CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

Objectives

1. Recursion
2. Tree Data Structure
3. Binary Trees
4. Pre-order traversal
5. In-order traversal
6. Post-order traversal
1. Exercise

1. Recursion

- Recursion – When a program calls another instance of itself.
- Recursive definition – A definition in which something is defined in terms of a smaller version of itself.
- Recursive algorithm – To solve a problem by reducing the problem into smaller versions of itself.
- Recursive function – The function that calls itself.
- Base case – the first/final call of the program that requires no recursion and the solution is computed directly.
- General case – Rest of the function calls where the solution is obtained indirectly using recursion.

Note:- For the recursion to terminate, every recursive function call must terminate at the base case.

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

Example: Factorial

A factorial of a number is the product of all the numbers starting from 1 to that number.

factorial(10) = 10*9*8*7*6*5*4*3*2*1
factorial(3) = 3*2*1

To solve this recursively, we create

- Recursive definition: factorial(n) = n*factorial(n-1)
- Base case: factorial(1) = 1
- General Case: factorial(n) = n * factorial(n-1)

Thus, the above examples can be recursively written as

factorial(10) = 10 * factorial(9)
factorial(3) = 3 * factorial(2)
= 3 * (2 * factorial(1))
= 3 * (2 * 1)
= 3 * 2
= 6

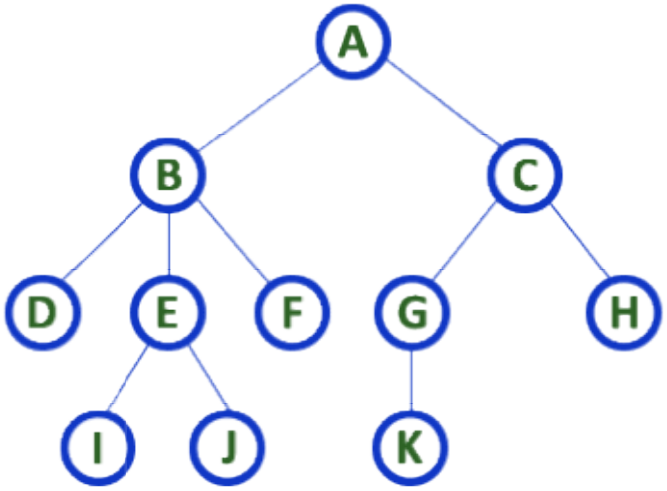
Where the functions are recursively called till n = 1. As n reduces by 1 at every function call, the problem is bound to reach the base case, after which it terminates.

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

2. Tree Data Structure



Technical Definition: A tree is a collection of entities called nodes. Nodes are connected by edges. Each node contains a value or data, and it may or may not have a child node.
Tree is an example of non-linear data structures. A structure is a way of representing the hierarchical nature of a structure in graphical form.

In simple terms, a tree is a data structure that is similar to a linked list but instead of each node pointing simply to the next node in a linear fashion, each node points to a number of nodes.

In trees ADT(Abstract data type), order of elements is not important. If we need ordering information linear data structures like linked lists, stacks, queues, etc can be used.

- The root of a tree is the node with no parents. There can be at most one root node in a tree.
- An edge refers to the link from parent to child.
- A node with no children is called leaf node.
- Children of the same parent are called siblings.
- A node p is an ancestor of node q if there exists a path from root to q and p appears on the path.
- Set of all nodes at a given depth is called level of the tree.The root node is at level 0.

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

2. Binary Trees

A tree is called a binary tree in which each node has at the most two children, which are referred to as the left child and the right child i.e each node has zero child, one child or two children. Empty tree is also a valid binary tree. We can visualize a binary tree as consisting of a root and two disjoint binary trees, called the left and right subtrees of the root.

Types of binary trees

1. Full Binary Tree - A full binary tree is a binary tree in which all nodes except leaves have two children.
2. Complete Binary Tree - A complete binary tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

3. Perfect Binary Tree - A perfect binary tree is a binary tree in which all internal nodes have two children and all leaves are at same level.

Example code to create a Binary Tree:

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

Applications of trees data structure

1. Expression trees are used in compilers.
2. Manipulate hierarchical data.
3. Used to implement search algorithms.

3. Binary Tree Traversals

In order to process trees, we need a mechanism for traversing them. The process of visiting all nodes of a tree is called tree traversal. Each node is processed only once but it may be visited more than once. As we have seen already that in linear data structures like linked lists, stacks

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

and queues the elements are visited in sequential order. But, in tree structures, there are many different ways.

Traversal possibilities

Starting at the root of a binary tree, there are 3 main steps that can be performed and the order in which they are performed defines the traversal type. These steps are: performing an action on the current node, traversing to the left child node, and traversing to the right child node. This process can be easily defined through recursion.

1. LDR: Process left subtree, process the current node data and then process the right subtree
2. LRD: Process left subtree, process right subtree, process current node data.
3. DLR: Process current node data, process left subtree, process right subtree.
4. DRL: Process current node data, process right subtree, process left subtree.
5. RDL: Process right subtree, process current node data,, process left subtree.
6. RLD: Process right subtree, process current node data, process left subtree.

Classifying the traversals

The sequence in which these nodes are processed defines a particular traversal method. The classification is based on the order in which current node is processed. That means, if we are classifying based on current node(D) and if D comes in the middle it does not matter whether L is on the left side of D or R is on the left side of D. Similarly, it doesn't matter whether L is on the right side of D or R is on the right side of D. Due to this, the total possibilities are reduced to 3 and these are:

1. Preorder Traversal (DLR)
2. Inorder Traversal (LDR)
3. PostOrder Traversal (LRD)

There is another traversal method which does not depend on above orders and it is :
Level Order Traversal. (We will cover this later.)

Note - The below traversal techniques can be done using both using recursion and iterative techniques. However for this class we will only focus on recursive techniques, since it is much simpler and easy to understand.

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

PreOrder Traversal

In preorder traversal, each node is processed before(pre) either of its subtrees. This is the simplest traversal to understand. However, even though each node is processed before the subtrees, it still requires that some information must be maintained while moving down the tree. Processing must return to the right subtree after finishing the processing of the left subtree. To move to the right subtree after processing left subtree, we must maintain the root information. The obvious ADT for such information is a stack. Because of its LIFO structure, it is possible to get the information about the right subtrees back in reverse order.

Pre order Traversal is defined as follows.

1. Visit the root.
2. Traverse the left subtree in Preorder.
3. Traverse the right subtree in Preorder.

```
void preorder(Node *root)
{
      if(root != NULL)
      {
            print(root->data);
            preorder(root->left);
            preorder(root->right);
      }
}
```

InOrder Traversal

In inorder traversal the root is visited between the subtrees, inorder traversal is defined as follows.

1. Traverse the left subtree in Inorder.
2. Visit the root.
3. Traverse the right subtree in Inorder.

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

```
void inorder(int *root)
{
      if(root != NULL)
      {
            inorder(root->left);
            print(root->data);
            inorder(root->right);
      }
}
```

PostOrder Traversal

In post order traversal, the root is visited after both subtrees. Postorder traversal is defined as follows.

1. Traverse the left subtree in PostOrder.
2. Traverse the right subtree in PostOrder.
3. Visit the root.

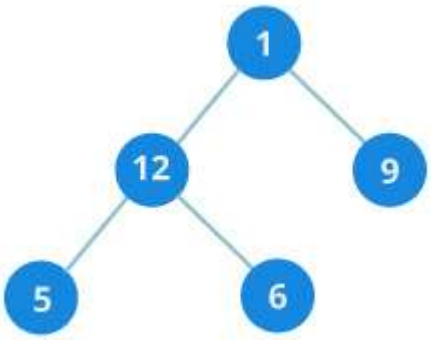
```
void postorder(int *root)
{
      if(root == NULL){
         return;
      }

      postorder(root->left);
      postorder(root->right);
      print(root->data);
}
```

CSCI 2270 – Data Structures

Recitation 6, Fall 2021

Recursion, Trees and Traversal techniques

Example tree with PreOrder, InOrder and PostOrder Traversals:



Inorder (Left, Root, Right) : 5, 12, 6, 1, 9
Preorder (Root, Left, Right) : 1, 12, 5, 6, 9
Postorder (Left, Right, Root) : 5, 6, 12, 9, 1

4. Complexity Analysis of Binary Tree Traversals

For all of these traversals - whether done recursively or iteratively we visit every node in the binary tree. That means that we'll get a runtime complexity of O(n) - where n is the number of nodes in the binary tree.

Exercise

Download the Lab6 zipped file from Canvas. It has the header and implementation files for the tree class. Follow the TODO details.

1. Implement sumNodes function which sums the data of all the nodes of the tree. (Silver Problem - Mandatory)
1. Implement printLeafNode function which prints just the leaf nodes of the tree. (Gold Problem)