any). Otherwise, the thread itself deals with the searching.
2. An unbalanced binary search tree may make the multi-threading program ineffective because there will always be one thread dealing with a long subtree while another thread only deals with a very short tree (see Figure 2), you have to design to make thread assignment fair. (You do not need to consider the case, however, it's a plus for making.)

Important

Remember to write a README document that briefly introduce your code in each task and how to run them (e.g., environment, command). Also, you have to answer some of the questions in your README.