CS2710 - Programming and Data Structures Lab

Lab 9 (graded)

Sept 25, 2024

Instructions

- You are expected to solve ALL the problems in the lab, using the local computer, C++ language, and g++ compiler. (Bonus problems can fetch 5% capped bonus, and Optional problems are just for fun and won't be graded.)
- You should submit your code to the course **moodle** on time, i.e., on/before 4.45pm (so that TAs can subsequently grade your submissions for graded lab assignments using the private test cases).
- You must strictly adhere to the following naming convention for your .cpp files and the single .zip file submission. For example, for a Lab Session 9 consisting of 2 questions, a student with roll number CS23B000 should
  - Name their .cpp files as **CS23B000_LAB9_Q1.cpp** and **CS23B000_LAB9_Q2.cpp**
  - Put both these .cpp files in a directory named **CS23B000_LAB9**
  - Zip this directory into a file named **CS23B000_LAB9.zip**
  - Submit only this single .zip file to moodle.
- If you need assistance, ask your TA, not your classmate.
- Internet access and mobile devices are prohibited in the lab.
- Check course Moodle for the public test cases and the evaluation script, which you can use to test your programs.
- Solve each problem in this lab session using *binary tree or stack or queue* as the primary data structure (do not use any other data structure; string and vector data types are fine for helper or book-keeping tasks - please clarify with the TAs if you have any questions about this).

1. [The right way of looking at a binary tree] Consider a binary tree holding an integer value in each node. Your task is to determine the values of the nodes that would be visible if you were to stand on the right side of the tree and look at it from that perspective.

   The “right-side view” of the tree consists of the rightmost nodes at each depth level, starting from the root. In other words, for each level of the tree, if you look at it from the right side, you can only see one node – the rightmost one. The goal is to output a list of these values ordered from top to bottom, reflecting how you would see them in real life if the tree were laid out in front of you.

   Input Format:

   - Input is a single string – contains binary tree in Newick format (note: empty tree is given by the string ”” or ”-”).

   Output Format:

   - Print values of the nodes which are visible from right side of the tree, starting from the root and going down.

   Constraints:

   - Input string contains (,), -, and non-negative integers
   - Number of nodes in the tree in [0,1000]
   - Nodes values are in [0,1000]

   Examples:

Input1:
(((-,5)4,-)2,3)1

Output1:
1 3 4 5

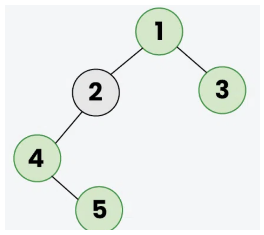

Figure 1: Binary tree for input1

Input2:
(2,(4,5)3)1

Output2:
1 3 5

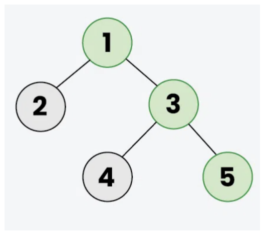

Figure 2: Binary tree for input2

2. [Diameter of a binary tree] The diameter of a tree is defined as the maximum distance between two nodes in the tree, where distance between two given nodes is defined as the number of edges on the path connecting the two nodes. As shown in the figure below, the longest path corresponding to the maximum distance in a binary tree does not necessarily pass through the root node; in some cases, the longest path may span across nodes entirely contained within one of the subtrees.

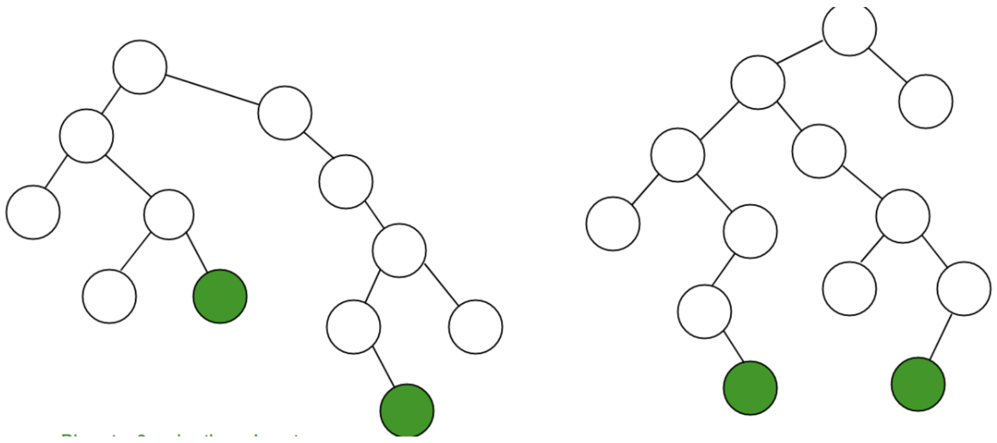

Figure 3: The longest path passes through the root in the tree shown on left, and doesn't involve the root in the tree shown on right.

Input Format:

- Input is a single string – contains tree in Newick format (note: empty tree is given by the string "" or "-").

Output Format:

- The diameter of the given binary tree.

Constraints:

- Input string contains (,), -, and non-negative integers
- Number of nodes in the tree in [0,1000]
- Nodes values are in [0,1000]

Examples:

Input1:
((40,60)20,30)10
Output1:
3

Input2:
((-,40)20,(-,50)30)10
Output2:
4

3. [BONUS: LEAST COMMON ANCESTOR] You need to find the LCA (Least Common Ancestor) of two given nodes in a general tree (i.e., the tree need not be binary). Given two nodes n1 and n2, the LCA is the deepest node in the tree that has both n1 and n2 as descendants. In other words, LCA of n1 and n2 is the shared ancestor of n1 and n2 that is located farthest from the root. You have to print the LCA of n1 and n2 if both these nodes exists in the tree, otherwise print -1.

- LCA of 50 and 30 is 10
- LCA of 70 and 30 is 30
- LCA of 60 and 50 is 20

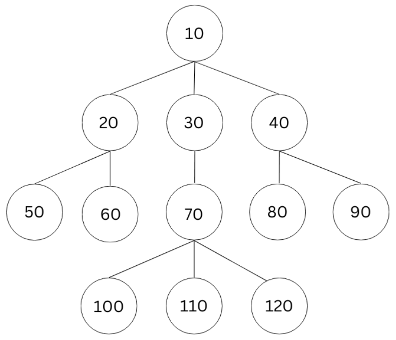

Figure 4: LCA of nodes

- LCA of 4 and 5 is 2
- LCA of 5 and 6 is 1
- LCA of 7 and 2 is 1

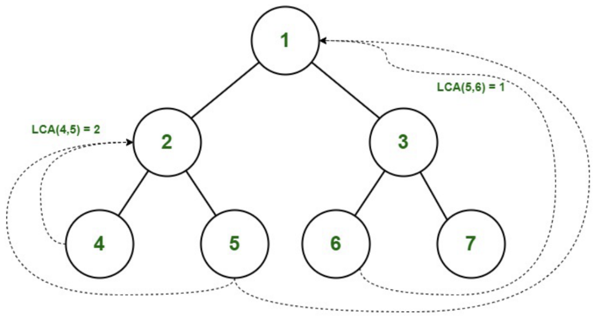

Figure 5: LCA of nodes

Input Format:

- First line of input is a string - contains (general) tree in Newick format
- Second line will contain two space-separated integers: n1 and n2 (values of the two nodes)

Output Format:

- Output is a single integer – value of the LCA node

Constraints:

- Input string contains (,), and non-negative integers
- Node values are unique, and are in [0,1000]
- Number of nodes in the tree in [0,1000], and number of children of a node in [0,100]
- n1, n2 in [-1000,1000]
- **You are NOT allowed to use a parent pointer** inside your tree node class. Only a vector of child pointers are allowed inside a tree node.

Examples:
Input1: ((4,5,6)2,(7,8)3)1
5 6
Ouput1:
2

Input2:
((4,5,6)2,(7,8)3)1
4 8
Output2:
1

Input3:
((4,5,6)2,(7,8)3)1
-100 8
Output3:
-1

---

4. [OPTIONAL: AVERAGE DISTANCE OF A BINARY TREE] While diameter is the maximum distance between two nodes (or equivalently two leaf nodes) in a tree, one may be interested in finding the average distance between two leaf nodes in a tree (i.e., the sum of distances between each pair of leaves, divided by $\binom{n_l}{2}$), where n_l is the number of leaves in the tree). What is the most efficient algorithm you can think of and code for this problem? You can assume the tree is a binary tree.