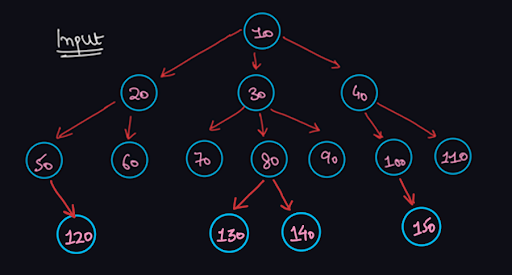
**1. Problem Discussion :**

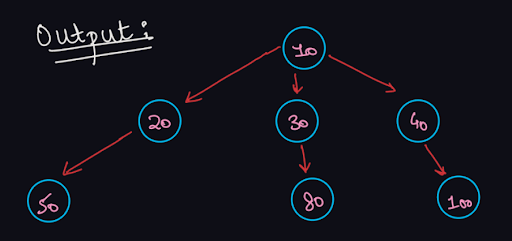
You are given a partially written GenericTree class. (Input and Output is managed for you.) You are required to complete the body of removeLeaves function. The function is expected to remove all leaves from the tree. A leaf node is a node which has no children, i.e. the size of the children array is 0. There will be at least one child of the root node, i.e. root node will not be a leaf node. For more details, check out the question video.

For more clarity of the question, watch the question video

Play Video

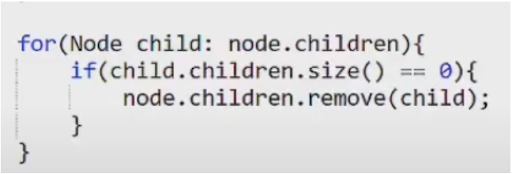
Example:

****

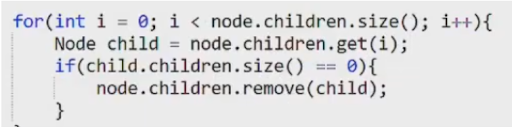
****

Approach :

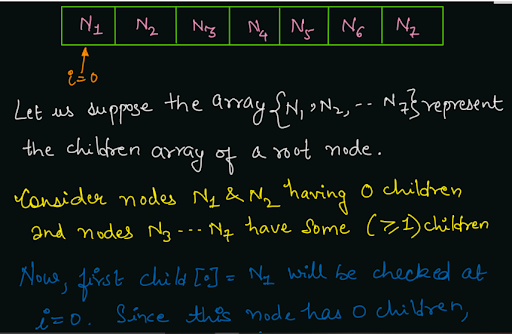
We need to remove all the leaf nodes, i.e. we will check whether any node has 0 child or not, and if yes, then we will delete this node. Thus, standing at a root node, we will check whether any of its child nodes have 0 children, and delete those child nodes which are leaf. How will we delete nodes from children arraylist? Can we use the following code to achieve it?

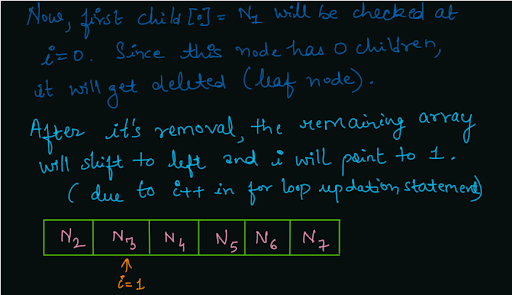
****

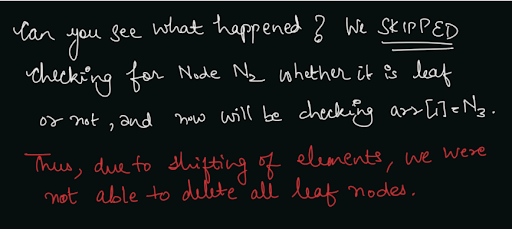
Answer is no. Inside a for-each loop, we cannot remove an element from the same array on which the for loop is iterating on. This code will give a Concurrent Modification Exception. Hence, we should run a normal for loop (using indices) instead of for-each loop. Now, will this code do the work?

****

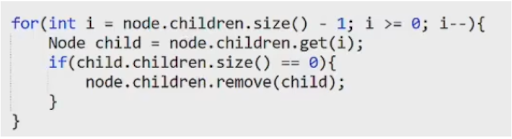
Answer is again no. There is a problem in this traversal also. Since, removing an element from arraylist shifts the remaining elements from the right to one place in the left, we will skip checking for some child nodes using this code. Let us take an example:

****

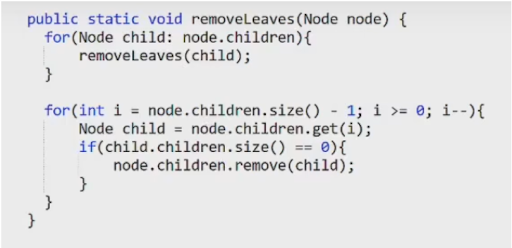
****

****

So, what is the solution to this unexpected problem? Instead of traversing the children arraylist from left to right, we can traverse it from right to left. If a node at index i gets deleted, then all nodes after i (from i + 1) will shift and there will be no impact on index i - 1. Thus, if we traverse it from length -1 to 0, it will cause no problem as the value at any index i will not change irrespective of whether the node at index i + 1 was deleted or not. So, we came to the following solution:

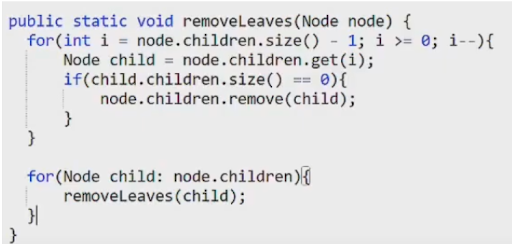
****

Now, we need to remove all the leaf nodes in the subtree of children as well. So, should we call to removeLeaves() function for the child nodes in the node-pre area or node-post area.

****

Above code (removeLeaves(child) in node-pre area) will be a big BLUNDER. You know why? If we call for removeLeaves for the child nodes first, then all the leaf nodes will get removed first. Then after returning to the current node, the current node will itself behave as a leaf node, and thus will get deleted. Eventually, all the nodes will keep on deleting (except the root of the tree). And the final tree will consist of only 1 node. Thus, we would have deleted non-leaf nodes also using the above code. So, the right solution is to call for the removeLeaves() on child nodes, in the node-post area, i.e. after removing the leaf nodes of children of the current node. In this way, we will call for removeLeaves() on only non-leaf nodes (which were not removed), and eventually only the leaf nodes will get deleted.

Pseudo Code

****

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

ConsoleJava

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;

}

public static void removeLeaves(Node node) {

for (int i = node.children.size() - 1; i >= 0; i--) {

Node child = node.children.get(i);

if (child.children.size() == 0) {

node.children.remove(i);

}

}

for (Node child : node.children) {

removeLeaves(child);

}

}

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);

removeLeaves(root);

display(root);

}

}

Analysis:

Time Complexity:

We are traversing all the nodes once, hence the time complexity will be O(n) where n = number of nodes in the generic tree. Space Complexity: We are not using any extra space in the form of any auxiliary data structure. Hence the space complexity is O(1). Note: We are using recursion which does take stack space of O(d) where d = maximum depth of the generic tree.

Follow Up:

1 . What if we were asked to remove leaves only if the node's value is equal to some target value?

In this case, along with checking whether the size of children of the node is 0 or not, we will also check whether the node itself has data equal to target value or not. Rest, the concept shall remain the same. Hope that you liked the article on Remove Leaves 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.