CSC 228 Data Structures and Algorithms, Spring 2019
Instructor: Dr. Natarajan Meghanathan

EXAM 2

Due: March 29th, 11.59 PM (submission through Canvas)

Submission: Submit a single word document that has the code and the answers for all the questions, clearly marking the question numbers for each code/answer.

Q1 - 25 pts) A binary tree is a complete binary tree if all the internal nodes (including the root node) have exactly two child nodes and all the leaf nodes are at level 'h' corresponding to the height of the tree.

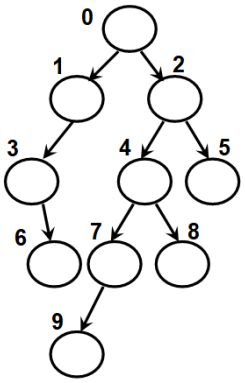
Consider the code for the binary tree given to you for this question. Add code in the blank space provided for the member function checkCompleteBinaryTree( ) in the BinaryTree class. This member function should check whether the binary tree input by the user (in the form of the edge information stored in a file) is a complete binary tree.

To test your code, come up with two binary trees of at least 12 vertices: one, a complete binary tree and another, a binary tree that is not a complete tree.

Prepare the input file for the two binary trees and input them to the code for this question. Capture the screenshots of the outputs.

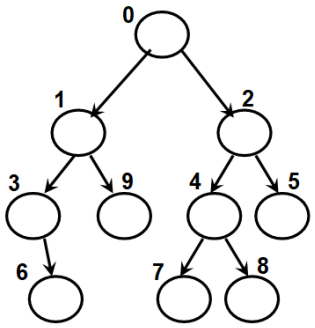
What to submit for Q1:
(i) The complete C++ Code
(ii) Draw the two binary trees (along with their node ids) that you have come up with, include the contents of the text files (corresponding to these two binary trees) that are input to the program as well as screenshots of the outputs.

Q2 - 25 pts) In this question, you will write the code to determine whether a binary tree input by the user (in the form of an edge file, as discussed in the slides/class) is height-balanced or not. A binary tree is said to be height-balanced if each internal node (including the root) in the tree is height-balanced. A node is said to be height-balanced if the absolute difference in the heights of its left sub tree and right sub tree is at most 1. The binary tree (a) below is not height-balanced (as nodes 1 and 2 are not balanced), whereas the binary tree (b) below is height-balanced.



Node	Height of Left subtree	Height of Right subtree	Abs. Diff.	Height Balanced
0	2	3	1	YES
1	1	-1	2	NO
2	2	0	2	NO
3	-1	0	1	YES
4	1	0	1	YES
7	0	-1	1	YES

(a) A Binary Tree that is "not" Height-Balanced



Node	Height of Left subtree	Height of Right subtree	Abs. Diff.	Height Balanced
0	2	2	0	YES
1	1	0	1	YES
2	1	0	1	YES
3	-1	0	1	YES
4	0	0	0	YES

(b) A Binary Tree that is Height-Balanced

Note that the height of a leaf node is 0 and the height of a non-existing tree (or sub tree) is -1.

You are given the code for the binary tree implementation discussed in class. You could first find the height of each node in the tree and then test for each internal node: whether the difference in the height of its left child and right child is at most 1. If this property is true for every internal node, then we exit the program by printing the binary tree is indeed height-balanced.

Come up with files for storing the edge information of the two binary trees (a) and (b), and demonstrate the execution of your code to determine whether a tree is height-balanced or not.

Add any member variable (an array) to keep track of the height of the individual nodes in the tree as well as use the information in this array to determine whether the binary tree is height-balanced or not.

What to submit for Q2:
(i) Complete code of the binary tree program, extended to determine if the tree is height-balanced or not.
(ii) Screenshot of the outputs when the program is run for the above two binary trees (a) and (b).

Q3 - 25 pts) Use a character-based Stack to print a string input by the user in the reverse order. To test your program, enter your full name, including middle initial, if any (say, **John S. Peter**) and the program should print **reteP .S nhoJ** as output.

Note that you should not use any built-in function of the string library (like reverse(..)) to accomplish this.

You need to use the 'doubly linked list'-based implementation of the stack. **You could use any of the code provided by the instructor (in Canvas) for modules 3-4**, modify and integrate them as needed to accomplish the above task.

What to submit for Q3:
(i) Complete C++ Code: the Node class, the List class (Doubly Linked List), the Stack class and the main function
(ii) Screenshot of the output that prints your full name (input) in reverse order.

Q4 - 25 pts) You are given the code for storing integer elements of an array in a Hashtable. Modify the code in such a way that you could **print the number of occurrences of every unique element in the array**. You could modify any class (including the Node class) as well as add suitable member functions and/or modify existing member functions, if needed. You need to also extend the main function to print the number of occurrences of every unique element in the array.

A sample screenshot of the required output is shown below. No

```
Enter the number of elements you want to store in the hash table: 25
Enter the maximum value for an element: 15
Enter the size of the hash table: 7
Elements generated: 0 6 12 1 9 8 4 2 7 11 6 5 7 5 14 1 8 7 9 14 6 0 2 10 8

Elements and their number of occurrences
0, 2
7, 3
14, 2
1, 2
8, 3
9, 2
2, 2
10, 1
4, 1
11, 1
12, 1
5, 2
6, 3
```

What to submit for Q4:
(i) Complete C++ Code: the Node class, the List class (Singly Linked List), the Hashtable class and the main function
(ii) Screenshot of the output wherein the inputs are as follows: number of elements to store in the hash table is 30, the maximum value for an element is 20 and the size of the hash table is 11.