CENG 301, Spring 2024
Homework Assignment 3

Due: 23:59, May 20, 2024

In this homework, you will implement a special kind of binary tree where each value appears at most once. This special binary tree will have the following functionalities. The details of these functionalities are given below:

1. Add a new node
2. Update a node's value
3. Remove a node
4. Check whether a value exists in the tree or not
5. Find the height of the node with given value
6. Find the depth of the node with given value
7. Perform preorder traversal
8. Perform inorder traversal
9. Perform postorder traversal
10. Find sum of values all nodes in a subtree
11. Find the minimum value in a subtree
12. Find the maximum value in a subtree

Add a new node: Given the value of the parent node and whether the newly added node is the right child or the left child of its parent, add the new node to the tree. If the value will be the root, it will be mentioned with a special boolean value as a parameter. In this case, the parent value will .

Since the values must be unique, the system must check whether or not the specified value already exists in the tree. If it exists, the operation must not be allowed. If the parent value does not exist in the tree and the node is not the root, the system must not allow the operation. If the parent node already has the corresponding child (e.g., if the newly added node will be a left child but the parent already has a left child), the system must not allow the operation. If the node is the root but a root node already exists, the operation must not be allowed.

The corresponding function will return:

- false if a node with the same value already exists
- false if the parent node does not exist and the node is not the root
- false if the parent node already has the corresponding child
- false if the node is the root but a root node already exists
- true if the operation is successful

Update a node's value: Given a previous value and a new value, find the node having the previous value and change its value with the new value. If the previous value does not exist in the special binary tree, the system must not allow the operation.

The corresponding function will return:

- false if the previous value does not exist
- true if the operation is successful

Remove a node: The removal of a node in the special binary tree follows a specific algorithm. When a node is deleted, its value is replaced with its left child's value (if the left child exists); if not, it's replaced with its right child's value. This replacement process continues recursively until a leaf node is reached. Once a leaf node is encountered, it is entirely removed from the tree.

The corresponding function will return:

- false if the node to be removed does not exist
- true if the removal operation is successful

Check whether a value exists in the subtree rooted with the given value or not: Given a value to search and a subtree root value, check whether it is the value of any node in the subtree rooted with the given value or not.

The corresponding function will return:

- false if such subtree does not exist or the value does not exist in the subtree
- true if the value exists in the tree

Find the height of the node with the given value: Height refers to the length of the longest path from the node in question to a leaf node in the subtree rooted at the node in question. Note that the height of a leaf node is always 0.

The corresponding function will return:

- -1 if a node with the given value does not exist
- The height of the node as an integer if the node with the given value exists

Find the depth of the node with the given value: Depth refers to the level of a node within a tree structure. It is defined as the number of edges from the root node to the node in question. Note that, depth of the root node is always 0.

The corresponding function will return:

- -1 if a node with the given value does not exist
- The depth of the node as an integer if the node with the given value exists

Perform preorder traversal: Perform preorder traversal in the tree. If there is no node in the tree, print "There is no node in the tree. " Otherwise, print out the values separated by spaces. For both cases, at the end of the output, there should be a space and a newline.

Example output 1:
1 2 3 4
Example output 2:
There is no node in the tree.

Perform inorder traversal: Perform inorder traversal in the tree. If there is no node in the tree, print "There is no node in the tree. " Otherwise, print out the values separated by spaces. For both cases, at the end of the output, there should be a space and a newline.

Example output 1:
1 2 3 4
Example output 2:
There is no node in the tree.

Perform postorder traversal: Perform postorder traversal in the tree. If there is no node in the tree, print "There is no node in the tree. " Otherwise, print out the values separated by spaces. For both cases, at the end of the output, there should be a space and a newline.

Example output 1:
1 2 3 4
Example output 2:
There is no node in the tree.

Check whether the node with the given value is a leaf node: Check whether the node with the given value is a leaf node or not. Return an integer value as follows:

- -1 if the node does not exist
- 0 if the node is a non-leaf node
- 1 if the node is a leaf node

Find the sum of values of all nodes in a subtree: Find and return the sum of values of all nodes in the subtree rooted at the node with the given value.

The corresponding function will return:

- -1 if the node with the given value does not exist
- The sum of values of all nodes in the subtree rooted at the node with the given value, if the node with given value exists

Find the minimum value in a subtree: Find and return the minimum value in the subtree rooted at the node with the given value.

The corresponding function will return:

- -1 if the node with the given value does not exist
- The minimum value in the subtree rooted at the node with the given value if the node exists

Find the maximum value in a subtree: Find and return the minimum value in the subtree rooted at the node with the given value.

The corresponding function will return:

- -1 if the node with the given value does not exist
- The maximum value in the subtree rooted at the node with the given value if the node exists

Below is the headers for `BinaryTreeNode` and `SpecialBinaryTree`, classes that you must write in this assignment. You should implement them in separate .cpp files.

```
// BinaryTreeNode.h
// DO NOT MODIFY THIS FILE

#ifndef BINARY_TREE_NODE_H
#define BINARY_TREE_NODE_H
class BinaryTreeNode
{
private:
```

```
    int value;
    BinaryTreeNode *left;
    BinaryTreeNode *right;

public:
    BinaryTreeNode();
    ~BinaryTreeNode();
    int get_value();
    void set_value(int value);
    BinaryTreeNode* get_left();
    void set_left(BinaryTreeNode* left);
    BinaryTreeNode* get_right();
    void set_right(BinaryTreeNode* right);
};
#endif
```

```
// SpecialBinaryTree.h
// DON'T CHANGE THE EXISTING FUNCTION SIGNATURES
// IF YOU NEED, YOU CAN ADD NEW FUNCTIONS TO SpecialBinaryTree CLASS

#ifndef SPECIAL_BINARY_TREE_H
#define SPECIAL_BINARY_TREE_H

class BinaryTreeNode; // This is called a "forward decleration"

class SpecialBinaryTree
{
private:
    BinaryTreeNode *root;
public:
    SpecialBinaryTree();
    ~SpecialBinaryTree();
    bool add_node (int new_value, int parent_value, bool is_left_child, bool
        is_right_child, bool is_root);
    bool update_value (int previous_value, int new_value);
    bool remove_node (int value);
    int is_leaf_node (int value);
    bool does_exist_in_subtree(int subtree_root_value, int value_to_search);
    int find_depth (int value);
    int find_height (int value);
    void preorder_traversal();
    void postorder_traversal();
    void inorder_traversal();
    int find_sum_of_values_in_subtree(int subtree_root_val);
    int find_minimum_value_in_subtree(int subtree_root_val);
    int find_maximum_value_in_subtree(int subtree_root_val);
};
#endif
```

Here is an example test program that uses this class and the corresponding output. Your implementation should use the format given in the example output to display the messages expected as the result of the defined functions.

Example test code:

```
#include<iostream>
#include<string>
#include "SpecialBinaryTree.h"

using namespace std;

int main() {

    string truth_table[2] = {"false", "true"};

    SpecialBinaryTree t;

    cout<<truth_table[t.add_node( 8, -1, false, false, true)]<<endl; // we are adding
        the root node
    cout<<truth_table[t.add_node( 5, -1, false, false, true)]<<endl; /* we can't add a
         second root while there is a current root */
    cout<<truth_table[t.add_node( 3, 8, true, false, false)]<<endl;
    cout<<truth_table[t.add_node( 5, 8, false, true, false)]<<endl;
    cout<<truth_table[t.add_node( 1, 5, true, false, false)]<<endl;
    cout<<truth_table[t.add_node( 2, 5, false, true, false)]<<endl;
    cout<<truth_table[t.add_node( 4, 3, true, false, false)]<<endl;
    cout<<truth_table[t.add_node( 5, 3, false, true, false)]<<endl; /* operation
        should not be allowed since there is already a node with value 5 */
    cout<<truth_table[t.add_node( 6, 3, false, true, false)]<<endl;
    cout<<truth_table[t.add_node( 11, 5, false, true, false)]<<endl; /* operation
        should not be allowed since the node with value 5 already has a right child */
    cout<<endl;

    cout<<truth_table[t.remove_node( 5 )]<<endl;
    cout<<truth_table[t.remove_node( 11 )]<<endl; /* operation should not be allowed
        since there is no node with value 11 */
    cout<<truth_table[t.update_value(8, 6)]<<endl; /* operation should not be allowed
        since there is already a node with value 6, therefore 6 cannot be the new
        value of a node */
    cout<<truth_table[t.update_value(8, 9)]<<endl;
    cout<<endl;

    t.preorder_traversal();
    cout<<endl;
    t.postorder_traversal();
    cout<<endl;
    t.inorder_traversal();
    cout<<endl;

    cout<<t.is_leaf_node( 5 )<<endl; /* Since there is no node with value 5 (node with
         value 5 is removed in one of the previous operations), return value shoud be
        -1 */

    cout<<t.is_leaf_node( 3 )<<endl; /* Since the node with value 3 is not a leaf node
        , return value should be 0 */

    cout<<t.is_leaf_node( 2 )<<endl; /* Since the node with value 2 is a leaf node,
        return value shoud be 1 */
    cout<<endl;
```

```
    cout<<truth_table[t.does_exist_in_subtree(1,1)]<<endl;
    cout<<truth_table[t.does_exist_in_subtree(1,2)]<<endl;
    cout<<truth_table[t.does_exist_in_subtree(1,9)]<<endl; /* Since the node with
        value 9, is not in the subtree rooted at the node with value 1, return value
        should be false */
    cout<<truth_table[t.does_exist_in_subtree(1,10)]<<endl; /* Since there is no node
        with value 10, return value should be false */
    cout<<endl;

    cout<<"Depth of node with value 6: " << t.find_depth(6)<<endl;
    cout<<"Height of node with value 6: " << t.find_height(6)<<endl;
    cout<<endl;

    cout<<"Sum of values: " << t.find_sum_of_values_in_subtree(9)<<endl;
    cout<<"Minimum value: " << t.find_minimum_value_in_subtree(9)<<endl;
    cout<<"Maximum value: " << t.find_maximum_value_in_subtree(9)<<endl;

    return 0;
}
```

Output of the example test code:

```
true
false
true
true
true
true
true
false
true
false

true
false
false
true

9 3 4 6 1 2

4 6 3 2 1 9

4 3 6 9 1 2

-1
0
1

true
true
false
false

Depth of node with value 6: 2
Height of node with value 6: 0
```

```
Sum of values: 25
Minimum value: 1
Maximum value: 9
```

IMPORTANT NOTES:
Do not start your homework before reading these notes!!!

NOTES ABOUT IMPLEMENTATION:

1. You ARE NOT ALLOWED to modify the given parts of the header file. You MUST use binary tree in your implementation. You will get no points if you use dynamic or fixed-sized arrays or any other data structures such as vectors/arrays/lists/sets/maps from the standard library.
2. Moreover, you ARE NOT ALLOWED to use any global variables or any global functions.
3. Output message for each operation MUST exactly match the format shown in the output of the example code. Otherwise, you cannot receive points.
4. Using STL (Standard Template Library) is strictly forbidden. If you use STL, you get a direct 0.
5. Cheating and any type of plagiarism is strictly forbidden. If you cheat and/or plagiarise, you get a direct 0.

NOTES ABOUT SUBMISSION:

1. In this assignment, you must have separate interface and implementation files (i.e., separate `.h` and `.cpp` files) for your class. Your class names MUST BE `SpecialBinaryTree`, `BinaryTreeNode`. Your file names MUST BE `SpecialBinaryTree.h`, `SpecialBinaryTree.cpp`, `BinaryTreeNode.h`, `BinaryTreeNode.cpp`. You will not submit `BinaryTreeNode.h`, since you should not modify anything in this file.
2. The code (`main` function) given above is just an example. We will test your implementation using different scenarios, which will contain different function calls. Thus, do not test your implementation only by using this example code. We recommend you to write your own driver files to make extra tests. However, you MUST NOT submit these test codes (we will use our own test code). In other words, do not submit a file that contains a function called `main`.
3. In the "edit" section of VPL, you can run your code with a few example test cases. You can see whether you pass or fail them. We recommend you to use it. It may help you pinpoint the errors in your codes.
4. This assignment is due by 23:59 on Monday, May 20, 2024. You should upload your work to the corresponding VPL activity in ODTUCLass before the deadline. No hardcopy submission is needed.
5. This homework will be graded by your TA **Burak Ferit Aktan** (aktan@ceng.metu.edu.tr). Thus, you may ask your homework related questions directly to him. There will also be a forum on ODTUClass for questions.