Print Left View of a Binary Tree

Given a Binary Tree, print left view of it. Left view of a Binary Tree is set of nodes visible when tree is visited from left side. Left view of following tree is 12, 10, 25.

```
          12
       /     \
     10       30
            /    \
          25      40
```

The left view contains all nodes that are first nodes in their levels. A simple solution is to **do level order traversal** and print the first node in every level.

The problem can also be solved **using simple recursive traversal**. We can keep track of level of a node by passing a parameter to all recursive calls. The idea is to keep track of maximum level also. Whenever we see a node whose level is more than maximum level so far, we print the node because this is the first node in its level (Note that we traverse the left subtree before right subtree). Following is the implementation-

C

```
// C program to print left view of Binary Tree
#include<stdio.h>
#include<stdlib.h>

struct node
{
    int data;
    struct node *left, *right;
};

// A utility function to create a new Binary Tree node
struct node *newNode(int item)
{
    struct node *temp =  (struct node *)malloc(sizeof(struct node));
    temp->data = item;
    temp->left = temp->right = NULL;
    return temp;
}

// Recursive function to print left view of a binary tree.
void leftViewUtil(struct node *root, int level, int *max_level)
{
    // Base Case
    if (root==NULL)  return;

    // If this is the first node of its level
    if (*max_level < level)
    {
        printf("%d\t", root->data);
        *max_level = level;
    }

    // Recur for left and right subtrees
    leftViewUtil(root->left, level+1, max_level);
    leftViewUtil(root->right, level+1, max_level);
}

// A wrapper over leftViewUtil()
void leftView(struct node *root)
{
    int max_level = 0;
    leftViewUtil(root, 1, &max_level);
}

// Driver Program to test above functions
int main()
{
    struct node *root = newNode(12);
    root->left = newNode(10);
    root->right = newNode(30);
    root->right->left = newNode(25);
    root->right->right = newNode(40);

    leftView(root);

    return 0;
}
```

Java

```
// Java program to print left view of binary tree

/* Class containing left and right child of current
node and key value*/
class Node
```

```
{
    int data;
    Node left, right;

    public Node(int item)
    {
        data = item;
        left = right = null;
    }
}

/* Class to print the left view */
class BinaryTree
{
    Node root;
    static int  max_level = 0;

    // recursive function to print left view
    void leftViewUtil(Node node, int level)
    {
        // Base Case
        if (node==null) return;

        // If this is the first node of its level
        if (max_level < level)
        {
            System.out.print(" " + node.data);
            max_level = level;
        }

        // Recur for left and right subtrees
        leftViewUtil(node.left, level+1);
        leftViewUtil(node.right, level+1);
    }

    // A wrapper over leftViewUtil()
    void leftView()
    {
        leftViewUtil(root, 1);
    }

    /* testing for example nodes */
    public static void main(String args[])
    {
        /* creating a binary tree and entering the nodes */
        BinaryTree tree = new BinaryTree();
        tree.root = new Node(12);
        tree.root.left = new Node(10);
        tree.root.right = new Node(30);
        tree.root.right.left = new Node(25);
        tree.root.right.right = new Node(40);

        tree.leftView();
    }
}
```

Python

```
# Python program to print left view of Binary Tree

# A binary tree node
class Node:

    # Constructor to create a new node
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None


# Recursive function pritn left view of a binary tree
def leftViewUtil(root, level, max_level):

    # Base Case
    if root is None:
        return

    # If this is the first node of its level
    if (max_level[0] < level):
        print "%d\t" %(root.data),
        max_level[0] = level

    # Recur for left and right subtree
    leftViewUtil(root.left, level+1, max_level)
    leftViewUtil(root.right, level+1, max_level)


# A wrapper over leftViewUtil()
def leftView(root):
    max_level = [0]
    leftViewUtil(root, 1, max_level)


# Driver program to test above function
root = Node(12)
root.left = Node(10)
root.right = Node(20)
root.right.left = Node(25)
root.right.right = Node(40)

leftView(root)

# This code is contributed by Nikhil Kumar Singh(nickzuck_007)
```

Output:

```
12      10      25
```

Time Complexity: The function does a simple traversal of the tree, so the complexity is O(n).

This article is contributed by Ramsai Chinthamani. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

61 Comments Category: Trees

Related Posts:

- Check if removing an edge can divide a Binary Tree in two halves
- Check sum of Covered and Uncovered nodes of Binary Tree
- Lowest Common Ancestor in a Binary Tree | Set 2 (Using Parent Pointer)
- Construct a Binary Search Tree from given postorder
- BFS vs DFS for Binary Tree
- Maximum difference between node and its ancestor in Binary Tree
- Inorder Non-threaded Binary Tree Traversal without Recursion or Stack

- Check if leaf traversal of two Binary Trees is same?

Like Share 25 people like this.

Writing code in comment? Please use code.geeksforgeeks.org, generate link and share the link here.