CSE2122 (Spring 2020)
Homework 5, due 3/20/2020 11:59pm

Before you start, please login your stdlinux account and download the template program from the class directory to your stdlinux directory. (There is no solution file for this assignment.) You can use the command "`cp /class/cse2122/hw5/* ./.`".

Submit your code to the designated dropbox on Carmen under the Assignments tab. Late submissions receive a 10% penalty for each day. No late submissions after the second day.

Skills needed to complete this assignment: linked lists, recursive algorithms.

In computer science, a tree is a widely used data structure that simulates a hierarchical diagram. A tree data structure can be defined as a collection of nodes (starting at a root node), where each node is a data structure consisting of a value and a list of references to nodes, with the constraints that no reference is duplicated, and none points to the root.

In this assignment, we will focus on binary trees. In a binary tree, each node has at most two children, which are referred to as the left child and the right child. Figure 1 shows an example of a binary tree.

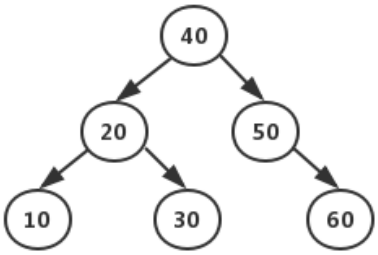


Figure 1 A Binary Tree Example

To implement binary trees, we start from the definition of a single node. Since each node consists of a value and at most two references to other nodes, we can define a binary tree node as

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

We will use a node's left and right pointers to point to its children nodes on either side. If a tree node has no left or right child, we set the corresponding pointer to `NULL`. Following figure shows how the binary tree in Figure 1 can be implemented with the `TreeNode` structure.

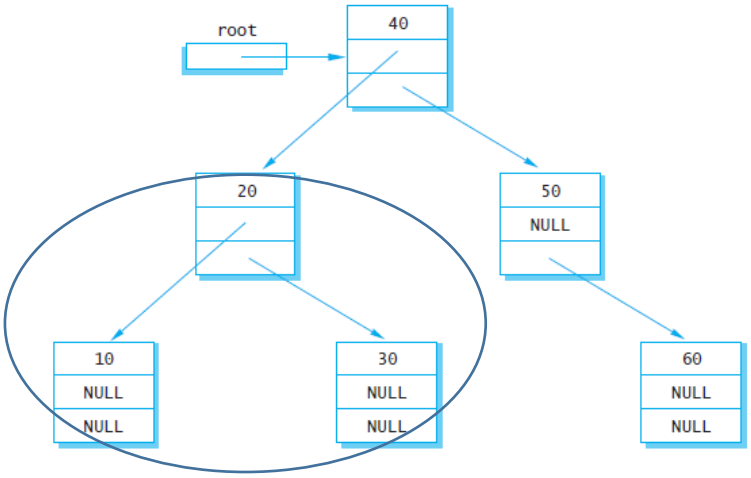


Figure 2 A Binary Tree using `TreeNode` structure (from textbook Display 13.12)

The topmost node in a tree is called the root node. The nodes that do not have child nodes, such as nodes 10, 30, and 60 in figure 2, are called leaf nodes.

We can use only one pointer, the root pointer, to represent a tree. The root pointer is a pointer pointing to the root node in the tree. If a tree is empty, the root pointer is set to `NULL`. For example, in figure 2, a reference to the node with value 20 can be used to represent the circled tree. Starting from the root pointer, we can easily find all nodes in a tree.

Reconsider the meaning of a node's `left` and `right` pointers. If a reference to a node can be regarded as a representation of a tree with this node as the root, we can say that a node's `left` and `right` pointers recursively point to smaller 'subtrees' on either side.

Thus, a recursive definition of the binary tree can be: a binary tree is either empty (represented by a `NULL` pointer), or is made of a single node, where the left and right pointers (recursive definition ahead) each point to a binary tree.

In this assignment, you will write recursive functions as follows. The input binary trees for all functions consist of nodes with non-negative values. You must implement these functions **recursively** to get full credits.

- `int size(TreeNode* root)`
  Given the root node of a binary tree, count the number of nodes in the tree.

- `int count(TreeNode* root, int target)`
  Given the root node of a binary tree and a target value, count the number of occurrences of the target value in this tree.

- `int height(TreeNode* root)`
  The height of a tree is the number of edges on the longest path between the root node and a leaf. Given the root node of a binary tree, this function returns the height of this tree. If the binary tree is empty, the function should return -1.

- `bool isSameTree(TreeNode* root1, TreeNode* root2)`
  Given the root nodes of two binary trees, determine whether these trees are the same.

- `bool hasPathSum(TreeNode* root, int target)`
  Define a "root-to-leaf path" to be a sequence of nodes in a tree starting with the root node and proceeding downward to a leaf. Given the root node of a binary tree and a target sum, return true if the tree has a root-to-leaf path such that adding up all the values along the path equals the given sum, return false if no such path can be found. If the binary tree is empty, the function should return true only if the target sum is 0.

- `bool isBalanced(TreeNode* root)`
  We say a binary tree is balanced if the heights of the two subtrees of every node in this tree never differ by more than 1. Given the root node of a binary tree, this function will determine if this tree is balance or not.

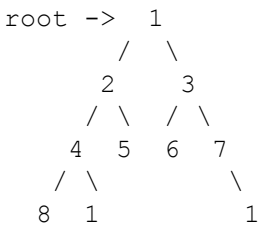
Each function is worth 20 points. You can implement any five functions properly to get the full credit. If you implement all six functions, you can earn up to 20 points bonus.

Do not use C/C++ libraries other than `iostream`, `cstddef`, and `cassert`. Do not change the function declarations.

You are free to modify the main function to add your own test cases or to define new helper functions.

Examples

The template program provides two hard-coded binary trees and some test cases (same as the following examples). Test cases are provided using assertion in the main function. If your program passes all the test cases, a message "Congratulation!" will be printed; otherwise your program will stop.



`size(root)` is `10`.
`size(root->left)` is `5`.

`count(root,1)` is `3`.
`count(root,9)` is `0`.

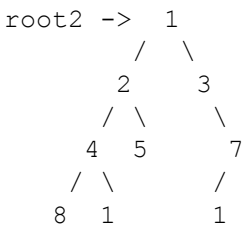
`height(root)` is `3`.
`height(root->left->right)` is `0`. (`root->left->right` points to the leaf node '5'.)

'Root-to-leaf' paths:
In this tree there are 5 'root-to-leaf' paths:
`1. 1->2->4->8`

```
2. 1->2->4->1
3. 1->2->5
4. 1->3->6
5. 1->3->7->1
```

`hasPathSum(root, 10)` returns `true` because in Path 4, 1+3+6=10.
`hasPathSum(root, 1)` returns `false`.

`isBalanced(root)` is `true`, because for every node in this tree, the difference between the heights of its two subtrees is either 0 or 1.



`isSameTree(root, root2)` is `false`.
`isSameTree(root->left, root2->left)` is `true`.
`isBalanced(root2)` is `false`. That's because for the node 3, its left subtree is of height -1, and its right subtree is of height 2. The difference is greater than 1.

Program Submission

Compile your code with the following command. Submissions do not compile with the following command will receive zeros.

```
g++ <your file name> -std=c++11 –pedantic-errors
```

Make sure your code is formatted properly. Up to 5% of the assignment credit can be deducted if your program does not have appropriate indention and necessary comments. See textbook on section 2.5 or any code example in textbook/lecture slides.

Submit your code (the .cpp file) to the designated dropbox on Carmen. You can find the dropbox under the Assignments tab.