<http://buttercola.blogspot.com/2019/04/lintcode-597-subtree-with-maximum.html>

Given a binary tree, find the subtree with maximum average. Return the root of the subtree.

### Example

**Example 1**

Input：

{1,-5,11,1,2,4,-2}

Output：11

Explanation:

The tree is look like this:

1

/ \

-5 11

/ \ / \

1 2 4 -2

The average of subtree of 11 is 4.3333, is the maximun.

**Example 2**

Input：

{1,-5,11}

Output：11

Explanation:

1

/ \

-5 11

The average of subtree of 1,-5,11 is 2.333,-5,11. So the subtree of 11 is the maximun.

### Notice

LintCode will print the subtree which root is your return node. It's guaranteed that there is only one subtree with maximum average.

**Attempt 1: 2023-01-01**

**Not like [Lint596.Minimum Subtree] Divide and Conquer without helper class is hard to implement since average value requires sync up sum and number of nodes at the same time, the most intuitive way is create helper class to return both simultaneously**

**Solution 1:  Pure Divide and Conquer with helper class Node return sum and TreeNode at the same time (10 min, the similar way as L865.Smallest Subtree with all the Deepest Nodes)**

**Style 1: Still with global variable 'result' to record the global maximum average 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) {

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.findSubtree2(two);

System.out.println(result);

}

class Node {

TreeNode node;

int sum;

int size;

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

this.node = node;

this.sum = sum;

this.size = size;

}

}

private Node result = null;

public TreeNode findSubtree2(TreeNode root) {

if(root == null) {

return null;

}

helper(root);

return result.node;

}

private Node helper(TreeNode root) {

if(root == null) {

return new Node(null,0, 0);

}

// Divide

Node left = helper(root.left);

Node right = helper(root.right);

// Process & Conquer

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

int curSize = 1 + left.size + right.size;

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

// 将除法变成乘法，去掉很多麻烦，还提高效率

if(result == null || result.sum \* curResult.size < curResult.sum \* result.size) {

result = curResult;

}

return curResult;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://yeqiuquan.blogspot.com/2017/03/lintcode-597-subtree-with-maximum.html>

**思路**

这一类的题目都可以这样做：

开一个ResultType的变量result，来储存拥有最大average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average，一个小弟找右子树的average。然后通过这两个来计算当前树的average。同时，我们根据算出来的当前树的average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大average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 maximum average of subtree

\*/

class ResultType {

TreeNode node;

int sum;

int size;

public ResultType(TreeNode node, int sum, int size) {

this.node = node;

this.sum = sum;

this.size = size;

}

}

private ResultType result = null;

public TreeNode findSubtree2(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, 0);

}

ResultType leftResult = helper(root.left);

ResultType rightResult = helper(root.right);

ResultType currResult = new ResultType(

root,

leftResult.sum + rightResult.sum + root.val,

leftResult.size + rightResult.size + 1);

// 将除法变成乘法，去掉很多麻烦，还提高效率

if (result == null

|| currResult.sum \* result.size > result.sum \* currResult.size) {

result = currResult;

}

return currResult;

}

}

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

import java.util.\*;

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);

// one.left = two;

// one.right = five;

// two.left = three;

// two.right = four;

// five.right = six;

// six.left = seven;

// seven.right = eight;

/\*\*

\* 2

\* / \

\* 1 3

\* / \

\* 4 5

\*/

two.left = one;

two.right = three;

three.left = four;

three.right = five;

//one.left = six;

TreeNode result = s.findSubtree2(two);

System.out.println(result);

}

class Node {

TreeNode node;

int sum;

int size;

double maxAvg;

public Node(TreeNode node, int sum, int size, double maxAvg) {

this.node = node;

this.sum = sum;

this.size = size;

this.maxAvg = maxAvg;

}

}

//private Node result = null;

public TreeNode findSubtree2(TreeNode root) {

if(root == null) {

return null;

}

return helper(root).node;

//return result.node;

}

private Node helper(TreeNode root) {

if(root == null) {

return new Node(null,0, 0, Double.MIN\_VALUE);

}

// Divide

Node left = helper(root.left);

Node right = helper(root.right);

// Process & Conquer

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

int curSize = 1 + left.size + right.size;

double avg = (double) curSum / curSize;

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

if(left.maxAvg > curResult.maxAvg) {

curResult.sum = left.sum;

curResult.size = left.size;

curResult.maxAvg = left.maxAvg;

curResult.node = left.node;

}

if(right.maxAvg > curResult.maxAvg) {

curResult.sum = right.sum;

curResult.size = right.size;

curResult.maxAvg = right.maxAvg;

curResult.node = right.node;

}

return curResult;

}

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<http://buttercola.blogspot.com/2019/04/lintcode-597-subtree-with-maximum.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 maximum average of subtree

\*/

public TreeNode findSubtree2(TreeNode root) {

// write your code here

if (root == null) {

return null;

}

ResultType ans = findSubtree2Helper(root);

return ans.node;

}

private ResultType findSubtree2Helper(TreeNode root) {

if (root == null) {

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

}

ResultType left = findSubtree2Helper(root.left);

ResultType right = findSubtree2Helper(root.right);

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

int numNodes = left.numNodes + right.numNodes + 1;

double ave = (double) sum / numNodes;

TreeNode node = null;

if (ave > left.maxAve && ave > right.maxAve) {

node = root;

} else if (left.maxAve > ave && left.maxAve > right.maxAve) {

ave = left.maxAve;

node = left.node;

} else {

ave = right.maxAve;

node = right.node;

}

return new ResultType(sum, numNodes, ave, node);

}

}

class ResultType {

int sum;

int numNodes;

double maxAve;

TreeNode node;

public ResultType(int sum, int numNodes, double maxAve, TreeNode node) {

this.sum = sum;

this.numNodes = numNodes;

this.maxAve = maxAve;

this.node = node;

}

}