<http://buttercola.blogspot.com/2019/04/lintcode-596-minimum-subtree.html>

Given a binary tree, find the subtree with minimum sum. Return the root of the subtree.

### Example

Example 1:

Input:

{1,-5,2,1,2,-4,-5}

Output:1

Explanation:

The tree is look like this:

1

/ \

-5 2

/ \ / \

0 2 -4 -5

The sum of whole tree is minimum, so return the root.

Example 2:

Input:

{1}

Output:1

Explanation:

The tree is look like this:

1

There is one and only one subtree in the tree. So we return 1.

### Notice

LintCode will print the subtree which root is your return node. It's guaranteed that there is only one subtree with minimum sum and the given binary tree is not an empty tree.

**Attempt 1: 2022-12-31**

**Wrong Solution: Duplicate sum calculation happening for each node**

**Since getSum() and getSumHelper() both calculate sum (based on current node as root) with same formula:**

**In getSum(): sum = root.val + getSumHelper(root.left) + getSumHelper(root.right)**

**In getSumHelper(): result = root.val + getSumHelper(root.left) + getSumHelper(root.right)**

**Since each node will be visited in getSum() already once, the second visit happen during sum calculation as 'root' node in getSumHelper() is a duplicate one**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

s.test(two);

}

int sum = 0;

private void test(TreeNode root) {

Map<TreeNode, Integer> map = new HashMap<>();

getSum(root, map);

System.out.println("done");

}

private void getSum(TreeNode root, Map<TreeNode, Integer> map) {

if(root == null) {

return;

}

sum = root.val + helper(root.left) + helper(root.right);

map.put(root, sum);

getSum(root.left, map);

getSum(root.right, map);

}

private int helper(TreeNode root) {

if(root == null) {

return 0;

}

int result = 0;

result += root.val;

result += helper(root.left);

result += helper(root.right);

return result;

}

}

**Solution 1:  Divide and Conquer (30 min)**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

//one.left = six;

TreeNode result = s.findSubtree(two);

System.out.println(result);

}

int minSum = Integer.MAX\_VALUE;

TreeNode result = null;

public TreeNode findSubtree(TreeNode root) {

if(root == null) {

return null;

}

helper(root);

return result;

}

private int helper(TreeNode root) {

if(root == null) {

return 0;

}

// Divide

int left = helper(root.left);

int right = helper(root.right);

// Process & Conquer

int curSum = root.val + left + right;

if(minSum > curSum) {

minSum = curSum;

result = root;

}

return curSum;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<http://buttercola.blogspot.com/2019/04/lintcode-596-minimum-subtree.html>

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the root of binary tree

\* @return: the root of the minimum subtree

\*/

private TreeNode minNode = null;

private int minSum = Integer.MAX\_VALUE;

public TreeNode findSubtree(TreeNode root) {

// write your code here

if (root == null) {

return null;

}

findSubtreeHelper(root);

return minNode;

}

private int findSubtreeHelper(TreeNode root) {

if (root == null) {

return 0;

}

int left = findSubtreeHelper(root.left);

int right = findSubtreeHelper(root.right);

int ret = root.val + left + right;

if (ret < minSum) {

minSum = ret;

minNode = root;

}

return ret;

}

}

**Solution 2:  Pure Divide and Conquer with helper class Node return sum and TreeNode at the same time (30 min, the similar way as L865.Smallest Subtree with all the Deepest Nodes)**

**Style 1: Still with global variable 'result' to record the global minimum sum during traversal**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

TreeNode result = s.findSubtree(two);

System.out.println(result);

}

class Node {

TreeNode node;

int sum;

public Node(TreeNode node, int sum) {

this.node = node;

this.sum = sum;

}

}

private Node result = null;

public TreeNode findSubtree(TreeNode root) {

if(root == null) {

return null;

}

helper(root);

// Note: Don't write as helper(root).node, since global variable is

// the actual variable to record the final minimum sum subtree root,

// helper(root) return a helper class object 'Node' which keep updating

// during traversal, not the recorded 'Node' contains minimum sum

return result.node;

}

private Node helper(TreeNode root) {

if(root == null) {

return new Node(null, 0);

}

// Divide

Node left = helper(root.left);

Node right = helper(root.right);

// Process & Conquer

int curSum = root.val + left.sum + right.sum;

Node curResult = new Node(root, curSum);

if(result == null || result.sum > curResult.sum) {

result = curResult;

}

return curResult;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://yeqiuquan.blogspot.com/2017/03/lintcode-596-minimum-subtree_8.html>

**思路**

这一类的题目都可以这样做：

开一个ResultType的变量result，来储存拥有最小sum的那个node的信息。

然后用分治法来遍历整棵树。

一个小弟找左子数的sum，一个小弟找右子树的sum。

同时，我们根据算出来的当前树的sum决定要不要更新result。

当遍历完整棵树的时候，result里记录的就是拥有最小sum的子树的信息。

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root the root of binary tree

\* @return the root of the minimum subtree

\*/

class ResultType {

TreeNode node;

int sum;

public ResultType(TreeNode node, int sum) {

this.node = node;

this.sum = sum;

}

}

private ResultType result = null;

public TreeNode findSubtree(TreeNode root) {

// Write your code here

if (root == null) {

return null;

}

ResultType rootResult = helper(root);

return result.node;

}

public ResultType helper(TreeNode root) {

if (root == null) {

return new ResultType(null, 0);

}

ResultType leftResult = helper(root.left);

ResultType rightResult = helper(root.right);

ResultType currResult = new ResultType(root,

leftResult.sum + rightResult.sum + root.val);

if (result == null || currResult.sum < result.sum) {

result = currResult;

}

return currResult;

}

}

**Style 2: Without global variable 'result' but only return to record the global minimum sum during traversal**

public class TreeSolution {

private class TreeNode {

public int val;

public TreeNode left, right;

public TreeNode(int val) {

this.val = val;

this.left = this.right = null;

}

}

public static void main(String[] args) {

/\*\*

\* 1

\* / \

\* 2 5

\* / \ \

\* 3 4 6

\* /

\* 7

\* \

\* 8

\*/

TreeSolution s = new TreeSolution();

TreeNode one = s.new TreeNode(1);

TreeNode two = s.new TreeNode(2);

TreeNode three = s.new TreeNode(3);

TreeNode four = s.new TreeNode(4);

TreeNode five = s.new TreeNode(5);

TreeNode six = s.new TreeNode(6);

TreeNode seven = s.new TreeNode(7);

TreeNode eight = s.new TreeNode(8);

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

TreeNode result = s.findSubtree(two);

System.out.println(result);

}

class Node {

TreeNode node;

int sum;

int minSum;

public Node(TreeNode node, int sum, int minSum) {

this.node = node;

this.sum = sum;

this.minSum = minSum;

}

}

//private Node result = null;

public TreeNode findSubtree(TreeNode root) {

if(root == null) {

return null;

}

return helper(root).node;

//return result.node;

}

private Node helper(TreeNode root) {

if(root == null) {

// For none existing node, assign minimum sum as Integer.MAX\_VALUE

// it will help to maintain correct minimum sum for leaf node

// e.g

// leaf node 1(sum=1,minSum=1)

// / \

// null(sum=0,minSum=MAX) null(sum=0,minSum=MAX)

// During logic check if below two formula, the actual minSum=1 will be kept

// (1) if(left.minSum < curResult.minSum) -> false, skip

// (2) if(right.minSum < curResult.minSum) -> false, skip

return new Node(null,0, Integer.MAX\_VALUE);

}

// Divide

Node left = helper(root.left);

Node right = helper(root.right);

// Process & Conquer

int curSum = root.val + left.sum + right.sum;

Node curResult = new Node(root, curSum, curSum);

if(left.minSum < curResult.minSum) {

curResult.minSum = left.minSum;

curResult.node = left.node;

}

if(right.minSum < curResult.minSum) {

curResult.minSum = right.minSum;

curResult.node = right.node;

}

return curResult;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<http://buttercola.blogspot.com/2019/04/lintcode-596-minimum-subtree.html>

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the root of binary tree

\* @return: the root of the minimum subtree

\*/

public TreeNode findSubtree(TreeNode root) {

// write your code here

if (root == null) {

return null;

}

ResultType ans = findSubtreeHelper(root);

return ans.minNode;

}

private ResultType findSubtreeHelper(TreeNode root) {

if (root == null) {

return new ResultType(0, Integer.MAX\_VALUE, null);

}

ResultType left = findSubtreeHelper(root.left);

ResultType right = findSubtreeHelper(root.right);

ResultType ans = new ResultType(root.val + left.sum + right.sum,

root.val + left.sum + right.sum,

root);

if (left.minSum < ans.minSum) {

ans.minSum = left.minSum;

ans.minNode = left.minNode;

}

if (right.minSum < ans.minSum) {

ans.minSum = right.minSum;

ans.minNode = right.minNode;

}

return ans;

}

}

class ResultType {

int sum;

int minSum;

TreeNode minNode;

public ResultType(int sum, int minSum, TreeNode minNode) {

this.sum = sum;

this.minSum = minSum;

this.minNode = minNode;

}

}

**Refer to**

<https://www.jiuzhang.com/problem/minimum-subtree/>

/\*\*

\* Definition of TreeNode:

\* public class TreeNode {

\* public int val;

\* public TreeNode left, right;

\* public TreeNode(int val) {

\* this.val = val;

\* this.left = this.right = null;

\* }

\* }

\*/

public class Solution {

/\*\*

\* @param root: the root of binary tree

\* @return: the root of the minimum subtree

\*/

public TreeNode findSubtree(TreeNode root) {

ResultType result = helper(root);

return result.minSubtree;

}

ResultType helper(TreeNode root) {

if (root == null) {

return new ResultType(null, 0, Integer.MAX\_VALUE);

}

// 获得左右子树的和

ResultType leftResult = helper(root.left);

ResultType rightResult = helper(root.right);

int minSum = Integer.MAX\_VALUE;

// 计算当前子树和，并更新答案

int sum = root.val + leftResult.sum + rightResult.sum;

minSum = Math.min(sum, leftResult.minSum);

minSum = Math.min(minSum, rightResult.minSum);

ResultType result = new ResultType(root, sum, sum);

if (leftResult.minSum < result.minSum) {

result.minSubtree = leftResult.minSubtree;

result.minSum = leftResult.minSum;

}

if (rightResult.minSum < result.minSum) {

result.minSubtree = rightResult.minSubtree;

result.minSum = rightResult.minSum;

}

return result;

}

}

class ResultType {

public int sum, minSum;

public TreeNode minSubtree;

public ResultType(TreeNode minSubtree, int sum, int minSum) {

this.minSubtree = minSubtree;

this.minSum = minSum;

this.sum = sum;

}

}