Editorial

X

Recursive Approach

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).

C++ Code for Recursive Approach

```
void leftViewHelper(node *root, int level, int &max_level) {
    if(root == NULL) {
        return;
    }
    if(max_level<level) {
        cout<<root->data<<" ";
        max_level = level;
    }
    leftViewHelper(root->left, level+1, max_level);
    leftViewHelper(root->right, level+1, max_level);
}

void leftView(node *root) {
    int max_level = 0;
    leftViewHelper(root, 1, max_level);
}
```

Java Code for Recursive Approach

```
 public  void LeftView() {

        int[] maxLevel = new int[1];
        maxLevel[0] = -1;
        dfs(root, 0, maxLevel);

    }

    public  void dfs(Node root, int level, int[] maxLevel) {

        if (root == null) {
            return;
        }


        if(level > maxLevel[0]){
            System.out.print(root.data+" ");
            maxLevel[0] = level;
        }

        dfs(root.left, level + 1, maxLevel);
        dfs(root.right, level + 1, maxLevel);
    }
```

Iterative Approach (Better)

Do a level order traversal.
Push NULL into the queue to mark the end of each level.
Print the data of root of data and start with level order traversal.
As you encounter a NULL, print the next element.
Otherwise skip all the other elements

C++ Code for Iterative Approach

```
void printLeftSide(node *root)
{
    queue<node *> q;
    q.push(root);
    q.push(NULL);

    bool flag = true;

    while (!q.empty())
    {
        node *f = q.front();
        if (f == NULL)
        {
            q.pop();
            if (!q.empty())
            {
                q.push(NULL);
            }
            flag = true;
        }
        else
        {
            if (flag)
            {
                cout << f->data << " ";
                flag = false;
            }
            q.pop();

            if (f->left)
            {
                q.push(f->left);
            }
            if (f->right)
            {
                q.push(f->right);
            }
        }
    }
    return;
}
```

Java Code for Iterative Approach

```
static void printLeftSide(TreeNode root) {
    Queue<TreeNode> q = new LinkedList<>();
    q.add(root);
    q.add(null);

    boolean flag = true;

    while (!q.isEmpty()) {
        TreeNode f = q.peek();
        if (f == null) {
            q.remove();
            if (!q.isEmpty()) {
                q.add(null);
            }
            flag = true;
        } else {
            if (flag) {
                System.out.print(f.data + " ");
                flag = false;
            }
            q.remove();

            if (f.left != null) {
                q.add(f.left);
            }
            if (f.right != null) {
                q.add(f.right);
            }
        }
    }
}
```