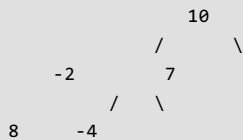


Find the maximum sum leaf to root path in a Binary Tree

Given a Binary Tree, find the maximum sum path from a leaf to root. For example, in the following tree, there are three leaf to root paths 8->-2->10, -4->-2->10 and 7->10. The sums of these three paths are 16, 4 and 17 respectively. The maximum of them is 17 and the path for maximum is 7->10.



Solution

- 1) First find the leaf node that is on the maximum sum path. In the following code `getTargetLeaf()` does this by assigning the result to `*target_leaf_ref`.
- 2) Once we have the target leaf node, we can print the maximum sum path by traversing the tree. In the following code, `printPath()` does this.

The main function is `maxSumPath()` that uses above two functions to get the complete solution.

C

```
#include<stdio.h>
#include<limits.h>

/* A tree node structure */
struct node
{
    int data;
    struct node *left;
    struct node *right;
};

// A utility function that prints all nodes
// on the path from root to target_leaf
bool printPath (struct node *root,
                struct node *target_leaf)
{
    // base case
    if (root == NULL)
        return false;

    // return true if this node is the target_leaf
    // or target leaf is present in one of its
    // descendants
    if (root == target_leaf ||
        printPath(root->left, target_leaf) ||
        printPath(root->right, target_leaf) )
    {
        printf("%d ", root->data);
        return true;
    }

    return false;
}

// This function Sets the target_leaf_ref to refer
// the leaf node of the maximum path sum. Also,
// returns the max_sum using max_sum_ref
void getTargetLeaf (struct node *node, int *max_sum_ref,
                   int curr_sum, struct node **target_leaf_ref)
{
    if (node == NULL)
        return;

    // If the current node is a leaf node, check if it is the
    // maximum sum leaf node
    if (node->left == NULL && node->right == NULL)
    {
        if (*max_sum_ref < curr_sum)
        {
            *max_sum_ref = curr_sum;
            *target_leaf_ref = node;
        }
    }

    // Traverse the left and right subtrees
    curr_sum += node->data;
    getTargetLeaf(node->left, max_sum_ref, curr_sum, target_leaf_ref);
    getTargetLeaf(node->right, max_sum_ref, curr_sum, target_leaf_ref);
    curr_sum -= node->data;
}
```

```

    if (node == NULL)
        return;

    // Update current sum to hold sum of nodes on path
    // from root to this node
    curr_sum = curr_sum + node->data;

    // If this is a leaf node and path to this node has
    // maximum sum so far, then make this node target_leaf
    if (node->left == NULL && node->right == NULL)
    {
        if (curr_sum > *max_sum_ref)
        {
            *max_sum_ref = curr_sum;
            *target_leaf_ref = node;
        }
    }

    // If this is not a leaf node, then recur down
    // to find the target_leaf
    getTargetLeaf (node->left, max_sum_ref, curr_sum,
                  target_leaf_ref);
    getTargetLeaf (node->right, max_sum_ref, curr_sum,
                  target_leaf_ref);
}

// Returns the maximum sum and prints the nodes on max
// sum path
int maxSumPath (struct node *node)
{
    // base case
    if (node == NULL)
        return 0;

    struct node *target_leaf;
    int max_sum = INT_MIN;

    // find the target leaf and maximum sum
    getTargetLeaf (node, &max_sum, 0, &target_leaf);

    // print the path from root to the target leaf
    printPath (node, target_leaf);

    return max_sum; // return maximum sum
}

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

/* Driver function to test above functions */
int main()
{
    struct node *root = NULL;

    /* Constructing tree given in the above figure */
    root = newNode(10);
    root->left = newNode(-2);
    root->right = newNode(7);
    root->left->left = newNode(8);
    root->left->right = newNode(-4);

    printf ("Following are the nodes on the maximum "
           "sum path \n");
    int sum = maxSumPath(root);
    printf ("\nSum of the nodes is %d ", sum);
}

```

```

    getchar();
    return 0;
}

```

Java

```

// Java program to find maximum sum leaf to root
// path in Binary Tree

// A Binary Tree node
class Node
{
    int data;
    Node left, right;

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

// A wrapper class is used so that max_no
// can be updated among function calls.
class Maximum
{
    int max_no = Integer.MIN_VALUE;
}

class BinaryTree
{
    Node root;
    Maximum max = new Maximum();
    Node target_leaf = null;

    // A utility function that prints all nodes on the
    // path from root to target_leaf
    boolean printPath(Node node, Node target_leaf)
    {
        // base case
        if (node == null)
            return false;

        // return true if this node is the target_leaf or
        // target leaf is present in one of its descendants
        if (node == target_leaf || printPath(node.left, target_leaf)
            || printPath(node.right, target_leaf))
        {
            System.out.print(node.data + " ");
            return true;
        }

        return false;
    }

    // This function Sets the target_leaf_ref to refer
    // the leaf node of the maximum path sum. Also,
    // returns the max_sum using max_sum_ref
    void getTargetLeaf(Node node, Maximum max_sum_ref,
        int curr_sum)
    {
        if (node == null)
            return;

        // Update current sum to hold sum of nodes on
        // path from root to this node
        curr_sum = curr_sum + node.data;
        int a = max_sum_ref.max_no;

        // If this is a leaf node and path to this node

```

```

// If this is a leaf node and path to this node
// has maximum sum so far, then make this node
// target_leaf
if (node.left == null && node.right == null)
{
    if (curr_sum > max_sum_ref.max_no)
    {
        max_sum_ref.max_no = curr_sum;
        target_leaf = node;
    }
}

// If this is not a leaf node, then recur down
// to find the target_leaf
getTargetLeaf(node.left, max_sum_ref, curr_sum);
getTargetLeaf(node.right, max_sum_ref, curr_sum);
}

// Returns the maximum sum and prints the nodes on
// max sum path
int maxSumPath()
{
    // base case
    if (root == null)
        return 0;

    // find the target leaf and maximum sum
    getTargetLeaf(root, max, 0);

    // print the path from root to the target leaf
    printPath(root, target_leaf);
    return max.max_no; // return maximum sum
}

// driver function to test the above functions
public static void main(String args[])
{
    BinaryTreeNode tree = new BinaryTreeNode();
    tree.root = new Node(10);
    tree.root.left = new Node(-2);
    tree.root.right = new Node(7);
    tree.root.left.left = new Node(8);
    tree.root.left.right = new Node(-4);
    System.out.println("Following are the nodes "+
        "on maximum sum path");
    int sum = tree.maxSumPath();
    System.out.println("");
    System.out.println("Sum of nodes is : " + sum);
}
// This code has been contributed by Mayank Jaiswal

```

Output:

```

Following are the nodes on the maximum sum path
7 10
Sum of the nodes is 17

```

Time Complexity: Time complexity of the above solution is $O(n)$ as it involves tree traversal two times.