06.11.2020-YOGSHWARAN -SOLUTION DESIGN

NUMBER OF ROOT TO LEAF PATHS:

The problem here is to print the number of paths from root to leaf nodes in a binary tree in a format such that the length of the path is increasing followed by the number of paths to that length.

- ✓ The basic operation is to traverse through the entire tree to find the leaf nodes
- ✓ The control enters the recursive function through the main function.
- ✓ The problem is broken into subproblems by identifying the tri-node.
- ✓ The value of the pathlength will be incremented in each step.
- ✓ The base-condition is that the node is not null if the node becomes null then 0 Is returned.

The problem is solved by using the overlapping sub-problems approach and the values are stored in a map to avoid redundancy and the values in the map are printed

CONSTRUCT TREE FROM LINKED LIST:

We are given a linked list and root node for a tree. Our task is to find a way to convert the linked list into a tree.

- ✓ The basic operation is to traverse through the entire LinkedList to find the nodes.
- ✓ The control enters the recursive function through the main function.
- ✓ The problem is broken into subproblems by identifying the tri-node.
- ✓ The LinkedList is traversed to next element in each step.
- ✓ The base-condition is that the LinkedList iterator is not null if the base condition is true then the root of the tree is returned.

The nodes of the tree are created in each step of the incremental phase of the LinkedList

COUNT NUMBER OF BST NODES IN A GIVEN RANGE:

We are given a Binary Search Tree and two extremes of a range. Our task is to find the elements that are in the BST that fall into the given range.

- ✓ The basic operation is to traverse through the entire BST and compare the values of each nodes with the given range.
- ✓ The control enters the recursive function through the main function.
- ✓ The problem is broken into subproblems by identifying the tri-node.
- ✓ The LinkedList is traversed to left or right or both by checking the value of the root->data.

- ✓ The base-condition is that the node is not null if the node becomes null then 0 Is returned.

Here the idea is to use the data constraint of the BST to our advantage. The function is optimised to visit only to the nodes that fall under the specified range. The function begins with root and checks if the root->data is in-between the low and high if yes then both the root->left and root->right are visited else if root->data > low then only root->left is visited and if root->data is < high then only root->right is visited.

<u>FOLDABLE BINARY TREE:</u>

The problem here is to check whether the given binary tree is foldable or not and it is also given that an empty tree is foldable.

- ✓ The basic operation is to traverse through the entire tree to find whether the left subtree is an exact mirror image of the right subtree of the root.
- ✓ The control enters the recursive function through the main function.
- ✓ The problem is broken into subproblems by identifying the tri-node.
- ✓ The left and right tree structures are checked.
- ✓ The base-condition is that the node is not null if the node becomes null then true is returned.

The arguments are passed in such a way like if left->right is passed then right->left is passed to check the structure.

<u>SUBTREES WITH GIVEN SUM:</u>

The problem here is to print the number of subtrees that has the given sum.

- ✓ The basic operation is to traverse through the entire tree to find the sum of the subtrees.
- ✓ The control enters the recursive function through the main function.
- ✓ The problem is broken into subproblems by identifying the tri-node.
- ✓ The value of the root->data is added to the sum in each step.
- ✓ The base-condition is that the node is not null if the node becomes null then 0 Is returned.

The problem is solved by using the overlapping sub-problems approach and if the value of the subtree sum is equal to the given sum then the count variable is incremented.