IIT Ropar
CSL201 Data Structures
Semester 1, AY 2016/17

Lab Assignment 4 - 50 marks   Due on 25th October, 11:59 PM

Objective

To understand and implement AVL Trees and Red Black Trees.

Instructions

1. DO NOT use STL data structures. You have to implement all data structures on your own.
2. There are six programming questions; you only need to submit first three questions. Other questions are not graded; they are just for practise.
3. You are to use C++ programming language to complete the assignment.
4. Provide a Makefile to compile your final code, keep class definitions in .cpp file and class declarations in .h file.
5. This is an individual assignment. You can have high level discussions with other students, though.
6. Include a “Readme.txt” file on how to compile and run the code. Your program should on institute ubuntu machines.
7. Upload your submission to moodle by the due date and time. After due date, your submission will be evaluated with 10% penalty per day for next two days. After that, your submission will not be evaluated.

Questions

1. [20 marks] Implement an ordered map with height balanced binary search tree (AVL tree). The ordered map should contain the following functions in its interface:-

   1. **firstEntry(k):** Return an iterator to the entry with smallest key value; if the map is empty, it returns end.
   2. **lastEntry(k):** Return an iterator to the entry with largest key value; if the map is empty, it returns end.
   3. **ceilingEntry(k):** Return an iterator to the entry with the least key value greater than or equal to k; if there is no such entry, it returns end.

4. **floorEntry(k):** Return an iterator to the entry with the greatest key value less than or equal to k; if there is no such entry, it returns end.
5. **lowerEntry(k):** Return an iterator to the entry with the greatest key value less than k; if there is no such entry, it returns end.
6. **higherEntry(k):** Return an iterator to the entry with the least key value greater than k; if there is no such entry, it returns end.
7. **size():** Return the number of entries in M.
8. **empty():** Return true if M is empty and false otherwise.
9. **find(k):** If M contains an entry e=(k,v),with key equal to k,then return an iterator p referring to this entry, and otherwise return the special iterator end.
10. **put(k,v)**: If M does not have an entry with key equal to k, then add entry (k,v) to M, and otherwise, replace the value field of this entry with v; return an iterator to the inserted/modified entry.
11. **erase(k):** Remove from M the entry with key equal to k; an error condition occurs if M has no such entry.
12. **erase(p):** Remove from M the entry referenced by iterator p; an error condition occurs if p points to the end sentinel.

Insert a given sequence of numbers in an initially empty AVL tree. Display the resulting binary tree after each step. If you have to rotate the tree after inserting an element, display the tree after each rotation. You can use the algorithm given on page page 301 of the textbook to display the tree. During the assessment, we can ask you to explain the code of any functionality (rotation, rebalancing, 12 functions, etc.). If you are unable to explain even a small part of your code, we will assume the code is not fully written by you and grade accordingly.

2. [10 marks] Read a sequence of n space separated integers from a file and insert into the ordered map implementation of the previous question. Now, for a given target sum, write a function that returns true if there is a pair with sum equals to target sum, otherwise returns false. The time complexity of the algorithm should be O(n) and only O(log n) extra memory space can be used. Do not modify the structure of the binary search tree. Note that you cannot read the numbers out in an array as this will require O(n) memory space. Your algorithm should work on the AVL tree.

3. [20] Implement a Red Black tree. Along with **insert(k,v)**, **remove(k)**, and **search(k)**, include the following functions:

a. Delete a node from the red black tree.
b. Count the number of leaves in a tree.
c. Return the height of a tree.
d. Return a list of all keys in a tree between a and b.

The program should display the tree after each step. You can use the same display function you implemented for question 1 as both (AVL and Red Black) are binary search trees. During the assessment, we can ask you to explain the code of any functionality. If you are unable to explain even a small part of your code, we will assume the code is not fully written by you and grade accordingly.

Assessment

- You will be assessed individually in the lab session (you must attend your assigned lab!). You'll have approximately 15 minute Q&A session with Dr. Mukesh or one of the TAs (Raghu, Neeraj, or Shreya).
- You will have to run the code directly from your Moodle submission, you won't be allowed to bring an updated version to the lab.
- After you run the code, we will ask you anything about your code we want. Be prepared to answer the questions.

Additional Questions

4. You are given two binary search trees (not necessarily balanced). Design an algorithm that merges the two given trees into a balanced binary search tree in linear time.
5. Let A be an array of n elements with the following property: there exists an $i \in [0, n-1]$ such that $A[0] < A[1] < \ldots A[i-1] < A[i] > A[i+1] > \ldots A[n]$. Show how to find such an i in $O(\log n)$ time.
6. Given two binary search trees, design a generic algorithm that uses left and right rotations to convert one binary search tree into another binary search tree.