CSci 242: Algorithms and Data Structures

Date: February 7th, 2019

Due: 11:59 PM, February 15th (Sat.), 2020. Name: Elena Corpus

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

Write your answer in blue or in black, not in Red. Please read the submission instruction 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*), 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*.

Algorithm LCA(v,w, r)

r <- root

if r = null then

return null

if r = v or r = w then

return r

node left <- LCA(v,w, r.leftChild)

node right<- LCA(v,w, r.rightChild)

if left != null and right != null

return r

if left = null then

return right

else

return left

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

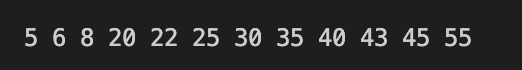
**Implement** a BST ADT with the following algorithms and other required algorithms in Java or in Python and print its output with the given data: *k* – key, *v, w* – node

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

25, 35, 45, 20, 30, 5, 55, 43, 22, 6, 8, 40

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

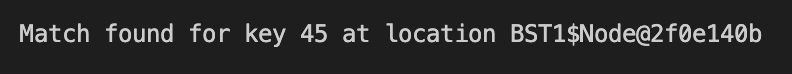
From the created BST in 1)



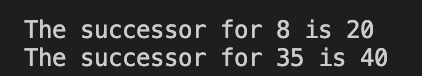
1. [10] *root()* : return a root node *v* and print its key.



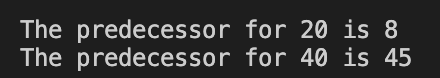
1. [10] *Search(k, v)*: Search/return a key 45 from a BST rooted at *v* and print the returned key.



1. [10] *Successor(v)*: Find a node of immediate successor of a node with (a) a key 8 and print its key. Then, (b) find/print it for a key 35.



1. [10] *Predecessor(v)*: Find a node of immediate predecessor of a node with (a) a key 20 and print its key. Then, (b) find/print the immediate predecessor of a key 40.

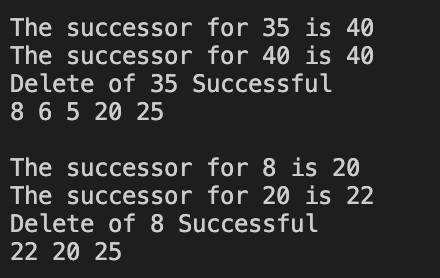


1. [20] *removeAboveExternal*(*w*) remove an external node *w* and its parent node *v*, then reconnect *v*’s parent with *w*’s sibling.

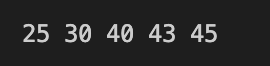
*Remove(k)*: Remove a node of (a) the key 35, then that of (b) the key 5

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

*PostOrder(v)*: Then, (c) print the keys after each deletion in 7) by PostOrder traversal.



1. [10] *rangeQuery*(*k1, k2, v*): find and print the keys in the range of [25, 45].



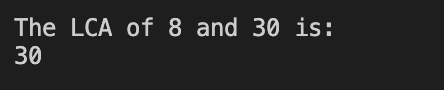
1. [10] *isExternal(v)*: Test whether a node *v* with a key 40 is an external node.



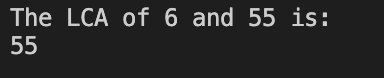
1. [5] *isRoot(v)*: Test whether a node *v* with a key 25 is the root of BST.



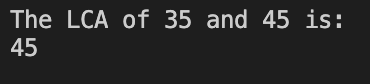
1. [15] Implement the LCA(v, w) algorithm of Q1 and find/print the key of LCA(*v, w*) where
   1. [5] *v.key* = 30, *w.key* = 40



* 1. [5] *v.key* = 6, *w.key* = 55



* 1. [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 whose root is a node *v*..

Algorithm RangeQuery(k1, k2, v)

Output : the numbers within the range given

If current = null,

Return null

If current > k1

Current. Left, k1,k2

If k1 <= current and k2 >= current

Print current

If current < k2

Current. Right, k1, k2

1. [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.

Algorithm Largest (root)

Output: Largest value

While (root != null and rightChild != null) do

Root = rightChild

1. [10] Implement *SelectL(i, v)* in the BST ADT in Q2 and print the *9th largest key* (*i.e. i=9*)in the BST of Q2.1).

