Lab_trees Tempestuous Trees,

Week #6 – February 21-23, 2018

Welcome to Lab Trees!

Course Website: https://courses.engr.illinois.edu/cs225/labs

Overview

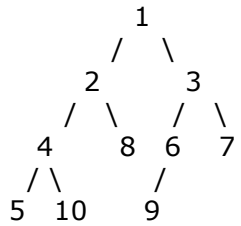
In this week's lab, you will work with binary trees, learn about tree traversals and implement them using iterators, thinking recursively while working with helper functions.

Tree Traversals

Tree traversals refer to the process of ***visiting each node in the tree data structure exactly once***. The order in which the nodes are visited determines the type of traversal:

- **Inorder:** left subtree -> current node -> right subtree (useful for checking if the tree is a Binary Search Tree)
- **Preorder:** current node -> left subtree -> right subtree (useful for copying the tree)
- **Postorder:** left subtree -> right subtree -> current node (useful for deleting the tree)

Exercise 1.1: What will an Inorder traversal of the following tree print out if we start al the root (node **1**)??



Inorder traversal: 5 4 10 2 8 1 9 6 3 7

Exercise 1.2: Write the printInOrder() function.

tree.h

```
1  class Tree{
2    public:
3        struct Node {
4              int value;
5              Node* left;
6              Node* right;
7              Node(int value = 0, Node left = NULL,
8              Node right = NULL):
9              value(value),left(left),right(right){}
10       };
11       Node* root;
12       void printInOrder(const Node* subRoot) const
13       {
14       //YOUR CODE HERE
15       if(subRoot == NULL) {
16             return;
17       }
18       printInOrder(subRoot->left);
19       cout << subRoot->value << " ";
20       printInOrder(subRoot->right);
21       }
22 };
23
24
```

Exercise 1.3: What is the worst case running time of the following operations, assuming you start at the root node?

Printing the value of the root node	**O(1)**
Printing the value of any leaf node	**O(n)**
Printing the value of each node in the tree	**O(n)**
Finding the node with the largest value	**O(n)**

Using a Helper Function

When using recursive functions, we would like the client code to directly call the functions on their object without needing to know/provide information about subproblems and parameters. This is why we use helper functions: they allow us to pass in parameters, separately handle the subproblems, and make our code much more user-friendly and readable.

Exercise 2: Now that we know the runtime of finding the largest value in a binary tree from the previous exercise, write a function that will find the largest node value. In the code below, findLargest() is a public member function of the Tree class introduced in **Exercise 1**.

tree.h (CONTINUED)

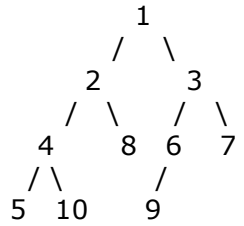
```
1  //CONTINUED FROM EXERCISE 1
2  int findLargest() {
3   //YOUR CODE HERE
4   return findLargest(root);
5  }
6
7  int findLargest(const Node* subRoot) const{
8     //YOUR CODE HERE
9     //SHOULD BE RECURSIVE FUNCTION!
10    int currentMax = subRoot->value;
11    if(subRoot->left!=NULL){
12      currentMax = std::max(currentMax,
13                   findLargest(subRoot->left));
14    }
15    if(subRoot->right!=NULL){
16      currentMax = std::max(currentMax,
17                   findLargest(subRoot->right));
18    }
19 }
20 };
```

Iterators

In the previous exercise, we saw how to traverse trees using recursion. Another way to do this is by using a temporary data structure such as a **stack** to keep track of where we are as we traverse the tree. You will use this method when implementing tree traversal iterator classes in the coding part of this lab.

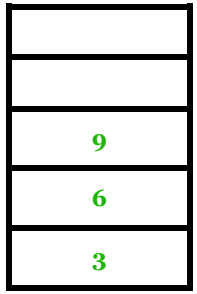
Exercise 3.1: Fill in the blanks in this pseudo code implementation of Inorder traversal using a stack as a placeholder for the traversal.

```
1) Create an empty stack S.

2) Initialize current node as root

3) Until current = NULL, push current node to S and set

   current = current->left

4) If current = NULL and S is not empty then

   a) Pop the top item from stack

   b) Print the popped item

   c) Set current = popped_item->right

   d) Go to step 3

5) If current = NULL and S is empty then we are done
```



Exercise 3.2: Suppose we run this pseudo code on the tree above, what does the stack **S** look like when **current** is pointing to node **9** ?

S =



In the programming part of this lab, you will:

- Get familiar with binary trees
- Practice writing recursive functions using helper functions
- Complete the implementation of an Inorder tree iterator

As your TA and CAs, we're here to help with your programming for the rest of this lab section!