Informatics II, Spring 2023, Exercise 8

Publication of exercise: April 24, 2023
Publication of solution: April 30, 2023
Exercise classes: May 2 - 5, 2023

Binary Tree

Task 1

1. Which statements of the followings are correct about binary tree:

A. There exists a binary tree satisfying the following conditions: i. having more than one node, ii. the value of each node is unique, iii. the results of the three traversals (pre/in/post-order) are the same.

B. When applying three traversals (pre/in/post-order) on any binary tree, the orders of appearance of its leaf nodes are the same.

C. For a binary search tree, if the root node doesn't have left child, the root must be the node with the smallest value.

D. If we list the values of a binary tree's nodes according to its inorder traversal, it will be in ascending order.

2. Given that the preorder and inorder traversal of a binary tree is ABDEGCF and DBGEACF, find its postorder traversal.

A. DGEBFCA

B. ACFBEGD

C. FCAEGBD

D. FCGEDBA

3. Create a binary search tree and insert 1,7,4,6,5,3,8 sequentially. After delete 7,6 sequentially, find its postorder traversal. (In the deletion, we choose to find the largest value in the left subtree.)

A. 34851

B. 15438

C. 13458

D. 83451

4. Create a binary search tree for 5 unique numbers. How many different trees can be obtained.

A. 42

B. 14

C. 21

D. 5

Task 2

The structure of the tree node is defined as follow. The entry to a binary search tree is the root that is also a tree node. It's also used in **Task 3**.

```
struct TreeNode {
  int val;
  struct TreeNode* left;
  struct TreeNode* right;
};
```

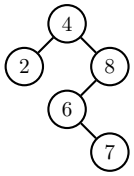
Write a C program that contains the following functions:

a) Write the function *void insert(struct TreeNode** root, int val)* that inserts an integer `val` into the binary search tree. Note that, you need to create a tree node for `val`, and find the correct position to insert the new tree node.

b) Write the function *void delete(struct TreeNode** root, int val)* that deletes the node with value `val` from the tree. (In the deletion, we choose to find the largest value in the left subtree.)

c) Write *void traverseTree(struct TreeNode* root)* which prints the results of pre/in/post-order of the tree in the console in separate lines.

d) Write *void printTree(struct TreeNode* root)* which prints all edges with their *level* of the tree from root in the console in the format `Node A -- Node B : level`, and each edge is printed in a separate line. The ordering of the printed edges does not matter and may vary based on your implementation. Here, the *level* of an edge is defined by the larger depth of the two adjacent nodes.



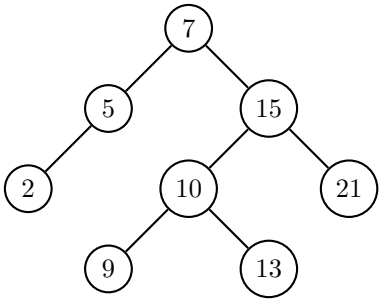
Output of *printTree* for the example above should be:
4 −− 2 : 1
4 −− 8 : 1
8 −− 6 : 2
6 −− 7 : 3

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 traversals to the console.
- Delete the values 4, 12, 2 from the tree.
- Print tree to the console.
- Print traversals to the console.

Task 3

Given a rooted binary tree T, the `largest root-leaf path (LRLP)` of T is defined as a straight path from the root to a leaf in T, which has the largest `sum` of values.



For example, the `LRLP` of the tree above should be `7--15--10--13`, and the `sum` is 45.
Given a binary search tree, implement in C the function *void lrlp(struct TreeNode* root)* that print the `LRLP` and its sum of the tree in the console. If there are multiple `LRLP`, just print one of them.
Write in C a program to test your implementation by performing the following operations:

- Create an empty binary search tree and insert the nodes 7, 5, 2, 15, 21, 10, 9, 13 using method described in **Task 2**.
- Print out the `LRLP` and its `sum` of the tree.