



Informatics II

Exercise 8

April 6, 2020

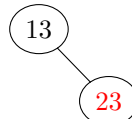
Binary Search Trees

Task 1. Binary search tree is created by inserting nodes 13, 23, 17, 40, 16, 3, 10, 5, 2, 49, 20. Show structure of the tree after insertion of each node.

Insert 13



Insert 23



Task 2. Considering binary search tree created in **Task 1**, show the structure of the tree after deleting each of the following nodes 5, 40, 17, 23, 13.
(When the node that is going to be deleted has two children, find the child with the largest value in the left subtree.)

Task 3.

```
1 struct TreeNode {  
2     int val;  
3     struct TreeNode* left;  
4     struct TreeNode* right;  
5 };
```

Given the above definition of a binary search tree with positive integer values, create a C program that contains the following functions:

- the function `void insert(struct TreeNode** root, int val)`, which gets a root node and a value `val` as an input and inserts a node with the value `val` into the proper position of the binary search tree.



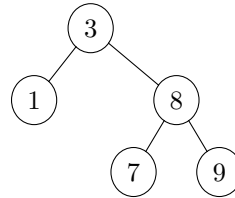
- b) the function `struct TreeNode* search(struct TreeNode* root, int val)`, which finds the tree node with value `val` and returns the node.
- c) the function `void delete(struct TreeNode** root, int val)`, which deletes the node with value `val` from the tree.
- d) `void printTree(struct TreeNode* node)` which prints the tree with root `root` in the console in the format `graph g { (all the edges in the form NodeA -- NodeB) }`, where each edge should be printed on a separate line. The ordering of the edges is not relevant and may vary based on your implementation.
- e) `struct TreeNode* maximum(struct TreeNode* node)`, which returns the node with the largest value in the subtree with root node `n`.
- f) `struct TreeNode* minimum(struct TreeNode* node)`, which returns the node with the smallest value in the subtree with root node `n`.
- g) `int distanceToRoot(struct TreeNode* root, int val)` that returns the distance of the node with value `val` from the root node `root`.

For example, if the values 3, 8, 7, 1 and 9 are inserted into an empty tree, your program should produce the binary tree shown below.

Print Form

```
graph g {  
    3 -- 1  
    3 -- 8  
    8 -- 7  
    8 -- 9  
}
```

Binary Tree Form



Test your program by performing the following operations:

- Create a root node `root` and insert the values 4, 2, 3, 8, 6, 7, 9, 12, 1.
- Print tree to the console.
- Print the minimum value of the tree.
- Print the distanceToRoot of node 7.
- Delete the values 4, 12, 2 from the tree.
- Print tree to the console.
- Print the maximum value of the tree.
- Print the distanceToRoot of node 6.



Task 4. Given a rooted tree T , the *lowest common ancestor (LCA)* between two nodes $n1$ and $n2$ is defined as the lowest node in T that has both $n1$ and $n2$ as descendants (where we allow a node to be a descendant of itself). Consequently, the LCA of $n1$ and $n2$ in T is the shared ancestor of $n1$ and $n2$ that is located farthest from the root.

For example, given a tree as in Figure 1, the LCA of nodes 9 and 21 is node 15.

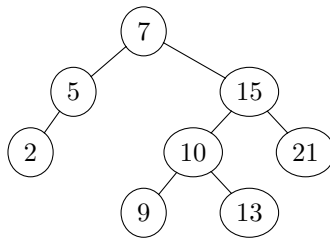


Figure 1: LCA of nodes 9 and 21 is node 15

Given values of two nodes in a Binary Search Tree, implement in C the function `struct TreeNode* lca(TreeNode* root, int n1, int n2)` that finds the *Lowest Common Ancestor (LCA)*. You may assume that both the values exist in the tree.

Write in C a program to test your implementation by performing the following operations:

- Create an empty tree and insert the nodes 7, 5, 6, 1, 9, 10, 8.
- Print the resulting tree using method described in **Task 3**.
- Print out the LCA of node 8 and node 9