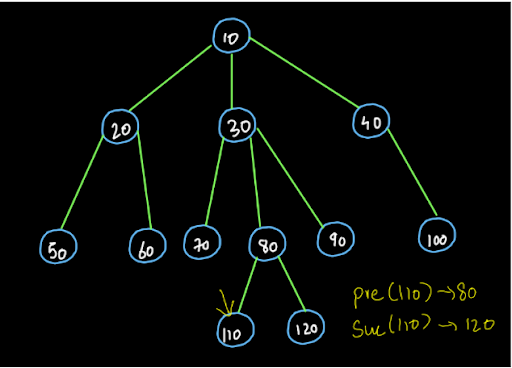
**1. Problem Discussion:**

We will be given an element and we will have to return its pre order predecessor and successor. Have a look at the diagram given below:

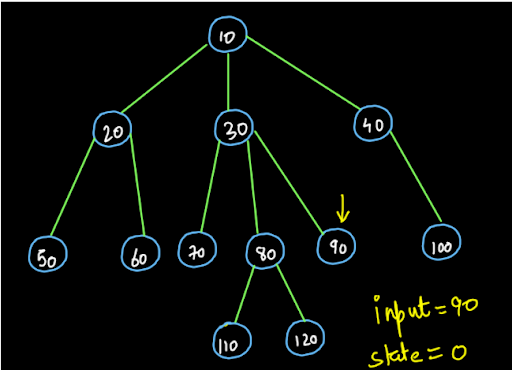
****

In the above given generic tree let's say that we have been asked to find the successor and predecessor of 110. So, the predecessor of 110 in pre-order is 80 and its successor is 120.

Also the root node will not have any predecessor and the last node of the tree will not have any successor. So, for them we have to return null.

**2. Approach:**

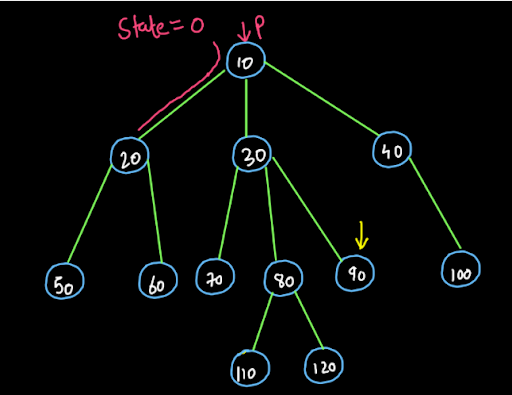
1• We will traverse the tree and follow the Euler path. We will take a variable named “state” and its value is 0 initially. We will traverse the tree till we reach at the element that has been provided to us as the input. When we reach at the input element, the state will be increased by 1 and after crossing it, the state will become 2 immediately. What? This sounds so confusing, right? Don't worry. Let's have a step-by-step understanding of the entire procedure.

****

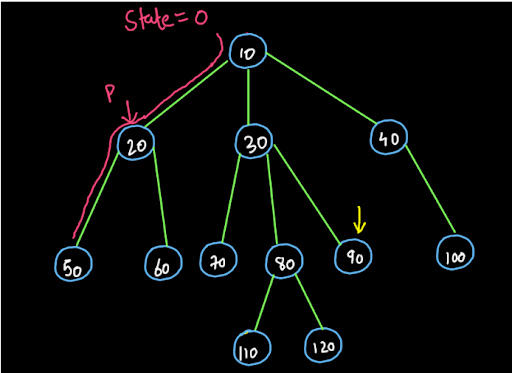
2• For the above tree we have to find the predecessor and successor of the node (90). So, the input that we received is 90. We have kept a variable state=0 initially. Now, let's start tracing the Euler path and see what the procedure says.

2• We start from the root node. We check whether the root node is the node with value 90 or not. Since the root node is not the node we are looking for, we will not change the state variable and we will move forward. Before moving forward in the Euler path, we keep our predecessor at the root node. (see fig-3)

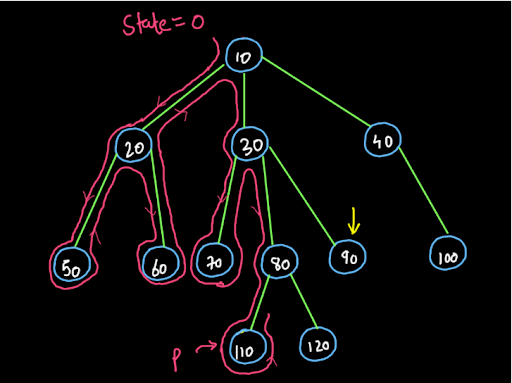
Note: We have denoted the state 0 with pink color. So, the Euler path in pink color signifies that we are in state 0.

****

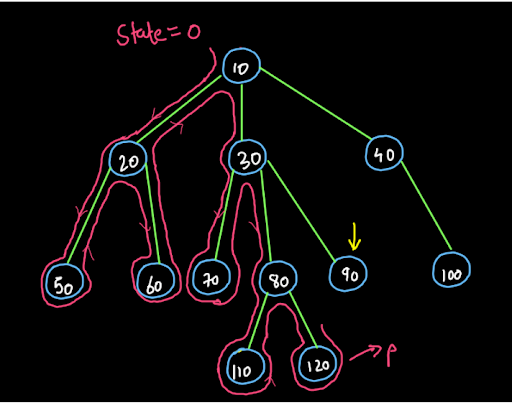
3• Again, we are at a node (node (20)) that does not match our input value. So, we keep on moving in the Euler path and the predecessor (shown by p in the diagram) will now move to this node i.e. on node (20) and we will move to the next node in the Euler path.

****

4• So, this is what we have to do till we reach the node that matches our input value. We will remain at state 0 only and the predecessor will keep on changing. We recommend you trace the Euler path till node (110). The Euler path, state and the predecessor for the same is shown below. You may verify your method.

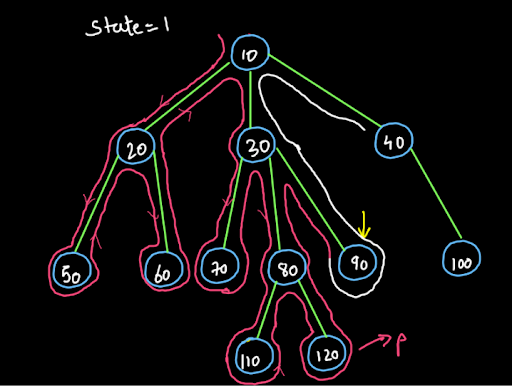
****

5• So, the predecessor is at node (110) and the state is still 0. Now, we move towards the next node. This node also does not match the input. So, the predecessor moves to this node (120) and we will move forward with our Euler Path.

****

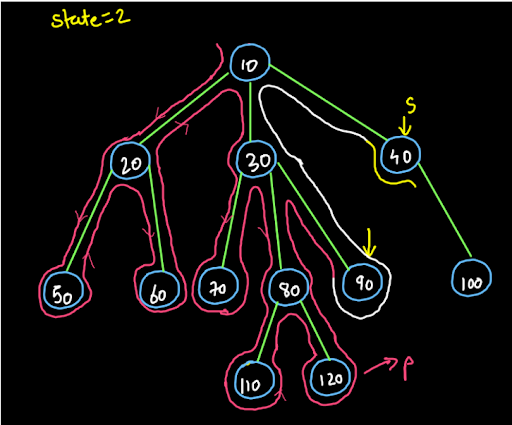
6• Now, we will move to the node (90). This is the node that we were searching for as the data given to us as the input i.e. 90 matches the data of the node.

7• When we reach this node the state changes to 1 (denoted by white color in the diagram). Also, now we will not move our predecessor and it will remain at node (120). Then, we will move forward towards the next node. So, dear reader, do you understand what happened here? When we found the node with the same input data, we changed the state to 1 indicating that we have found the predecessor. The predecessor remains at its correct position as we only change the predecessor if the state is 0. Since it has now become 1, we will not change its value. Thus the predecessor is found successfully. (see fig-7)

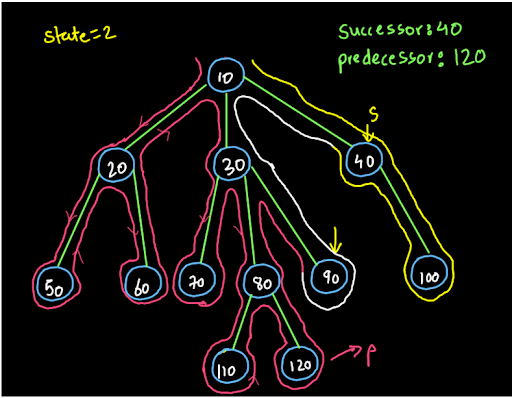
****

Now, the next node after this node in the tree should be the successor. So, let's see the next step in our procedure:

8• We reach at node (40) and since our state value was 1, we keep our successor on this node (40). We know that this should be our successor as the value of state changed from 0 to 1 in the previous node only. So, we will now make the state value=2.

****

Now we have got our successor. How? When we make the state value=2, we are ensuring that the successor does not change as the successor only sits at a particular node if the state value coming from the previous node is 1. So, the state value remains 2 now throughout the remaining Euler path and neither the successor nor the predecessor will change as successor changes if the state value is 1 and predecessor changes if the state value is 0. (see fig-9)

****

So dear reader, this is the entire procedure that we will follow to get the predecessor and the successor of any element in the tree. If you have any doubts about it, you may refer to the solution video to understand the above procedure.

**3. Recursion:**

Now, we know the procedure that we have to follow but, we don't know how we are going to traverse the tree in Euler Path. Well, actually we do know how to do it. We have been doing it for almost every question in the generic trees. Yes, you are right!!! Recursion is the answer.

Actually we are using recursion just to traverse every node and check whether the value of the node matches our input value. So, the function will be called upon recursively for all the child nodes of the root node .

Note: Here we are not discussing the recursion in terms of expectation, faith and relation and the low-level-thinking because the procedure explained above in the form of Euler Path traversal is itself the low-level explanation of this process and we are not using recursion to solve this problem particularly. We are not solving the small problem of the same kind using recursion. Here it is only used for traversing the nodes.

**4. PseudoCode:**

1• We will use the "travel-and-change" approach that we have learnt previously. We will keep a variable state=0 initially. Then, we will search for the element that we have received as our input and our state will remain 0 until we find it and we will keep our predecessor on every node that comes in our recursion path till we reach the input element.

2• After we find the input element the state will change to 1 but the predecessor will not come at this node. The predecessor will remain at the last node in state 0.

3• After this, when we reach the next element the state is 1. So, we keep our successor on this node and make our state equal to 2.

4• So, now that value of state is 2 and we will not do anything now. So, the successor remains at the first element in state 1.

5• We have got our successor and predecessor.

ConsoleJava

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 Node predecessor;

static Node successor;

static int state;

public static void predecessorAndSuccessor(Node node, int data) {

if (state == 0) {

if (node.data == data) {

state++;

} else {

predecessor = node;

}

} else if (state == 1) {

successor = node;

state++;

}

for (Node child : node.children) {

predecessorAndSuccessor(child, data);

}

}

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]);

}

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

Node root = construct(arr);

predecessor = null;

successor = null;

state = 0;

predecessorAndSuccessor(root, data);

if (predecessor == null) {

System.out.println("Predecessor = Not found");

} else {

System.out.println("Predecessor = " + predecessor.data);

}

if (successor == null) {

System.out.println("Successor = Not found");

} else {

System.out.println("Successor = " + successor.data);

}

}

}

**5. Analysis:**

Time Complexity: O(n)

The time complexity of the above solution is O(n) as we are traversing all the nodes of the tree.

Space Complexity: O(1)

The space complexity of the above solution is O(1) as we have not used any extra memory. But, if we consider the recursion space then it is O(logn) as the max height of the recursion stack will be equal to height of the tree i.e. O(logn).