Graph Theory-ITGT730E

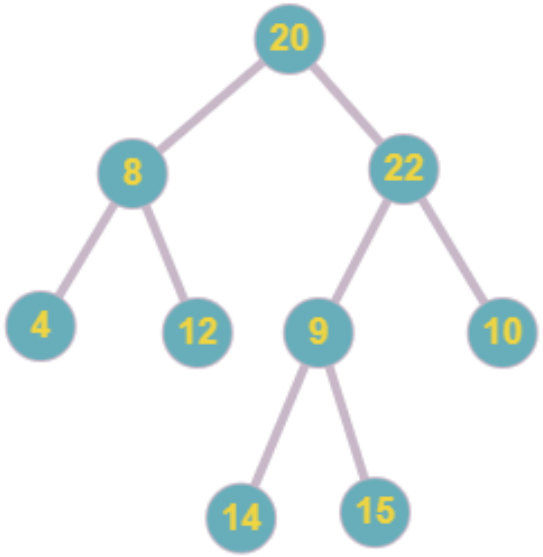
Assignment 3

Path Between two nodes in Full Binary Tree

Tara Prasad Tripathy- IHM2014003
Biki Chaudhary- ISM2014001

➢ Introduction

A full binary tree is a binary tree in which every node has exactly 0 or 2 children. The path between two nodes can be traced efficiently by finding a lowest common ancestor(LCA). The LCA of n1 and n2 in tree is the shared ancestor of n1 and n2 that is located farthest from the root. Some examples of full binary tree and LCA between two nodes are shown below.



LCA (4, 12) = 8
LCA (14, 10) = 22
LCA (4, 14) = 20
LCA (22, 14) = 22

➢ Algorithms

- *Algorithm 1: Brute-force approach*

Input: Full binary tree, node1, node2
Output: Path between node1 and node2

1.) Find path from root to node1 and store the path in array or vector.
2.) Find path from root to node2 and store the path in another array or vector.
3.) Trace both the paths until the values in the array are same.
4.) The common element just before the mismatch is our lowest common ancestor. Return the index of lowest common ancestor.
5.) Print the path from node 1 to LCA and then LCA to node2 which is our required path between two nodes node1 and node2.

Time Complexity: O(n) where n is the number of nodes. The tree is traversed twice and the path arrays are compared.

➢ Algorithm 2: Storing Ancestors

Input: Full binary tree, node1, node2
Output: Path between node1 and node2

1.) Create an empty hash table
2.) Insert node1 and all of its ancestors in hash table.
3.) check node2 or any of its ancestors exists in hash table.
4.) If yes then the first existing ancestor is our lowest common ancestor. Return LCA.
5.) Else, store it in stack and continue from step4.
6.) Finally, print the path from node1 to LCA and then LCA and all the elements from stack.
7.) This is the required path between two nodes node1 and node2.

Time Complexity: O(h) where h is the height of the tree.

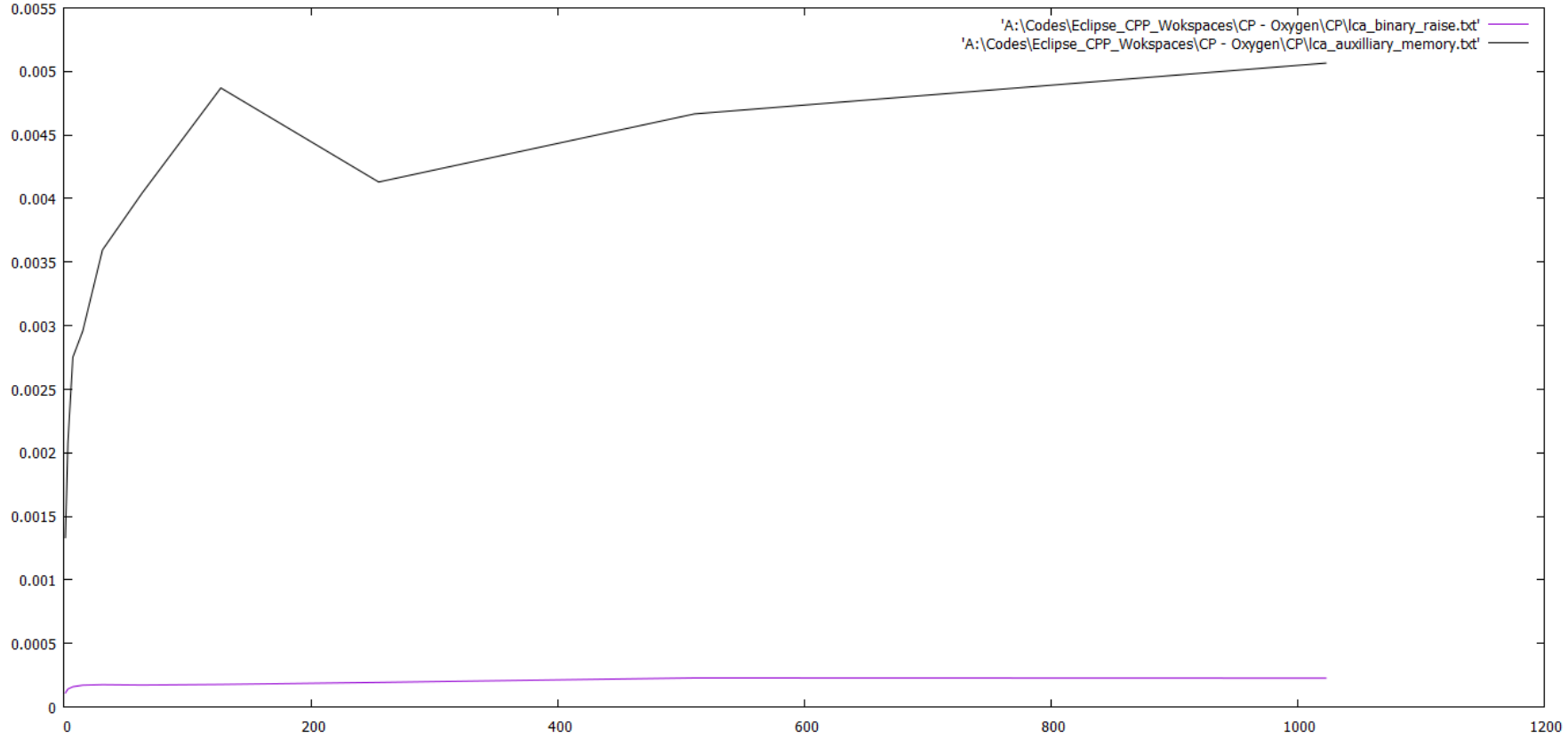
➢ Algorithm 3: Using depth of the tree

Input: Full binary tree, node1, node2
Output: Path between node1 and node2

1) height_l = height(node1), height_r = height(node2)
2) l = node1, r = node2
3) WHILE l is not equal to r
    i. IF height_l >= height_r:
        1. l = PARENT(l)
        2. height_l = height_l - 1
    ii. ELSE:
        1. r = PARENT(r)
        2. height_r = height_r - 1
4) LCA = r
5) RETURN path from node1 to LCA to node2

- *Time Complexity: O(h) where h is the height of the full binary tree.*

Performance Comparison



Thank You