

<https://course.acciojob.com/idle?question=09fc4541-95da-4be3-8c19-875d32fe59c1>

Binary Tree Maximum Path Sum

In a binary tree, a path is a sequence of nodes where each node is connected to the next by an edge. The path sum is the total of the values of the nodes in that path. The problem is to find the non-empty path which has the highest path sum among all the paths in the tree, given the root of the tree. The path doesn't need to go through the root node.

Input Format

Single line contains separated elements of tree in level order

Output Format

Print the maximum path sum.

Example 1

Input

```
1 2 3 4
```

Output

```
10
```

Explanation

```
1
```

```
/ \
```

2 3

/

4

Maximum path is 10 from 4 through 1 to 3.

Example 2

Input

1 2 3 4 N 5 6

Output

16

Explanation

1

/ \

2 3

/ / \

4 5 6

Maximum path sum is 4->2->1->3->6.

Constraints

2 <= N <=10000

Topic Tags

- Trees

My code

```
import java.util.*;

class TreeNode {
    int data;
    TreeNode left, right;
    public TreeNode(int data) {
        this.data = data;
        this.left = null;
        this.right = null;
    }
}

class Solution{

    static int max=0;
    static int maxPath(TreeNode root)
    {
        if(root==null)
            return 0;
        int l= maxPath(root.left);
        int r= maxPath(root.right);
        int t=l+r+root.data;
        if(t>max)
            max=t;
        if(l>r)
            return ((l+root.data)>0)?l+root.data:0;
        return ((r+root.data)>0)?r+root.data:0;
    }
    public static int maxPathSum(TreeNode root) {
```

```

//Write your code
        int t=maxPath(root);
        return max;
    }
}

```

```

class Main {
    public static TreeNode buildTree(String str) {
        if (str.length() == 0 || str.charAt(0) == 'N')
            return null;

        String[] nodes = str.split(" ");
        Queue<TreeNode> queue = new LinkedList<>();
        TreeNode root = new TreeNode(Integer.parseInt(nodes[0]));
        queue.add(root);

        int i = 1;
        while (!queue.isEmpty() && i < nodes.length) {

            TreeNode currNode = queue.poll();

            String currVal = nodes[i];

            if (!currVal.equals("N")) {
                currNode.left = new TreeNode(Integer.parseInt(currVal));
                queue.add(currNode.left);
            }
            i++;
            if (i >= nodes.length)
                break;
            currVal = nodes[i];

            if (!currVal.equals("N")) {
                currNode.right = new TreeNode(Integer.parseInt(currVal));
                queue.add(currNode.right);
            }
            i++;
        }

        return root;
    }
}

```

```
public static void main(String[] args) {  
    Scanner sc = new Scanner(System.in);  
    String s = sc.nextLine();  
    TreeNode root = buildTree(s);  
    int maxPath = Solution.maxPathSum(root);  
    System.out.println(maxPath);  
}  
}
```