EC330 Applied Algorithms and Data Structures for Engineers
Spring 2021

Homework 5

Out: March 29, 2021
Due: April 7, 2021

This homework has a written part and a programming part. Both are due at 11:59 pm on April 7. You should submit both parts on Gradescope.

This is an individual assignment. See course syllabus for policy on collaboration.

1. AVL 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 statement "the order in which elements are inserted into an AVL tree does not matter, since the same AVL tree will be established following rotations" true? If yes, explain why. If not, give a counterexample.

2. Red-Black Tree [10 pt]

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 [10 pt]

Devise 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 run in time $\boldsymbol{O}(n^3)$ or better.

4. Programming [60 pt]

a. We will implement a Bloom filter to check active phishing URLs in this problem. The files *phishing-links-ACTIVE.txt* and *phishing-links-INACTIVE.txt* contain two sets of phishing URLs obtained from https://github.com/mitchellkrogza/Phishing.Database. Each line in these two files is a URL. We will design a Bloom filter so that the Bloom filter contains the set of ACTIVE phishing links, and the false positive rate for classifying an INACTIVE phishing link as ACTIVE is minimized. The file *phishing-links-INACTIVE.txt* contains 10% of all the INACTIVE links that we will use to test your implementation. The size of the Bloom filter is set to 330. Submit *bfilter.cpp* on Gradescope. **[40 pt]**

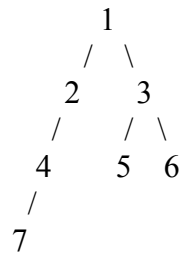
For full credit, your Bloom filter implementation should have *a false positive rate strictly less than 30%.*

The submission that has the lowest false positive rate will get **10** bonus points.

Below are some resources for hash functions that you can use. You are free to use other hash functions.
http://www.partow.net/programming/hashfunctions/index.html#AvailableHashFunctions
https://en.wikipedia.org/wiki/Double_hashing
http://hashlib2plus.sourceforge.net

b. Implement the method *isWeightBalanced(node* root, int k)* for checking if a binary tree is *weight balanced*. Similar to the notion of height balance in AVL trees, we define a binary tree to be *k*-weight balanced if for every node in the tree, the difference between the weight of the node's left subtree and the weight of the node's right subtree is no more than *k*. The weight of a (sub)tree is simply the number of nodes in that (sub)tree. Submit *balance.cpp* on Gradescope. **[20 pt]**

For example, in the binary tree below, the weight of the subtree rooted at *node2* is *3*, and



isWeightBalanced(*node1*, *0*) should return 0,
isWeightBalanced(*node2*, *1*) should return 0,
isWeightBalanced(*node3*, *0*) should return 1,
isWeightBalanced(*node4*, *1*) should return 1.