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. And traverse the tree in a manner that right subtree is visited before left subtree. Whenever we see a node whose level is more than maximum level so far, we print the node because this is the last node in its level (Note that we traverse the right subtree before left subtree) .

C++ Code for Recursive Approach

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

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

Java Code for Recursive Approach

```
public  void RightView() {

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

Iterative Approach

C++ Code for Iterative Approach

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

    int prev = 0;

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

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

Java Code for Iterative Approach

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

    int prev = 0;

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

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