Dear reader, welcome to the article on the problem named **‘**[**Node With Maximum Subtree Sum**](https://www.pepcoding.com/resources/online-java-foundation/generic-tree/node-with-maximum-subtree-sum-official/ojquestion)**’.**

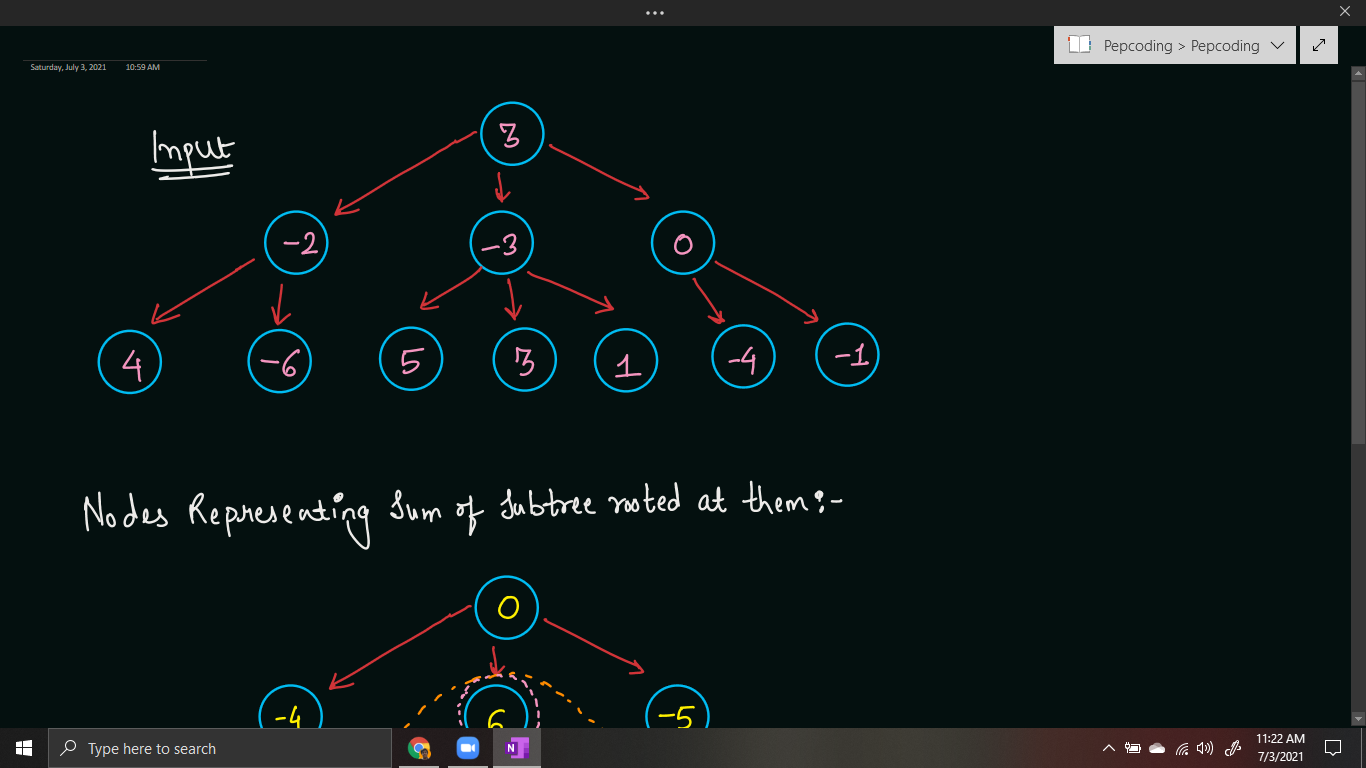
This problem is a very important problem, as you will get to learn about a *new strategy* to solve a variety of problems.

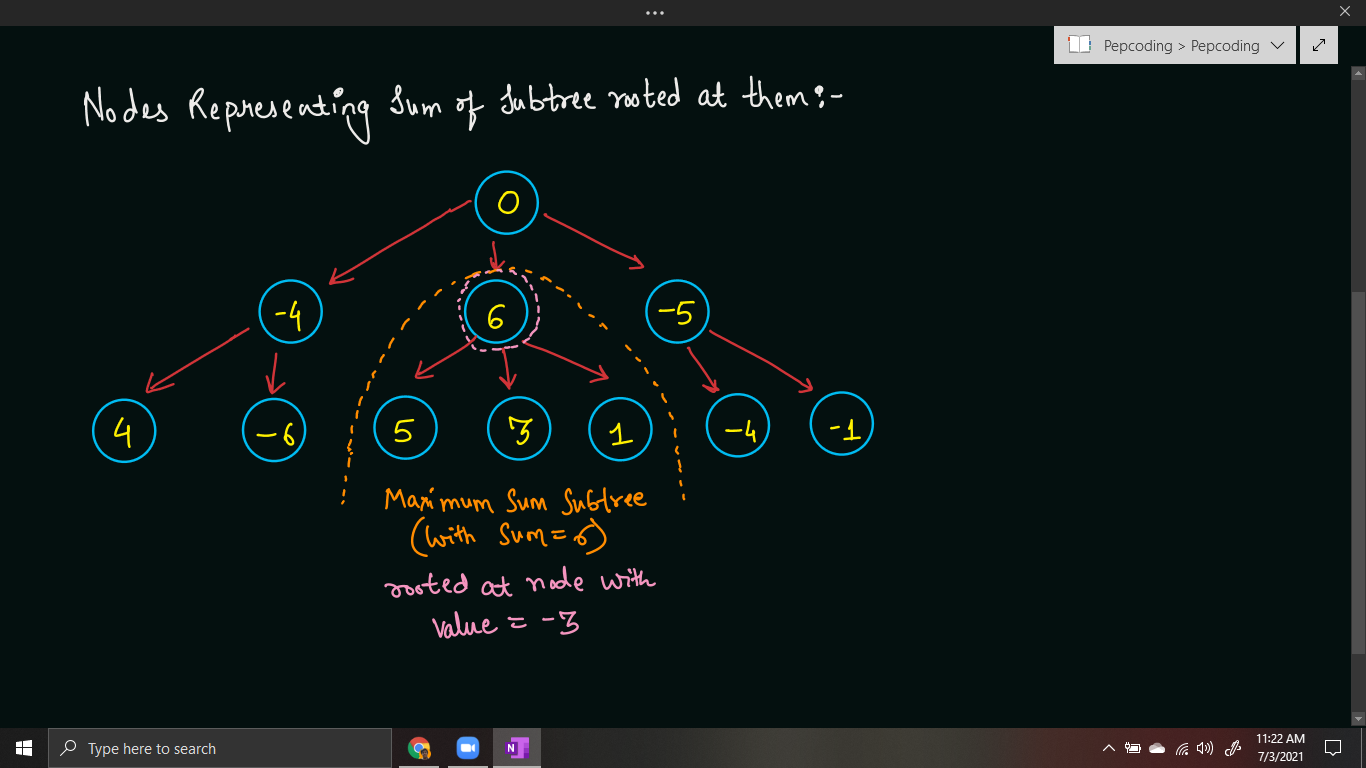
***Problem Statement:***

* You are given a partially written GenericTree class. (Input and Output is managed for you.)
* You are required to find and print the node which has the subtree with the largest sum. Also print the sum of the concerned subtree separated from node's value by an '@'.

For more details, check out the [question video](https://www.youtube.com/watch?v=TKqOZ3tp1D8&list=TLGGx6EvfVyBKd4wMzA3MjAyMQ).

***Example:***

******

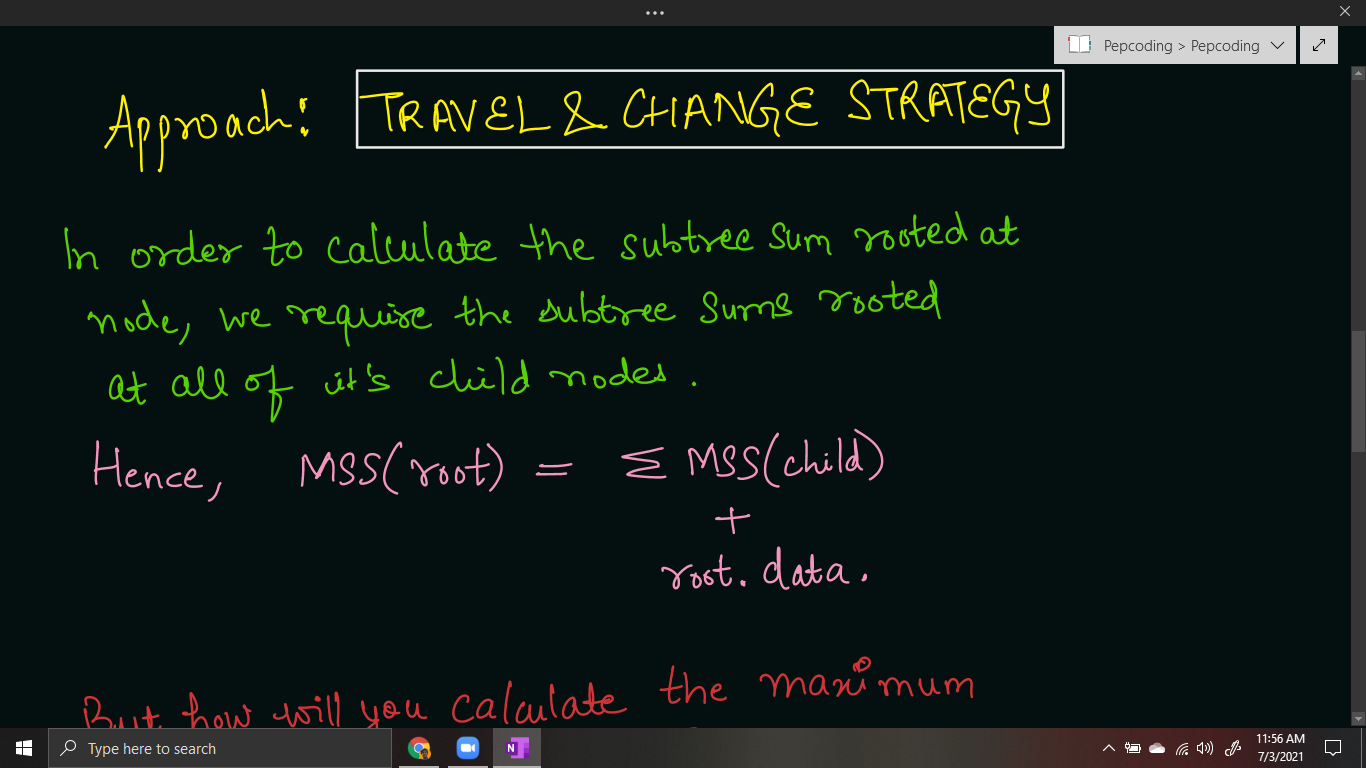
****

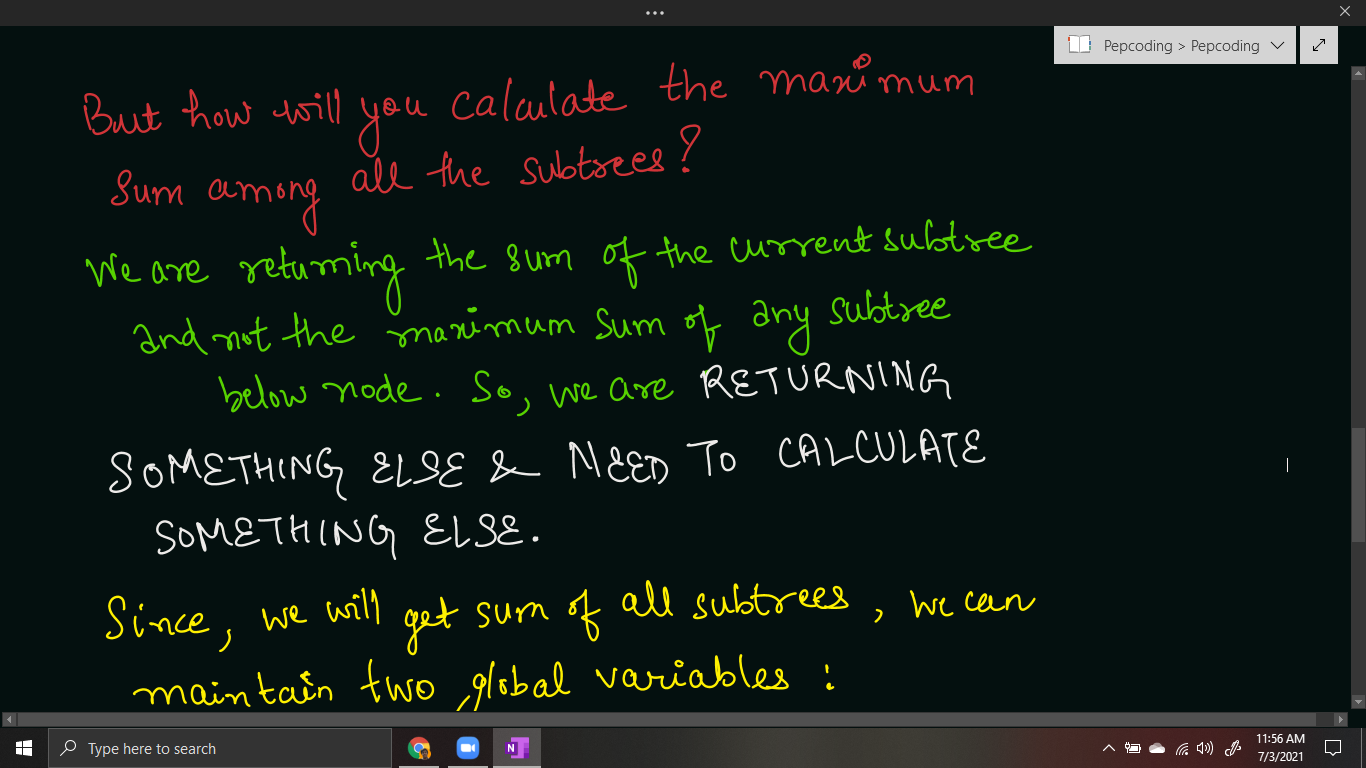
Hence, the answer will be: **-3@6**.

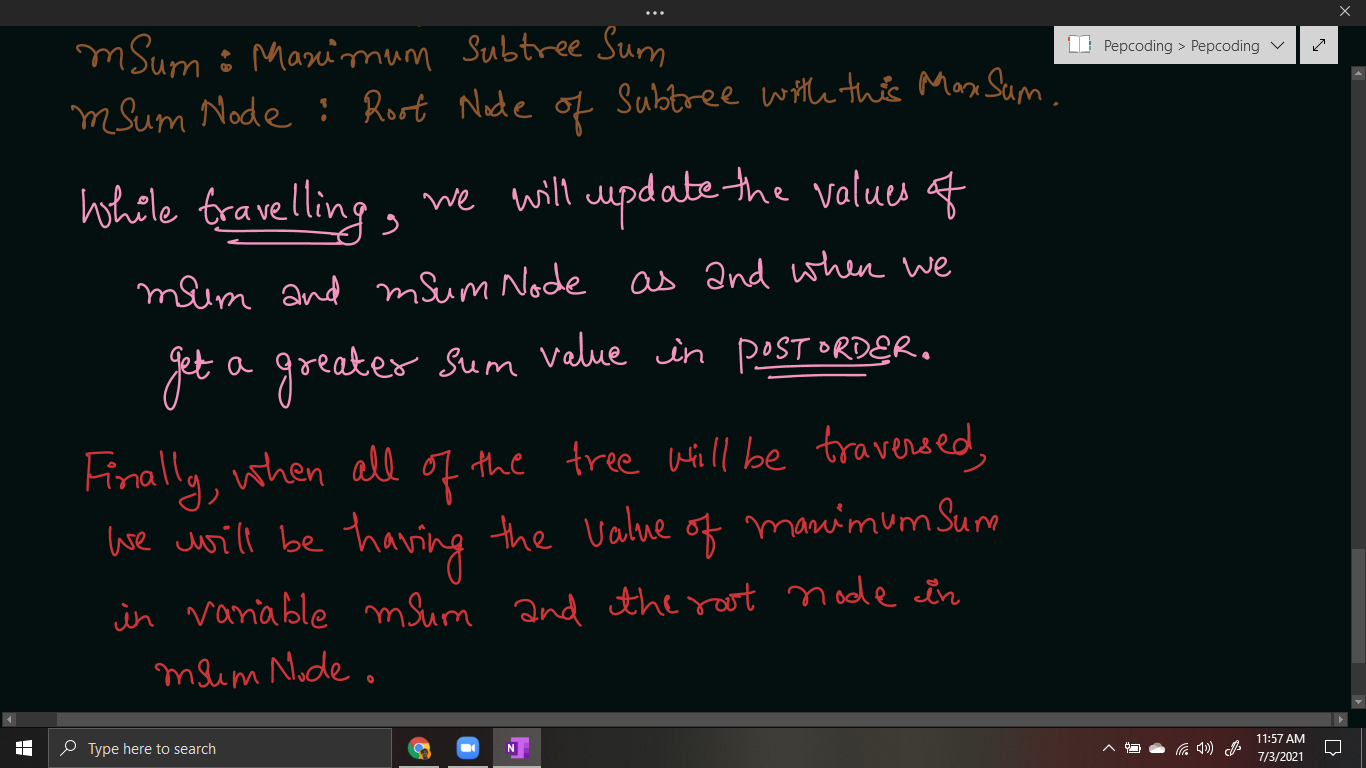
***Solution:***

Till this point, I can assume that you were able to figure out that we need to do ***postorder traversal*** of the tree: get some value from subtrees rooted at children, and then finally compute the result for the current node.

Now, let us look at the strategy we will be using in the current problem and further in a lot of problems.







***Pseudo Code/ Algorithm***

* Declare two static (global) variables ***mSum*** and ***mSumNode*** and initialize them as Integer.MIN\_VALUE.
* Initialize the current subtree ***sum = node.data*** (inside the function).
* Run a loop over all the child nodes of the current node:
  + Get the value of child Subtree Sum in a variable ***cstSum*** by calling ***nodeWithMaximumSubtreeSum*() on the child node**.
  + Add the child Subtree Sum to the current sum by ***sum += cstSum***.
* Now, check if the current subtree sum (in *sum*) is greater than the maximum subtree sum so far (in *mSum*):
  + If ***sum is greater than mSum***, then **update *mSum = sum* and *mSumNode = node.data***.
* Return the current subtree sum, ***return* *sum***.

*Note*: Output is managed already by the statement:

***System.out.println(mSumNode + "@" + mSum);***

***Implementation***

*Note*: Before reading the Code, we recommend that you must try to come up with the solution on your own. Now, hoping that you have tried by yourself, here is the Java code.

import java.io.\*;

import java.util.\*;

public class Main {

private static class Node {

int data;

ArrayList<Node> children = new ArrayList<>();

}

public static void display(Node node) {

String str = node.data + " -> ";

for (Node child : node.children) {

str += child.data + ", ";

}

str += ".";

System.out.println(str);

for (Node child : node.children) {

display(child);

}

}

public static Node construct(int[] arr) {

Node root = null;

Stack<Node> st = new Stack<>();

for (int i = 0; i < arr.length; i++) {

if (arr[i] == -1) {

st.pop();

} else {

Node t = new Node();

t.data = arr[i];

if (st.size() > 0) {

st.peek().children.add(t);

} else {

root = t;

}

st.push(t);

}

}

return root;

}

static int mSum = Integer.MIN\_VALUE;

static int mSumNode = Integer.MIN\_VALUE;

public static int nodeWithMaximumSubtreeSum(Node node){

int sum = node.data;

for(Node child: node.children){

int cstSum = nodeWithMaximumSubtreeSum(child);

sum += cstSum;

}

if(sum > mSum){

mSum = sum;

mSumNode = node.data;

}

return sum;

}

public static void main(String[] args) throws Exception {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

int n = Integer.parseInt(br.readLine());

int[] arr = new int[n];

String[] values = br.readLine().split(" ");

for (int i = 0; i < n; i++) {

arr[i] = Integer.parseInt(values[i]);

}

Node root = construct(arr);

mSum = Integer.MIN\_VALUE;

mSumNode = 0;

nodeWithMaximumSubtreeSum(root);

System.out.println(mSumNode + "@" + mSum);

}

}

This code is written and explained by our team in the [solution video](https://www.youtube.com/watch?v=tDx3PPwgSxs&list=TLGGdRSrK_huXE0wMzA3MjAyMQ). Do check it out to understand the concept completely.

* What is the ***time complexity*** of the above code?

We are visiting each node exactly once, hence the total time complexity will be ***O(n)*** where n = number of nodes in the tree.

* What is the ***space complexity*** of the above code?

We are just taking two integer variables mSum and mSumNode to find the maximum subtree sum. Hence, we are taking ***O(1)*** *auxiliary space*.

However, again, due to recursion, the recursion call stack will take up ***O(d)*** space where d = maximum depth of the tree.

Hope that you liked the article on *Node With Maximum Subtree Sum in Generic Tree*.

Subscribe to Pepcoding’s youtube channel for more such amazing video content on Data Structures & Algorithms.

You can suggest any improvements to the article on our telegram channel, or on the youtube channel’s comment section.

Article Contributed by: [Archit Aggarwal](https://www.linkedin.com/in/archit-aggarwal-6a7716189/)