CISC-235 Data Structures
Assignment 2

October 2, 2019

In this assignment, you will implement a binary tree structure and associated functions. The goal of this assignment is to:

1. get familiar with the implementation of a binary tree.
2. implement and use stack.
3. get familiar with operations on a binary tree.

1 Implementing Stack Using Array-based List: 10 points

Create a class named **Stack** using array-based list provided by a particular programming language (e.g., List in Python). Your class must contain an initialization function, as well as the following functions:

- **isEmpty**, this function checks whether the current Stack instance is empty. It should return true if the Stack is empty.
- **push**, this function should takes a data item as input and add it into the Stack.
- **pop**, this function should return the data item on the top of the current Stack and remove it from the Stack.
- **top**, this function should return the data item on the top of the current Stack without modifying the Stack.
- **size**, this function should return the number of data items in the Stack.

2 Implementing a Binary Tree: 30 points

Your task is to create a class named **BinaryTree** satisfying the following requirements (you can create more instance variables if you want):

1) Each node inside the binary tree has at least these attributes: value(data stored in the node), leftChild (reference to another node), and rightChild (reference to another node)

2) Must have an **insert** function (i.e., insert a new node).

3) Must have a **levelOrderTraversal** function that performs level-order traversal of current tree, and print out the value at each node when visit that node. If you choose "queue" to implement level-order traversal, you need write your own queue class.

3 Reconstruct a Binary Tree From a File Using a Stack: 30 points

Create a **loadTreeFromFile** function in the above BinaryTree class. This function takes a String filename as input, and return a BinaryTree object. The file lists the nodes of the tree in post-order sequence, one node per line. Each line contains a string which is the data value stored in the node, and two other 0/1 tags. The first tag indicates whether or not this node has a left child, the second tag indicates whether the node has a right child. For example, to reconstruct the sample binary tree in Figure 1, the input file contains the following lines:
2 0 0
7 0 0
4 1 1
10 0 0
9 0 1
8 1 1

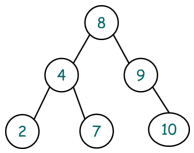

Figure 1: Binary Tree Sample

The above information is sufficient to uniquely determine any binary tree. The tree can be reconstructed from the file using the following algorithm:

```
create an empty stack of BinaryTree
while there is another line of input in the file:
    read a line (containing data and 2 tags)

    if the second tag is 1:
      pop an element from the stack and call it right_tree
```

```
if the first tag is 1:
  pop an element from the stack and call it left_tree

create a new binary tree with data as its root,
 and left_tree and right_tree as its subtrees (if they exist)

push the new tree onto your stack
```

When you exit the while loop, the stack should contains one element and that is the tree you want to return. Note that you should check the second tag (indicating if the node has right child or not) first. And if your right_tree and left_tree is empty, you should create a new tree with only one node, i.e., the root node. You should use the Stack you created in Section 1.

4 Test Code and Style: 30 points

You must comment your code, and write test code. In your test code, you should:

1) Create the tree in Figure 1 by inserting a set of nodes to an empty binary tree.

2) Perform level-order traversal by calling the levelOrderTraversal function inside your BinaryTree class.

3) Design the a string representating the following expression: ((((3 +1)×3)/((9-5)+2))-((3×(7-4))+6)) . Ref. slide 22 of lecture 11.

4) Save the above input in a file and restore a tree from the file using the loadTreeFromFile function.

5) Perform level-order traversal by calling the levelOrderTraversal function.

Assignment Requirements

This assignment is to be completed individually (we have the right to run any clone detection tool). IDE is highly suggested for this assignment. Otherwise, you should consider adding a print() function to check the current states of your tree. You need submit your documented source code (Stack class + BinaryTree class + test code). If you have multiple files, please combine all of them into a .zip archive and submit it to OnQ system. The .zip archive should contain your student number. If you have any question regarding this assignment, you can post a question on Piazza or send me an email, or come during office hours. The deadline will be specified on Onq.