CSci 242: Algorithms and Data Structures

Instructor: Dr. M. E. Kim Date: 2-1-2022

Due: 11:59 PM, February 12th (Sat.), 2022. (No Extension) Name: Adam Roy

**Home Assignment 2: 150 points + 20 points (optional)**

Write your answer in blue or in black, not in Red.

Read the submission instruction on BB and strictly follow it.

Q1. [10] **Binary Tree**

Let T be a binary tree with *n* nodes. Define the ***Lowest Common Ancestor*** (**LCA**) between two nodes *v* and *w* as the ***lowest node*** in T that has both *v* and *w* as descendants. Given two nodes *v* and *w*, write an efficient algorithm, LCA(*v, w*), in a ***pseudo code*** for finding the LCA of *v* and *w*, using the structure of the tree, not the values of keys.

Note: A node is a descendant of itself and *v*.*depth* gives a depth of a node *v*.

* The Lowest Common Ancestor is the shared ancestor of nodes v and w within binary tree T that are the furthest from or have the greatest distance from the root of the tree.

Algorithm LowestCommonAncestor(Troot, v, w):

Input: A Binary Tree, T, A node, T(v), A second node, T(w)

Output: A LCA of the two nodes

If (not root)

Return NULL

If (root == (is equal to) v or root == (is equal to) w)

Return root

nodeLeft ← LowestCommonAncestor(root.left, v, w)

nodeRight ← LowestCommonAncestor(root.right, v, w)

if (not left)

Return right node

else if (not right)

Return left node

else

Return root

Q2. [115] **(Link-based) Binary Search Tree (BST)**

**Implement** a BST ADT that includes the following operations and other supporting (or required)

operations in Python. For each operation, print its output with the given data.

(*k, e*)– an **item** of (*key*, *element*),

*v, w* – node

1. [15] ***Insert*(*k, e*)** : Create a binary search tree by inserting the following items, (*k, e*), to an initial empty BST:

(25, C), (35, G), (45, B), (20, P), (30, Q), (5, Z), (55, L), (43, F), (22, A), (6, U), (8, N), (40, R)

***InOrder*(*v*)**: Then, Print the *keys* of the BST by *InOrder* traversal at a tree rooted at *v*.

So, v will be a root of the created tree in the initial call of *InOrder*(v),

In the created BST in 1)

1. ~~[10]~~***~~root~~***~~( )~~~~: Return a root node of the tree~~~~and print its item.~~
2. ~~[10]~~***~~Search~~*~~(~~*~~k, v~~*~~)~~**~~: Search/return a node whose key is 45 in the BST rooted at~~ *~~v,~~* ~~and print the returned item.~~
3. ~~[10]~~***~~Successor(v)~~***~~:~~
   1. Find/return a successor of a node *v* whose key is 8 and print its item.
   2. Then, find/return a successor of a node with a key 35, and print its item.
4. ~~[10]~~***~~Predecessor(v)~~***~~:~~ 
   1. Find/return a predecessor of a node whose key is 20 and print its key.
   2. Then, find/print the predecessor of a node with a key 40, and print its item.
5. [20]***removeAboveExternal*(*w*):** Remove an external node *w* and its parent node *v*, then reconnect *v*’s parent with *w*’s sibling.

***Remove(k)***: Remove (a) a node whose key is 35, then remove (b) the node with a key 5

–Implement it with *removeAboveExternal*(*w*).

***PostOrder(v)***: Then, (c) print the keys after each removal in 6.(a & b) by *PostOrder* traversal.

1. [10]***rangeQuery*(*k1, k2, v*)**: find and print the keys in the range of [25, 45].
2. ~~[10]~~ ***~~isExternal(v)~~*~~:~~** ~~Test whether a node~~ *~~v~~* ~~with a key 40 is an external node or not.~~
3. ~~[5]~~ ***~~isRoot(v)~~*~~:~~** ~~Test whether a node~~ *~~v~~* ~~with a key 25 is the root of BST or not.~~
4. [15] Implement the LCA(v, w) algorithm of Q1 and find/print the key of LCA(*v, w*) where
   1. [5] *v.key* = 30, *w.key* = 40
   2. [5] *v.key* = 6, *w.key* = 55
   3. [5] *v.key* = 35, *w.key* = 45

Q3. [25 pt.] **Range Query in BST**

1. [15] Implement a Range Query algorithm ***RangeQuery*(*k1*, *k2*, *v*)** in the BST ADT in Q2 to get the keys in the range [*k1*, *k2*] in the tree rooted at a node *v*.
2. [10] Print the outputs in 1) where *k1* = 10 and *k2* = 40, i.e. in the range [10, 40], in the BST of Q2.1.

Q4. [20 pt. optional] **Selection in BST**

1. [10] Write an algorithm, ***SelectL*(*i, v*)**, in a ***pseudo code*** to get the *i*th *largest* key of the BST rooted at a node *v*.
2. [10] Implement ***SelectL(i, v)*** in the BST ADT in Q2 and print the 8*th largest key* (*i.e. i=8*)in the BST of Q2.1).