EC330 Applied Algorithms and Data Structures for Engineers
Spring 2020

Homework 6

Out: April 2, 2020
Due: April 11, 2020

This homework has a written part and a programming part. Both are due at 11:59 pm on April 11. You should submit both parts on Gradescope.
For the written part, you should submit a single PDF file containing either typeset answers or scanned copies of hand-written answers. Make sure you write your answers clearly.
For the programming part, you should make sure your code compile and run on Gradescope. You should also try to create your own test cases to test and debug your code. Make sure you write your name and BU ID in a comment at the top of the program, and use clear comments to explain the key steps in your code.

1. AVL Tree and B-Tree [20 pt]

a. In a binary tree, a child is called an "only-child" if it has a parent node but no sibling, i.e. it is the only child node of its parent. Prove that for any AVL tree with n nodes ($n > 0$), the total number of only-children is at most $n/2$.
b. Is the following statement "the order in which elements are inserted into a B-tree does not matter, since the same B-tree will be constructed" true? If yes, explain why. If not, provide a counterexample.

2. Red-Black Tree [20 pt]

a. An *in-order* traversal of a red-black tree on the set of numbers 1 to 10 gives the following colors: R B R R R B R B R B. Produce the tree.
b. A *post-order* traversal of a red-black tree on the set of numbers 1 to 10 gives the following colors: B B B R B B R B B B. Produce the tree.

3. Binary Search Tree [30 pt]

Consider two binary search trees that contain the same set of *unique* keys, possibly in different orders. Design an efficient algorithm that will transform any given binary search tree into any other binary search tree (with the same keys) using only ZIG and ZAG rotations. Your algorithm should have a runtime in $O(n^2)$ or better.

The provided *BST.h*, and *BST.cpp* files contain a BST class, implementing a binary search tree, and a Rotation class, which stores a rotation.

Implement a new derived class of BST, MyBST, which extends the binary search tree with the aforementioned *transform* method. You are allowed to modify the source BST in *transform*.

To get full credit your solution must work on any two binary search trees that contain exactly the same set of unique keys.

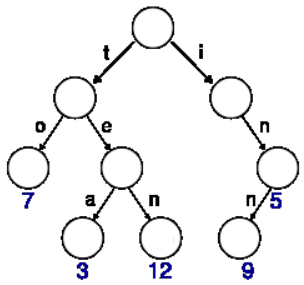
You may not modify *BST.h* or *BST.cpp*. You can add methods to but not remove any from *MyBST.cpp* (and modify *MyBST.h* accordingly) as you see fit. Submit both *MyBST.h* and *MyBST.cpp* on Gradescope.

4. Trie [30 pt]

Implement a **trie-based map**. A map stores data in the form of (key, value) pairs where every key is unique and each key maps to a value. You can assume that the keys are strings composed of lowercase letters only, and the values are positive integers.

Implement the *insert, search,* and *delete* methods in trie.cpp. For *delete*, when you remove a leaf node, you should not remove redundant ancestors if there are any.

For example, below is the result trie after the sequence of insertion ("to", 7), ("tea", 3), ("ten", 12), ("in", 5), ("inn", 9). Note that the size of the resulting map should be 5 and the size of the resulting tree should be 9.



Submit the *trie.cpp* file on Gradescope.