CSci 242: Algorithms and Data Structures

Date: September 17th, 2019

Due: 11:59 PM, September 24th (Tue.), 2019. Name: Elena Corpus

**Home Assignment 2: 150 points + 10 points (optional) 61/150**

Q1. [20] **Binary Tree**

* [10] 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*.

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

LCA(V,W,C); v and w being the targets and C being the root

if c = = null

null

if c = w or c = w

Return c

nodeLeft <-- LCA(v, w, c.leftChild)

Node right < -- LCA(v,w,c.rightChild)

If (left != null AND right != null)

Return c

If (left= null)

Return right

Else

Return left

* [10] What is the running time of your algorithm? Give the asymptotic tight bound (Θ) of its running time in terms of *n* and justify your answer.

O(n) for n is the height of the tree

In this algorithm there was six different comparisons, two different recursive calls, 2 child retrevals, and ultimately one return. Thus concluding that there is a max of 11 operations and a minimum of 2. The equation makes out to be T(n) = (2 if n = 1) or T(n-1) = 11)

T(n) = 11n – 9

O(n) = 11n – 9 : 11n – 9 ≤ cn

Let c = 11 and n0 = 1, then 11n – 9 ≤

11n thus 1 ≤ n

Hence, 11n – 9 = O(n)

Ω(n) = 11n – 9 : 11n – 9 ≥ cn

C > 0 and n0 > n

11n – 9 >= cn : assume c = 1, then 11n – 9 ≥ n

10n ≥ 9

N ≥ 9/10

Q2. [21/110] **(Link-based) Binary Search Tree (BST) (code)**

**Implement** a BST package 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

* [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)

* [0] *root()* : return a root node *v* and print its key.
* [0] *Search(k, v)*: Search/return a key 45 from a BST rooted at *v* and print the returned key.
* [3] *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.
* [3] *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.
* [0] *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.

* [0] *rangeQuery*(*k1, k2, v*): find and print the keys in the range of [25, 45].
* [0] *isExternal(v)*: Test whether a node *v* with a key 40 is an external node.
* [0] *isRoot(v)*: Test whether a node *v* with a key 25 is the root of BST.
* [0] Implement the LCA(v, w) algorithm of Q1 and find/print the key of LCA(*v, w*) where *v* is the node of keys 8 and *w* is the node of key 30.

Q3. [10] **Selection in BST (Code)**

* [10] Write an algorithm, *SelectL*(*i, v*), in a ***pseudo code*** to get the *i*th *largest* key of the BST.

While (root is not null) and (root -> right is not null)

Root = root --> right

Return root-key

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

Q4. [10]**Binary Tree**

For a binary tree T with *n* internal nodes, define the ***internal path length***, *I*(*T*) to be the *sum of the depths of all the internal nodes* in T and the ***external path length***, *E*(*T*) of T be the *sum of the depths of all the external nodes* in T. By Mathematical Induction, prove that if *E*(*T*) = *I*(*T*) + 2*n*.

By inductions on small tree

E(n) - 0, I(0) = 0

E(0) = I(0) + 2.0. is true

Considering a tree, let T1 have l internal nodes and T2 have r internal node

E(l) = I(l) + 2(l)

E(r) = I(r) + 3(r)

Here, n = l + r + 1 (Total number of internal nodes)

E(n) - I(n) =

(E(l) + l + E(r) + 3 + 1) - I(l)

=E(l) - I(l) + E(r) - I(r) + 2

=2(l + r + 1)

= 2n

E(n) - I(n) = 2n

E(n) = I (n) + 2n

We can write

E(t) = I(T) + 2n