Untitled

22 November 2025 09:06

What is an `ArrayList`?

An `ArrayList` is part of the **Java Collections Framework**.

- **Resizable Array:** Unlike a standard Java array, which has a fixed size, an `ArrayList` can grow and shrink dynamically as you add or remove elements.
- **Ordered Collection:** It maintains the insertion order of elements.
- **Allows Duplicates:** You can store the same element multiple times.
- **Allows Nulls:** You can store `null` values.
- **Generic:** You must specify the type of objects it will hold when you declare it (e.g., `ArrayList<String>`).
- **Not Thread-Safe:** If multiple threads access it concurrently, you must externalize the synchronization.

It is fundamental and very useful class.

How to Use `ArrayList`

1. Declaration and Initialization

You always specify the **object type** inside the angle brackets (`<>`). You cannot store primitive types (like `int`, `char`, `double`) directly; you must use their corresponding **wrapper classes** (`Integer`, `Character`, `Double`).

Java

```java
import java.util.ArrayList;

// 1. Common way (best practice: use the interface type on the left)
List<String> names = new ArrayList<String>();

// 2. Using the diamond operator (Java 7+ simplification)
List<Integer> ages = new ArrayList<>();

// 3. To initialize with a specific capacity (optional, for optimization)
ArrayList<Double> prices = new ArrayList<>(50);
```

2. Common Methods

The following methods cover most of the basic operations you'll perform:

Method	Description	Example
`add(E e)`	Appends the specified element to the end of the list.	`names.add("Alice");`
`add(int index, E e)`	Inserts the element at the specified position.	`names.add(0, "Bob");`
`get(int index)`	Returns the element at the specified position.	`String first = names.get(0);`
`set(int index, E e)`	Replaces the element at the specified position.	`names.set(1, "Charlie");`
`remove(int index)`	Removes the element at the specified position.	`names.remove(0);`
`size()`	Returns the number of elements in the list.	`int count = names.size();`
`contains(Object o)`	Returns `true` if the list contains the specified element.	`boolean hasBob = names.contains("Bob");`
`isEmpty()`	Returns `true` if the list contains no elements.	`boolean empty = names.isEmpty();`
`clear()`	Removes all elements from the list.	`names.clear();`

Export to Sheets

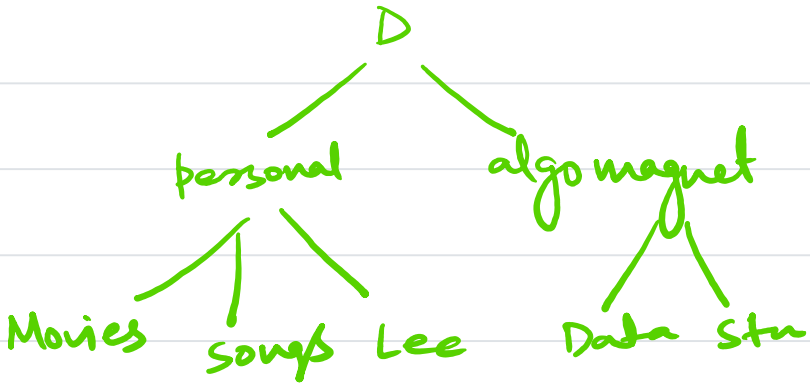
```java
import java.util.ArrayList;
import java.util.List;
public class main{
    public static void main(String[] args){
        List<Integer> number = new ArrayList<>();
        System.out.println(number.size());
        number.add(10);
        System.out.println(number.size());
    }
}
```

22 June 2022 18:40

Array and linked list have linear form of information

List is dynamic and size can grow.
Stack and queue were like array list but had the discipline of LIFO and FIFO.

Like folder information are not linear in nature.



This information is called hierarchical.

Nodes have their own data and information about the children nodes.

Terminologies:
Parents and children
Leaf nodes
Ancestors
Descendants
Root node

```java
public class Main {
    private class Node {
        int data;
        ArrayList<Node> children = new ArrayList<>();
    }

    public static void main(String[] args) {
        Node root;

    }

}
```

```java
import java.util.ArrayList;
import java.util.List;
public class genericTree{
    private class Node{
        int data;
        ArrayList<Node> children = new ArrayList<>();
    }

    public static void main(String[] args){
        Node root;
    }
}
```

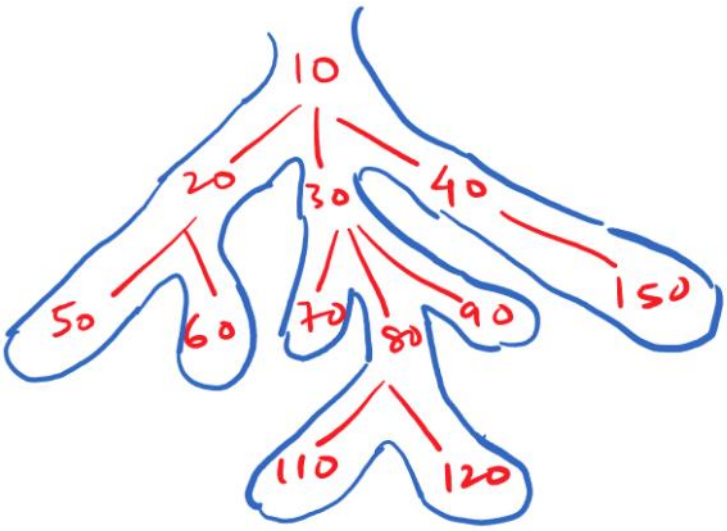
The entire tree is represented by one node and that is called as the root node.
[**Root node**] represents a given tree.

Use this template to start practicing the coding for generic tree
https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/

genericTreeTemplate.java

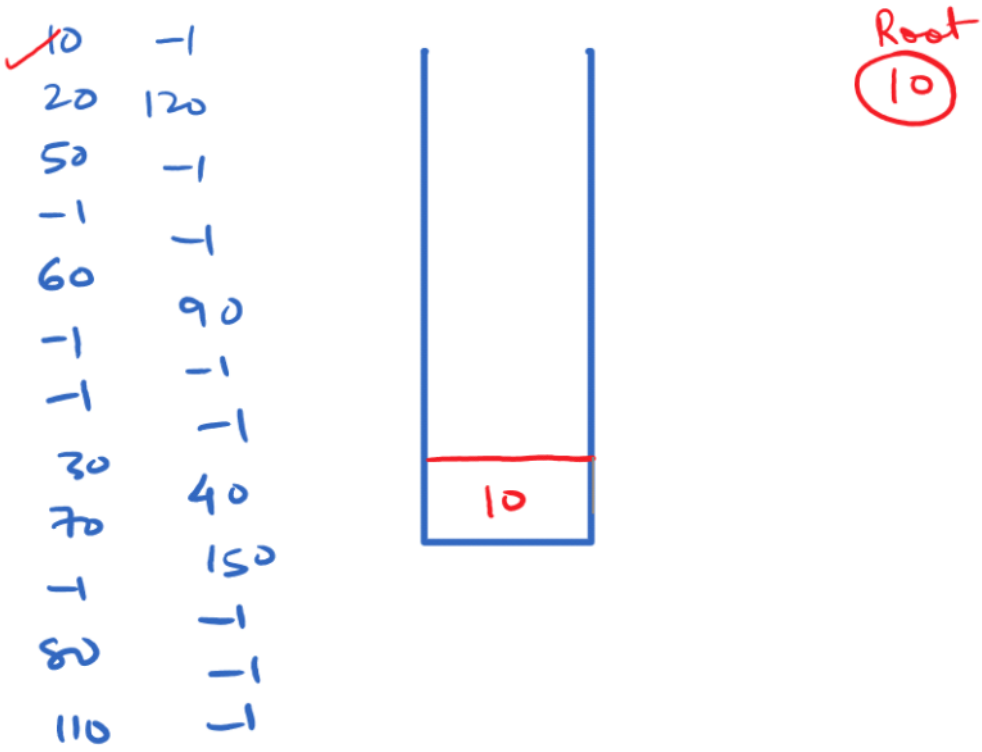
Generic tree constructor

22 June 2022 18:47

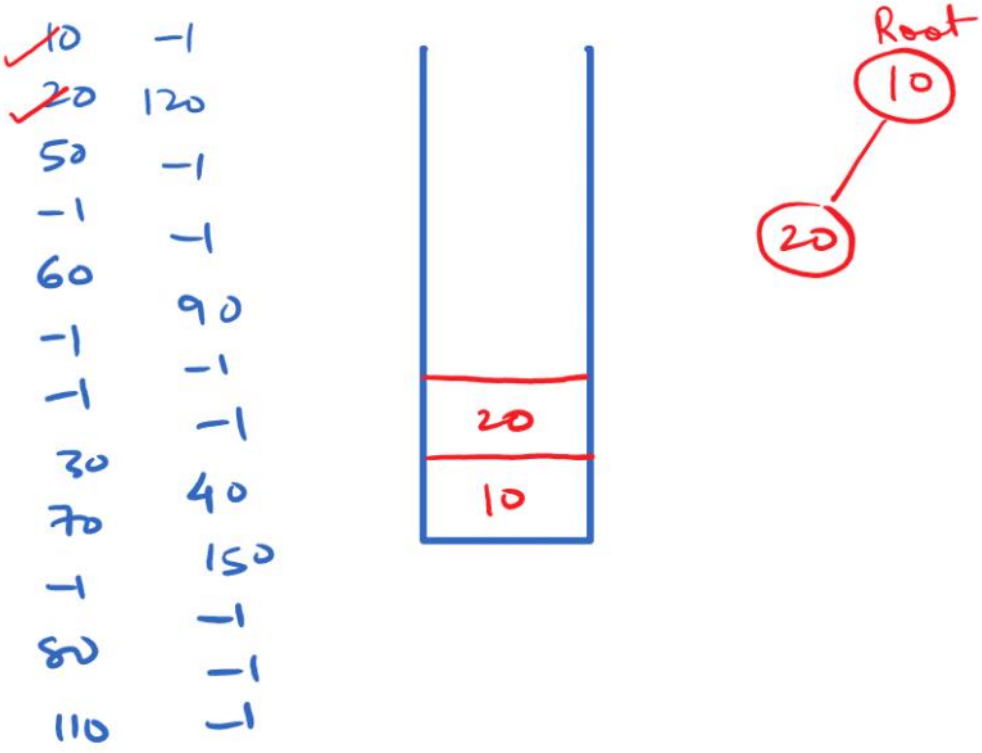


10	-1
20	120
50	-1
-1	-1
60	90
-1	-1
-1	-1
30	40
70	150
-1	-1
80	-1
110	-1

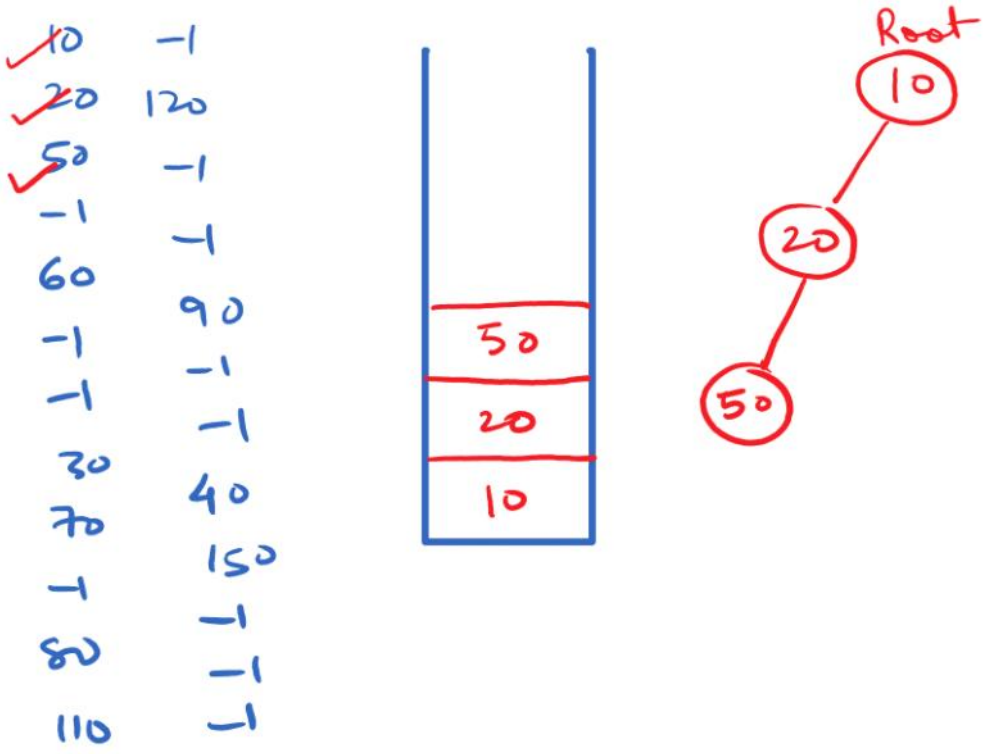
The data in the right will be passed as an input as an array, and we will use stack for this algorithm.



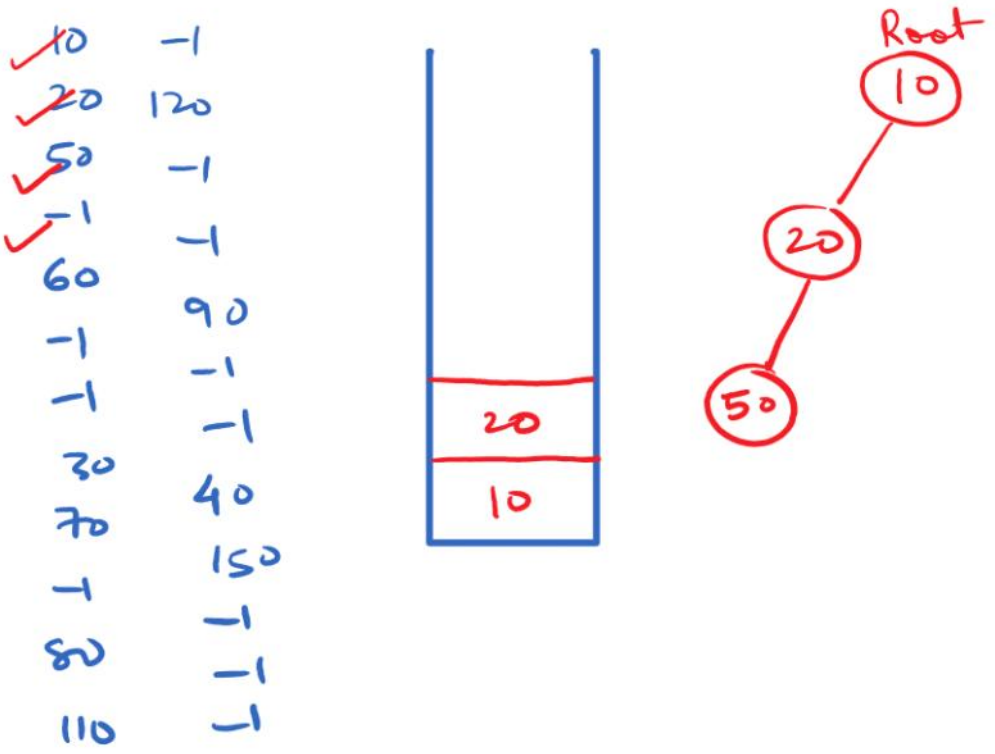
Pehle 10 ko parse kiya, node banaya, stack khali tha to node ko root banaya aur stack me push kar diya



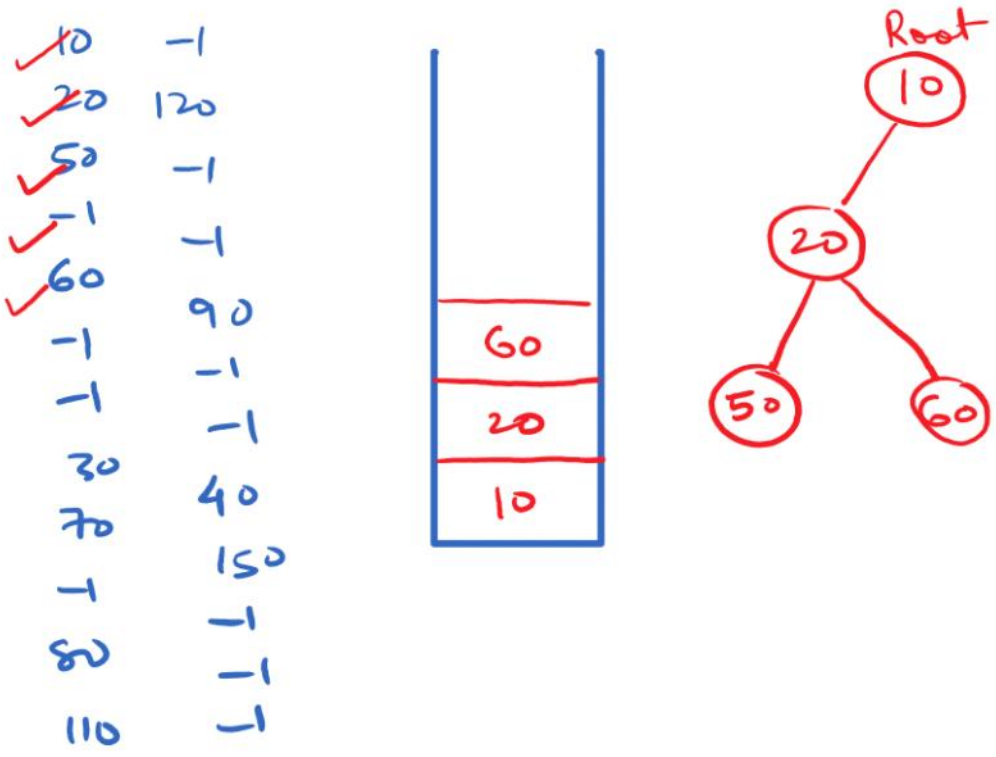
20 parse kiya, node banaya, node me data set kiya, stack ke top pe 10 dekha to 10 ka children bana diya aure stack me 20 push kar diya



50 parse kiya, node banaya, data set kiya, stack ke top pe 20 dekha to 20 ka children bana diya aur stack me 50 push kar diya



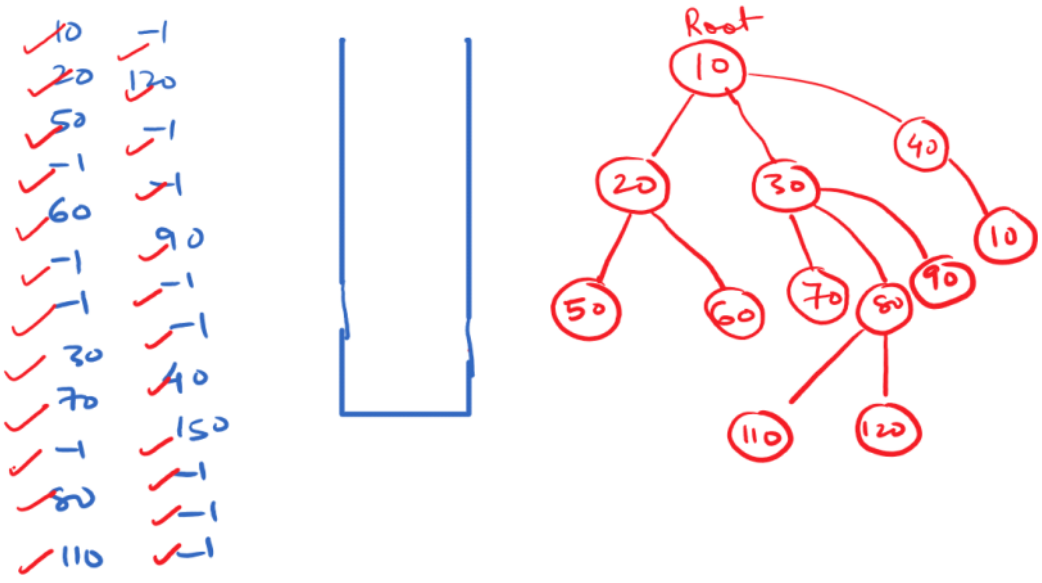
-1 parse kiya to pop kar diya



60 ko parse kiya, node bana ke data set kiya, stack

ke top pe 20 ko dekha to uska children bana diya aur stack me 60 push kar diya

Same kaam karte jayenge to hamara tree tayaar ho jayega



Now how we will code it.

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTree.java

Use this template to start practicing the coding for generic tree
https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeTemplate.java

🧠 Explanation of the Generic Tree Construction

The logic you are trying to implement is a standard, efficient way to build a Generic Tree from a linear array representation using a **Stack**. This array format is often called **Preorder Traversal with Markers**.

Array Interpretation

The array `arr` represents the nodes and the relationships between them:

- **Positive Integer (e.g., 10, 20):** Represents a new **Node** being created.
- `-1` **:** Represents the end of the current node's children list.

Stack Logic

The **Stack** is used to keep track of the **active path** from the root to the node whose children we are currently processing. The node at the **top of the stack (** `st.peek()` **)** is always the **parent** for the next incoming node.

1. **Start:** The stack is empty.
2. `arr[i]` **is a Node (e.g., 10):**
    - Create a new Node `t`.
    - If the stack is empty, `t` is the **Root**.
    - If the stack is **not** empty, `t` is a child of `st.peek()`.
    - Push `t` onto the stack (it becomes the new parent).
3. `arr[i]` **is** `-1` **:**
    - The children list for the node at `st.peek()` is complete.
    - Pop the node from the stack. The new top of the stack becomes the parent for the next sequence of children.

Steps

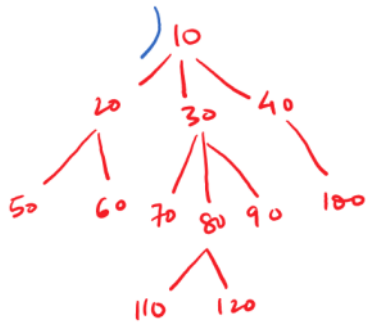
1. **Initialize:** Create an empty **Stack**, st. The `root` of the tree is initially $null$.
2. **Iterate:** Traverse the input array arr from the beginning to the end (index $i = 0$ to $arr.length - 1$).
3. **Process Element ($arr[i]$):**
   - **Case 1: $arr[i] = -1$ (End of Children / Backtrack)**
     - **Action:** Pop the top element from the stack, st. This signifies that all children for the node at the top of the stack have been processed, and we are moving back to its parent.
   - **Case 2: $arr[i] \neq -1$ (New Node Data)**
     - **Action A: Create Node:** Create a new `Node`, t, and set its `data` to $arr[i]$.
     - **Action B: Link Node:** Check the size of the stack:
       - **If $st.size() > 0$ (Not Root):** The new node t is a child of the node currently at the top of the stack.
         - Add t to the `children` list of $st.peek()$.
       - **If $st.size() = 0$ (Is Root):** The new node t is the **root** of the entire tree.
         - Set the tree's `root` to t.
     - **Action C: Push Node:** Push the newly created node, t, onto the stack, st. It now becomes the **active parent** for subsequent nodes until a **-1** is encountered.

Display a generic tree.

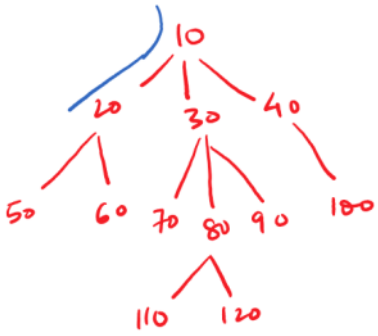
22 June 2022 21:18

Euler ki tarah chalta jayega.

Jab kisi ke left side me euler pahuchta hai to hum uske children print kar dete hai

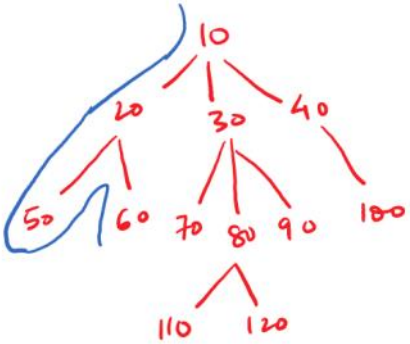


10 → 20, 30, 40, .



10 → 20, 30, 40, .

20 → 50, 60, .

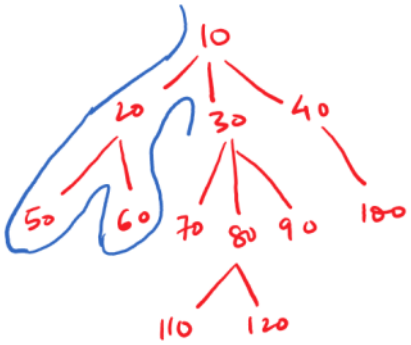


10 → 20, 30, 40, .

20 → 50, 60, .

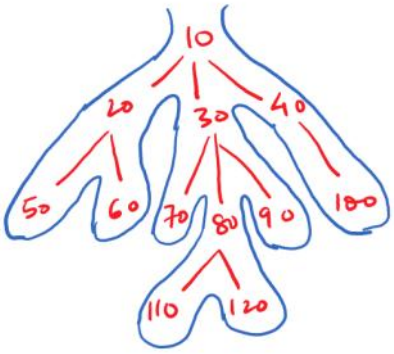
50 → .

60 →



10 → 20, 30, 40, .
20 → 50, 60, .
50 → .
60 → .
30 → 70, 80, 90, .

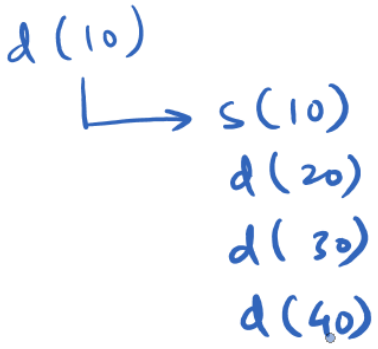
Jab 30 ke left me euler pahucha, to hum 30 ke samne 70 80 90 aur full stop print karenge



10 → 20, 30, 40, .
20 → 50, 60, .
50 → .
60 → .
30 → 70, 80, 90, .
70 → .
80 → 110, 120, .
110 → .
120 → .
90 → .
40 → .
100 → .

Isme faith aur expectation se solve karenge.

10 pass kiya gaya to pehle self ko print karaya, fir apne 20 ki family ko display karaya, fir 30 ki family ko display karaya, fir 40 ki family ko display karaya.



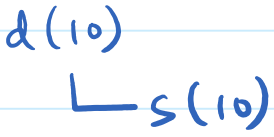
High level ko code karenge.

Faith ye hai display(20) khud ko display karna janta hai, display(30) khud ko display karna janta hai, display(40) khud ko display karna janta hai

To bas display of 10 me self ko print karenge aur display of their children invoke kar denge.

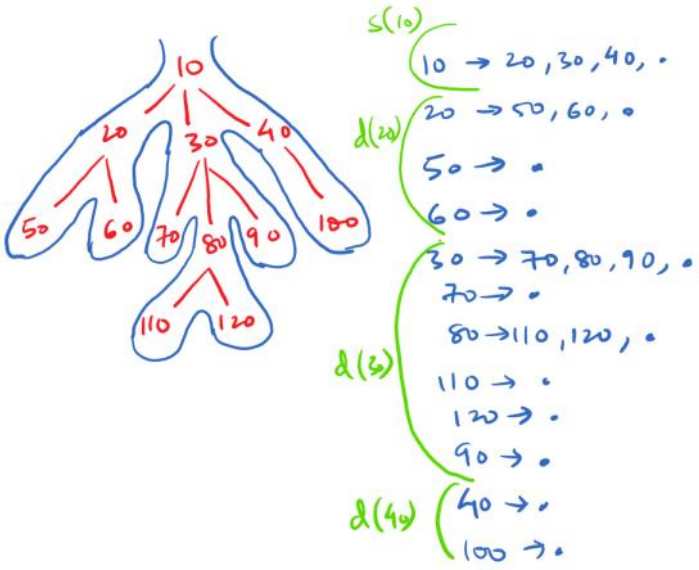
Agar maine display me 10 pass kiya to self ko print karne ke lie

```
private void display(Node node){
    String str = node.data + "->";
    for(Node child: node.children){
        str += child.data + ", ";
    }
    str += ".";

    System.out.println(str);
```



Sirf itna kaam hua hai, aur chote chote node apne apne ko display karana jante hai, ye hamara faith hai

```java
private 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);
    }
}
```



d(10) will print self of 10 and its family
And d(20), d(30), d(40) will print themselves and their families.

This part of code is self of 10

```
    private void display(Node node){
        String str = node.data + "->";
        for(Node child: node.children){
            str += child.data + ", ";
        }
        str += ".";

        System.out.println(str);
```

And give command to 20, 30 and 40

```
        for(Node child: node.children){
            display(child);
        }
    }
```

Full-fledged code is:

```
    // d(10) -> 10 will print itself and it's family
    // d(20), d(30), d(40) will print themselves and their families.
    // d(10) = s(10) + d(20) + d(30) + d(40);
    private 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);
        }
    }
```

Now do the dry run of code in the next page:
Do the low level analysis.

This analysis helps us to understand the recursion of tree.

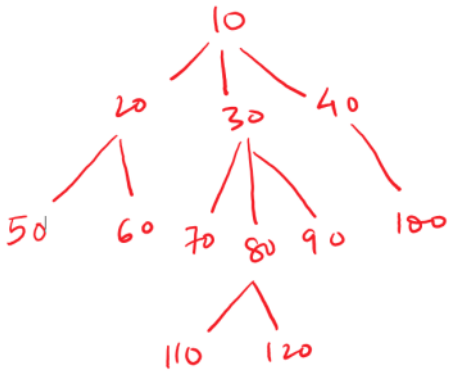
```
10->20,30,40,.
20->50,60,.
50->.
60->.
30->70,80,90,.
70->.
80->110,120,.
110->.
120->.
90->.
40->100,.
100->.
```

Dry run ground

24 June 2022 07:49

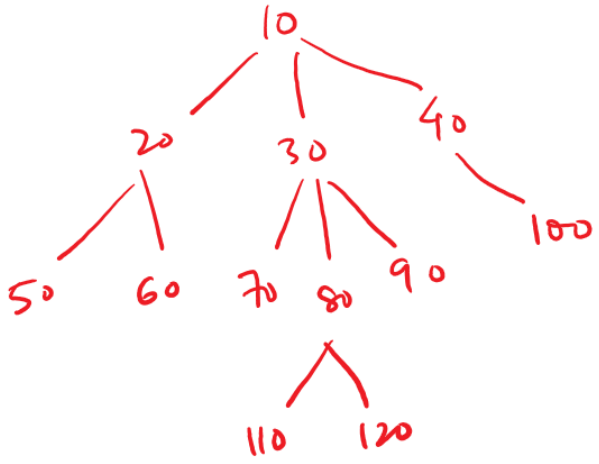
```
void display(Node* node)
{
    string str= to_string(node->data) + "-> ";
    for(Node* child: node->children){
        str += to_string(child->data) + ", ";
    }
    str += ".";
    cout<<str<<endl;
    for(Node* child: node->children){
        display(child);
    }
}
```

```
10-> 20, 30, 40, .
20-> 50, 60, .
50-> .
60-> .
30-> 70, 80, 90, .
70-> .
80-> 110, 120, .
110-> .
120-> .
90-> .
40-> 100, .
100-> .
```



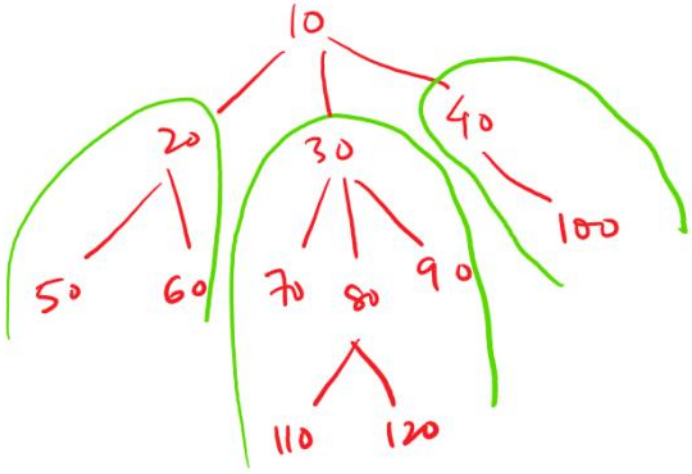
Size of a generic tree

23 June 2022 11:51



No of nodes = 12

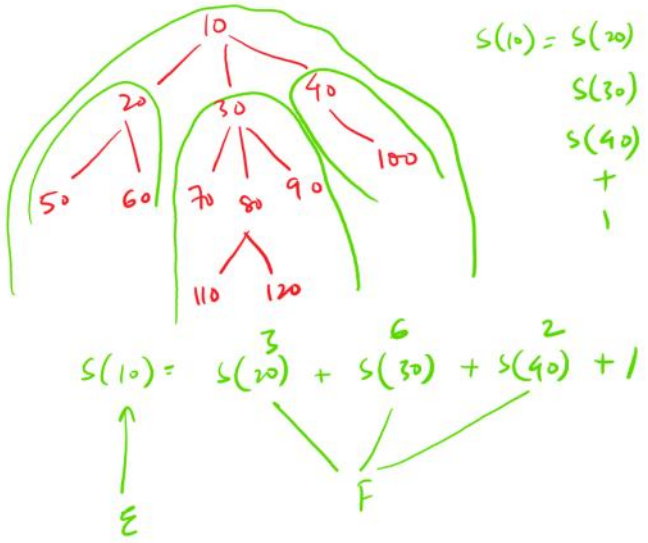
Expectation- faith technique



S(10) = S(20)
S(30)
S(40)
+
1

Expectation from size(10): wo 10 se niche sare nodes ko gin ke batayega

To hame faith rakhna hoga choti problem pe
Faith hai s(20), s(30), s(40)



s(10) = s(20) + s(30) + s(40) +1

```java
public int size(Node node) {
    // 1. Initialize the count for this subtree to zero.
    int s = 0;

    // 2. Recursively get the size of every child's subtree.
    for (Node child : node.children) {
        int childSubtreeSize = size(child);
        // Add the size of the child's entire branch to our total.
        s += childSubtreeSize;
    }

    // 3. Add 1 for the current node itself (Self-Inclusion).
    s = s + 1;

    // 4. Return the total size of the subtree.
    return s;
}
```

Step-by-Step Procedure

1. **Initialize Subtree Size:**
   - Initialize a local integer variable, s, to **0**. This variable will hold the running count of nodes in the current subtree.
2. **Iterate Through Children (Recursive Step):**
   - Loop through every `child` in the current `node` 's `children` list.
3. **Calculate Child Subtree Size:**
   - For each `child` , recursively call the `size` function: `childSubtreeSize = size(child)` . This will return the total count of nodes within that child's entire branch.
4. **Aggregate Size:**
   - Add the returned `childSubtreeSize` to the running total: $s = s + childSubtreeSize$. (This sums up the counts from all descendant branches.)
5. **Include Self:**
   - After iterating through all children, add 1 to the total: $s = s + 1$. This accounts for the **current node** itself.
6. **Return:**
   - Return the final value of s.

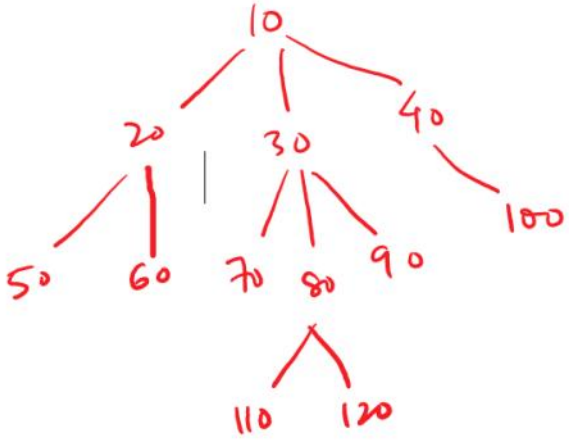
https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeSize.java

Dry run ground

23 June 2022 11:56

Do dry run of code

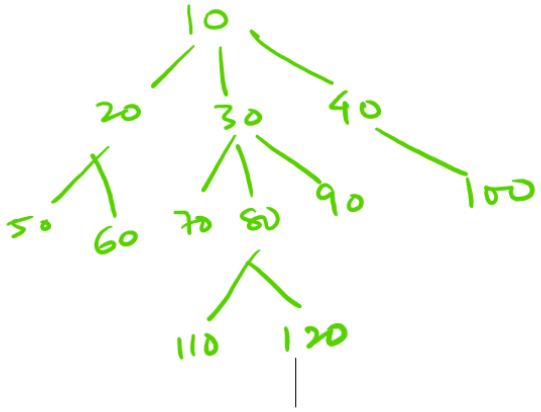
```
int size(Node *node)
{
    int s=0;
    for(Node* child: node->children){
        int cs= size(child);
        s = s+ cs;
    }
    s= s+1;
    return s;
}
```



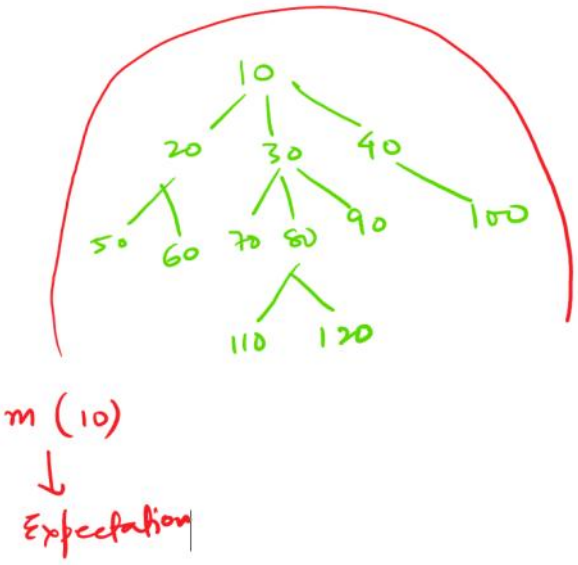
Euler ki help se dry run kar ke dekho kar pate ho ki nahi

23 June 2022 11:57

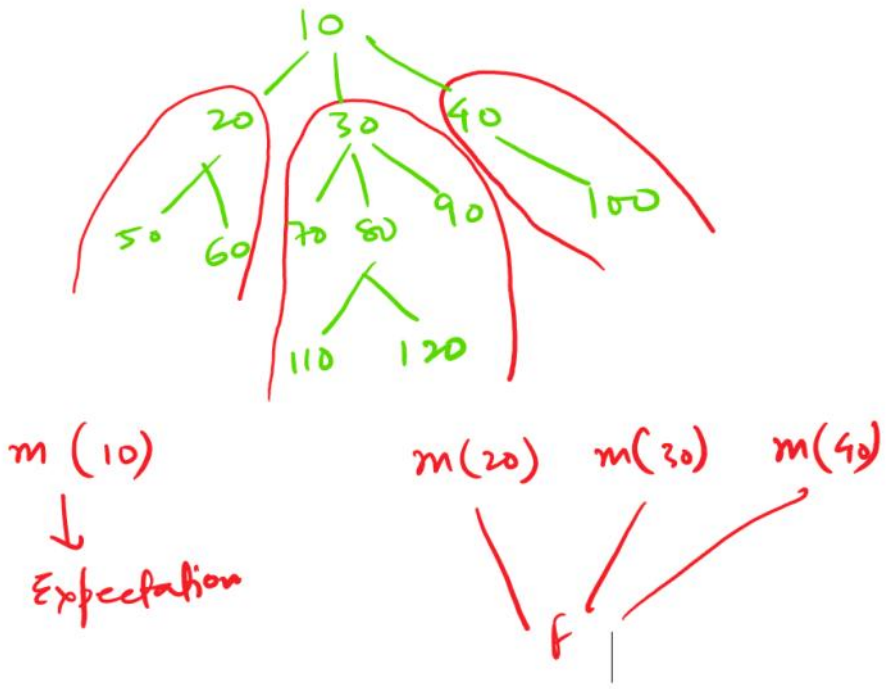
Maximum in a generic tree



Find node with maximum value.



Max(10) se ye expectation hai ki ye in sare nodes ko compare kar ke maximum nikale



Faith ye hai ki m(20),m(30), m(40) maximum dena jante hai

Then find max of 10, m(20), m(30), m(40)

Identity of max is minus infinity.

$x + 0 = x$

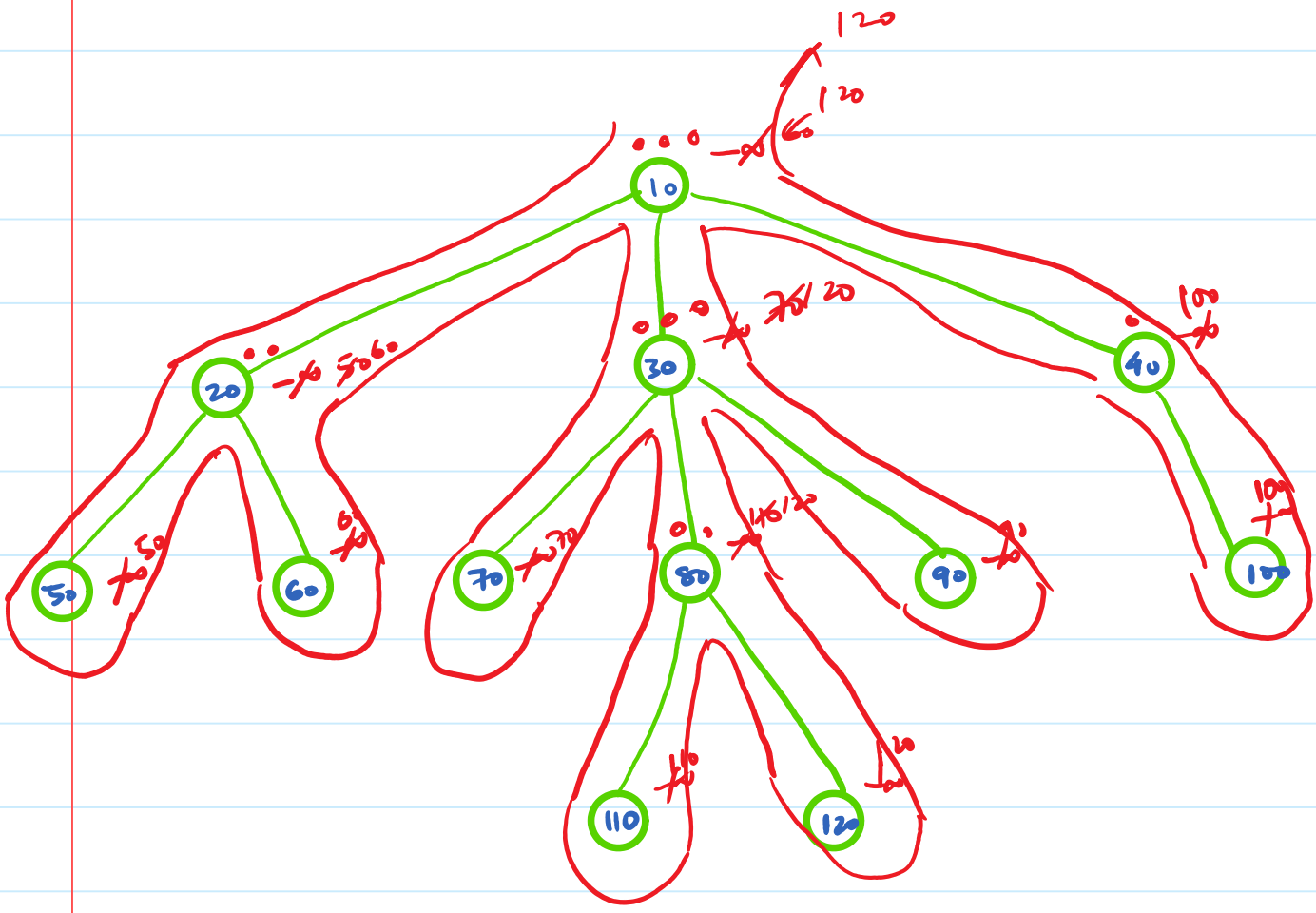
$I(+) = 0$

$x * 1 = x$

$I(*) = 1$

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeMax.java

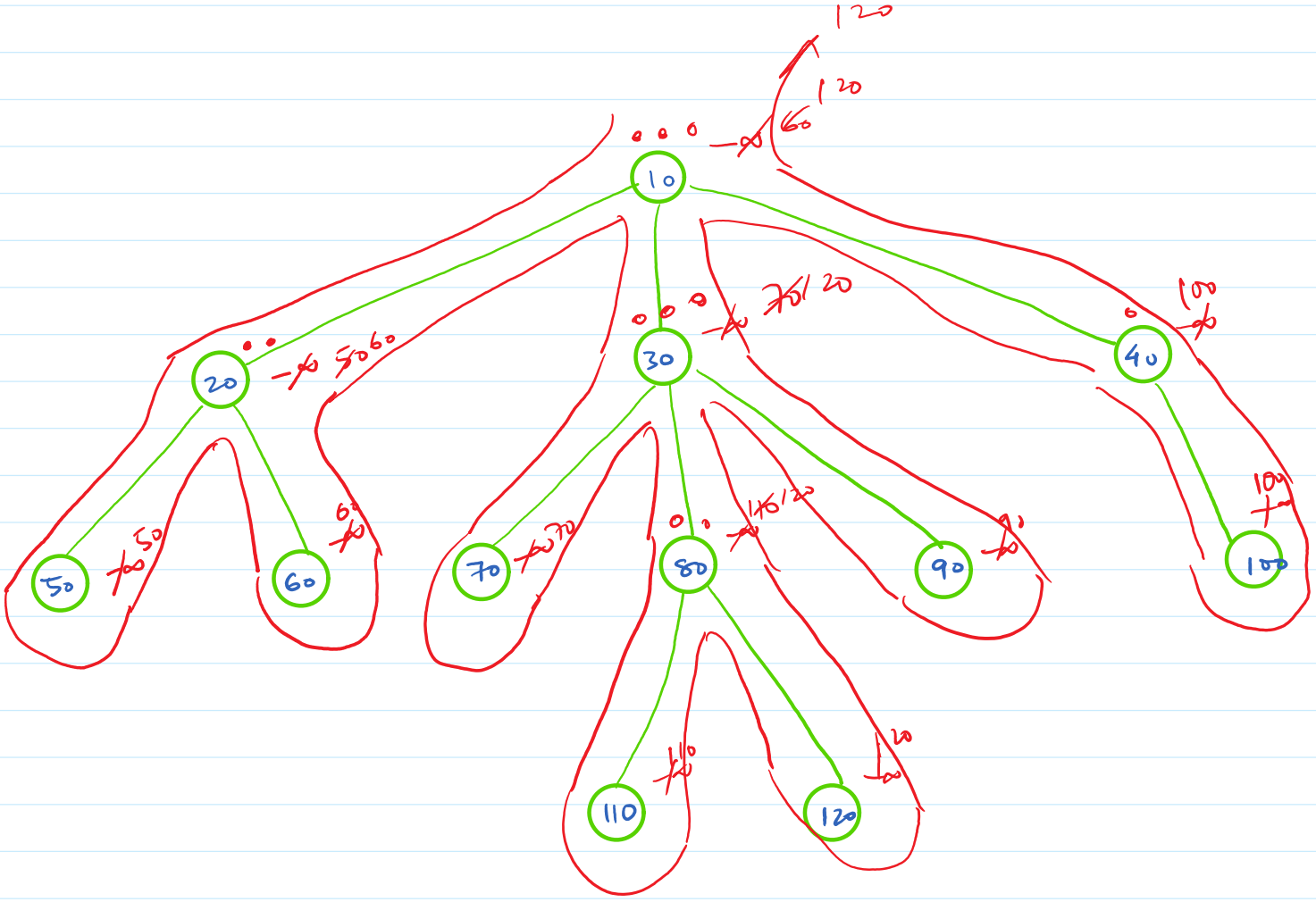
Kindly do this low level analysis



https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeMax.java

Dry run

26 June 2022 10:56

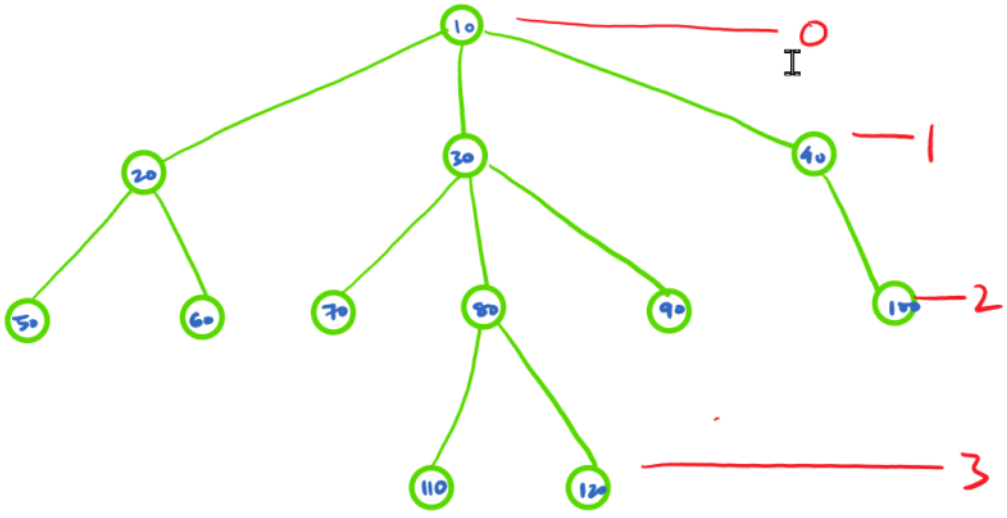


Height of the generic tree.

Height se pehle samajhiye depth ka concept.

Depth of node: How many edges in between from the root?

Height is:
-> depth of the deepest node.
-> maximum depth of a node in a given tree

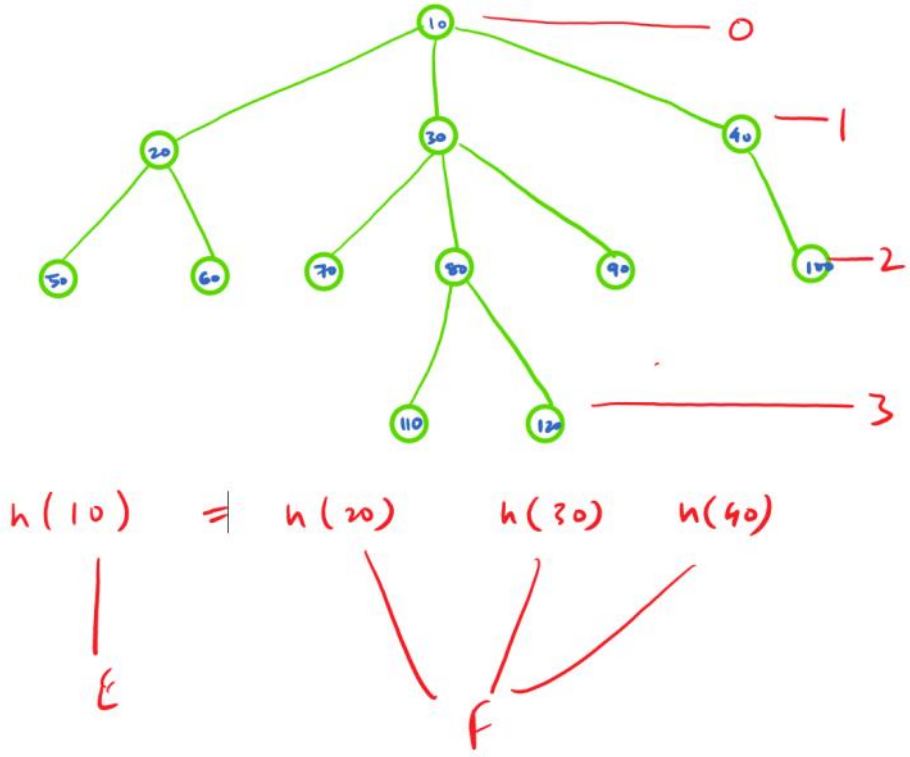


So height is 3.

We are counting depth in **EDGES** , not in **nodes.**

Do tareqe se height define karte hai,
Ek edges ke sath aur ek nodes ke sath.
Hum yaha edges se dekh rhe hai

Deepest node root se kitne edges door hai



https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeHeight.java

```
int height(Node* node)
{
    int ht=-1;
    for(Node* child: node->children){
        int ch= height(child);
        ht= max(ch, ht);
    }
    ht = ht+1;
    return ht;
}
```

Agar edge bola jaye to ht = -1 initialise karna hai

If there is no children and only one node, in that case height should be 0.

For an empty tree, height should be -1.

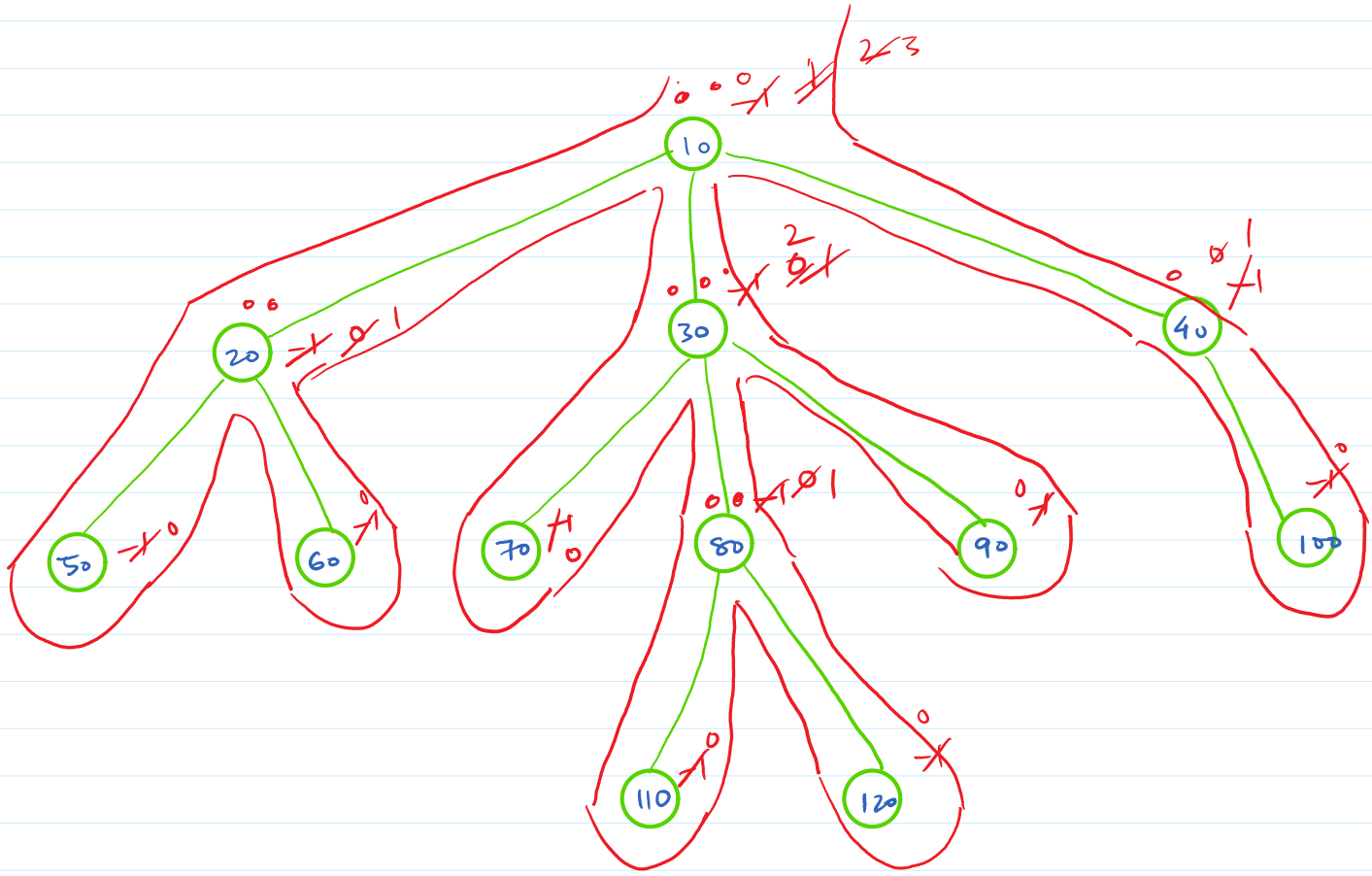
Kyunki jab sirf ek node hoga, to height tabhi 0 ayega

Kyunki hum edges ke term me define kar rhe hai to ht ko -1 initialize kiya hai
Agar nodes ke term me mange to 0 initialize karenge.

Iska dry run kare

Dry run

26 June 2022 10:58



🧠 1. Master the Recursive Template

Every algorithm on a Generic Tree (size, max, height, etc.) follows a variation of a single recursive template. Focus on these three questions for any node N:

Question	Logic for Traversal	Example: `Max(N)`
A. Recursive Faith	What am I assuming my children's calls will return?	I assume `Max(C)` will correctly return the maximum value in the entire subtree rooted at child C.
B. Meeting Point	How do I combine the children's results?	Find the overall maximum among all results returned by the children.
C. Self-Work	What specific work must I do for myself (N)?	Compare the maximum found in step B with my own data, $N.data$.
D. Return	What do I return to my parent?	The final maximum value found among all children and myself.

Conclusion: Your goal is to **memorize the four logical steps of recursion** and the **three traversal positions**, not the Java syntax. This allows you to apply the same core ideas to any new tree problem.

Untitled

04 July 2022 21:54

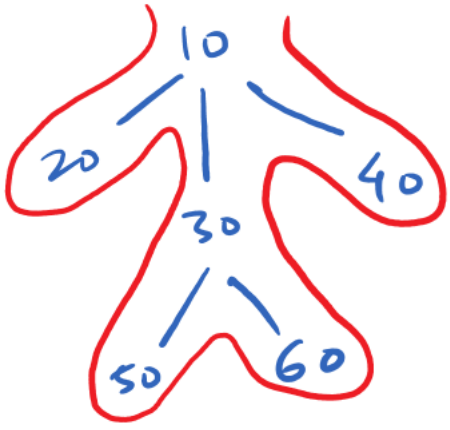
Preorder

Jab hum node ke left side se guzarte hai euler path me

Node ka left matlab : before going deep in recursion
Recursion me jate samay

Node hamesha children se pehle visit hota hai

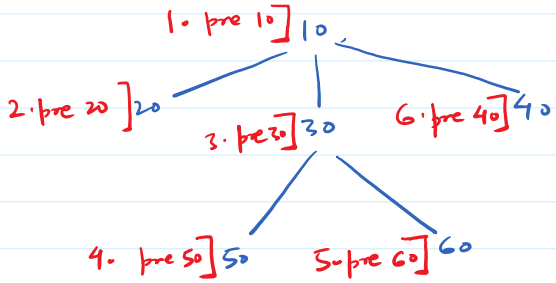
Root pehla banda hota hai



10
20
30
50
60
40

Pre Printing in Euler path

04 July 2022 21:41




```
static int index=1;
void prePrinting(Node* node)
{
    cout<<index++<<". Pre "<<node->data<<endl;
    for(Node* child: node->children){
        prePrinting(child);
    }
}
```

```
1. Pre 10
2. Pre 20
3. Pre 30
4. Pre 50
5. Pre 60
6. Pre 40
```

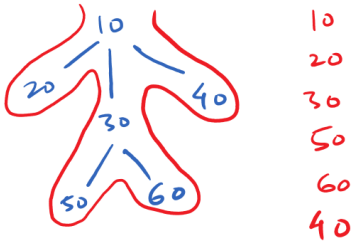
Preorder

Jab hum node ke left side se guzarte hai euler path me

Node ka left matlab : before going deep in recursion
Recursion me jate samay

Node hamesha children se pehle visit hota hai

Root pehla banda hota hai



Untitled

23 June 2022 19:49

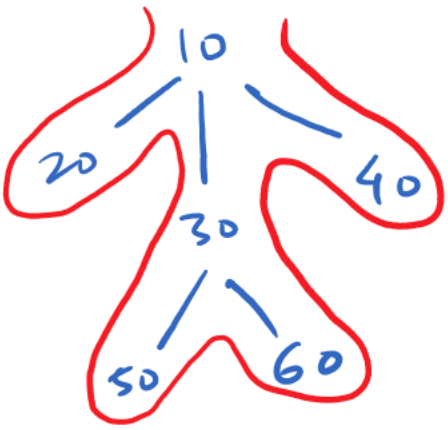
Post order

Jab hum euler path me node ke right side se guzarte hai

Node ka right matlab: recusion se bahar ate samay

Node hamesha children ke baad visit hota hai

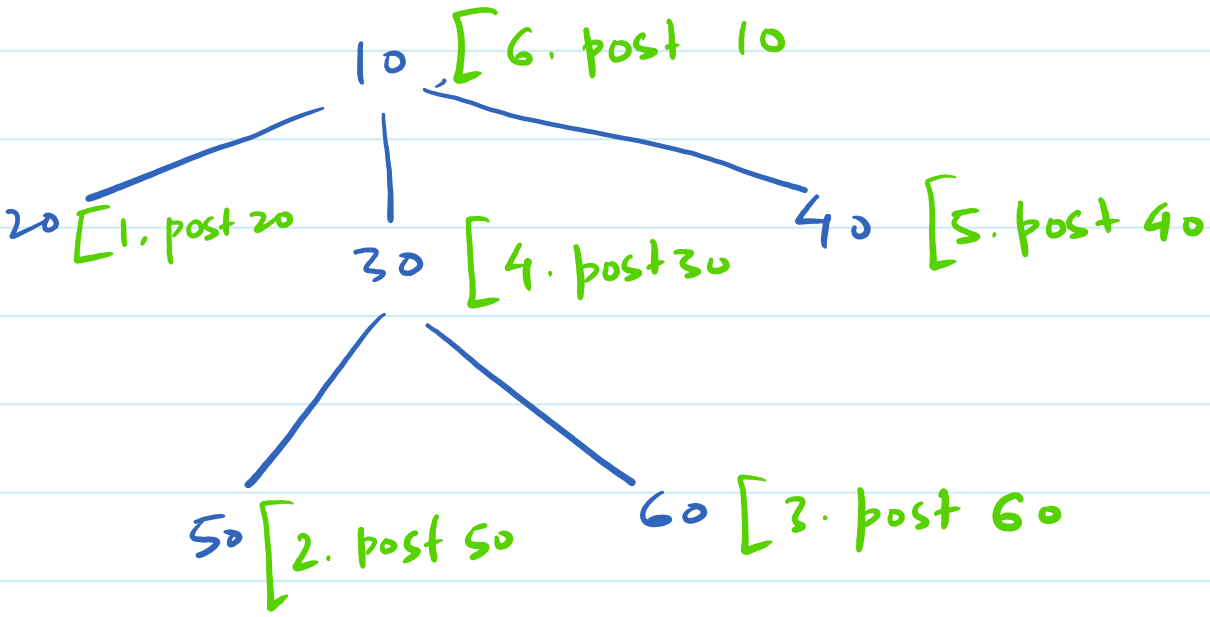
Root last banda hota hai



20
50
60
30
40
10

Post printing in euler path

04 July 2022 21:41

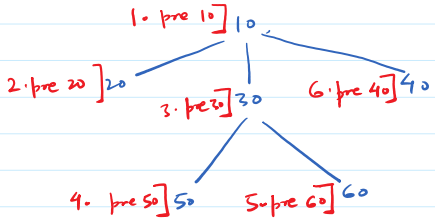



```
1. Post 20
2. Post 50
3. Post 60
4. Post 30
5. Post 40
6. Post 10
```

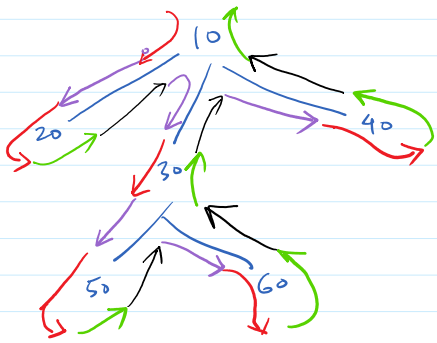
```
static int index=1;
void postPrinting(Node* node)
{
    for(Node* child: node->children){
        postPrinting(child);
    }

    cout<<index++<<". Post "<<node->data<<endl;
}
```

Generic tree traversal.

23 June 2022 19:45

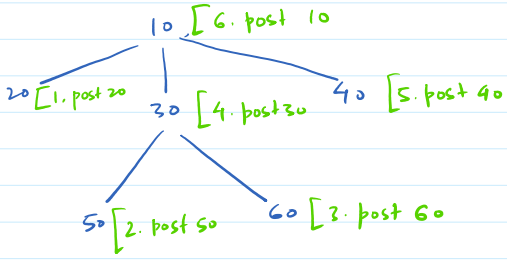


Basically looking at how to draw the euler path in case of this given tree



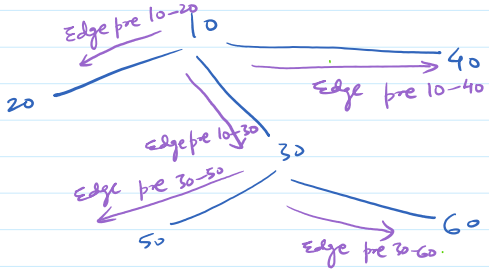

```
Node Pre 10
Edge pre 10- 20
Node Pre 20
Node Post 20
Edge post 10- 20
Edge pre 10- 30
Node Pre 30
Edge pre 30- 50
Node Pre 50
Node Post 50
Edge post 30- 50
Edge pre 30- 60
Node Pre 60
Node Post 60
Edge post 30- 60
Node Post 30
Edge post 10- 30
Edge pre 10- 40
Node Pre 40
Node Post 40
Edge post 10- 40
Node Post 10
```

Node pre printing

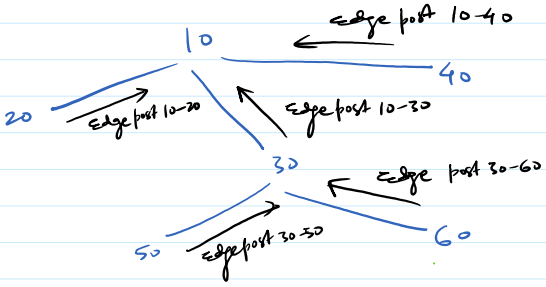



```
void display1(Node* node)
{
    cout<<"Node Pre "<< node->data<<endl;
    for(Node* child: node->children)
    {
        cout<<"Edge pre "<< node->data<<"- "<<child->data<<endl;
        display1(child);
        cout<<"Edge post "<< node->data<<"- "<<child->data<<endl;
    }
    cout<<"Node Post "<< node->data<<endl;
}
```

Node Post printing



Edge Pre printing



Edge Post printing.

Edge pre 10-20 tab print hoga jab hum edge pe honge 10 se 20
jane ke time pe.

Untitled

23 June 2022 19:54

Now let's see how the coding relation works with pre and post

Pre line recusion call se pehle likhte hai
Call se pehle likhte hai to recursion me jate hue chalta hai

Call ke baad likhte hai, to recursion se ate hue chalta hai,

```
void display1(Node* node)
{
    // Euler's left, on the way deep in recursion, node's pre area
    cout<<"Node Pre "<< node->data<<endl;
    for(Node* child: node->children)
    {
        display1(child);
    }
    cout<<"Node Post "<< node->data<<endl;
    // Euler's right, on the way out of recursion, node's post area
}
```

There is nothing like edge pre or edge post area
But for the sake of understanding lets see and analyse

```
void display1(Node* node)
{
    // Euler's left, on the way deep in recursion, node's pre area
    cout<<"Node Pre "<< node->data<<endl;
    for(Node* child: node->children)
    {
        //Edge pre
        cout<< "Edge pre "<< node->data <<"- "<<child->data<<endl;
        display1(child);

        //Edge post
        cout<< "Edge post "<< node->data <<"- "<<child->data<<endl;
    }
    cout<<"Node Post "<< node->data<<endl;
    // Euler's right, on the way out of recursion, node's post area
}
```

Do dry run this thing

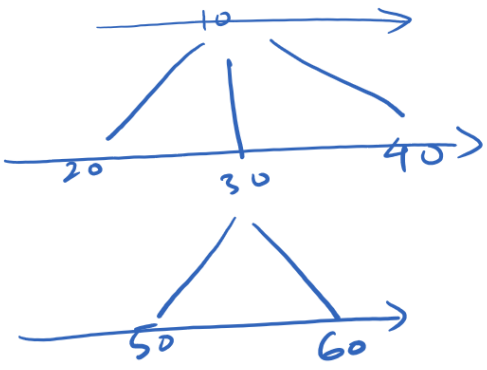
```
void display1(Node* node)
{
    cout<<"Node Pre "<< node->data<<endl;
    for(Node* child: node->children)
    {
        cout<<"Edge pre "<< node->data<<"- "<<child->data<<endl;
        display1(child);
        cout<<"Edge post "<< node->data<<"- "<<child->data<<endl;
    }
    cout<<"Node Post "<< node->data<<endl;
}
```

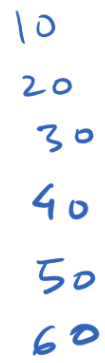
```
void display1(Node* node)
{
    // Euler's left, on the way deep in recursion,
node's pre area
    cout<<"Node Pre "<< node->data<<endl;
    for(Node* child: node->children)
    {
        //Edge pre
        cout<< "Edge pre "<< node->data <<"-
"<<child->data<<endl;
        display1(child);
        //Edge post
```

```
        cout<< "Edge post "<< node->data <<"-
"<<child->data<<endl;
    }
    cout<<"Node Post "<< node->data<<endl;
    // Euler's right, on the way out of recursion,
node's post area
}
```

Level order of generic tree

23 June 2022 19:57

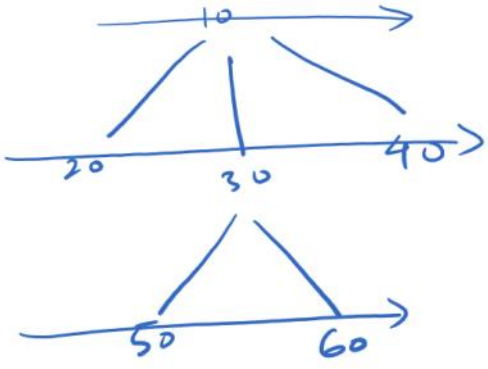




Level by level print karenge

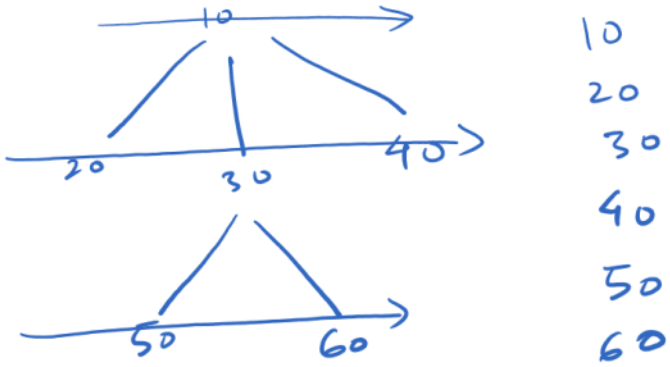
We will use queue data structure and it's a very common algorithm

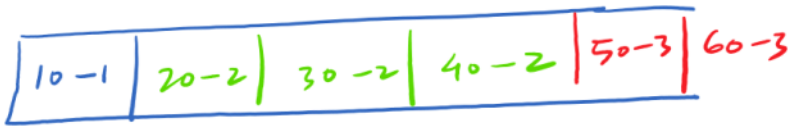
Remove, print and add (RPA)



RPA

Let's see with the levels





Sare first level wale process honge, fir second level honge, fir third level
Queue ka behaviour hi aisa hota hai, ki unke agle level ke childrens baad me dalenge

Remember: **Queue is for Level Order (BFS)!**

💡 Key Steps to Remember (The BFS Mantra)

The core idea of Level Order Traversal is to process nodes **level by level** using a **Queue**.

1. **Initialize:** Start with the **Queue** and add the **root node**.
2. **Loop:** While the Queue is **not empty**...
3. **Process: Dequeue** the next node (that's the one at the front). **Print/Process** it.
4. **Enlist: Enqueue all** of its **children**.

❓ Queue Operations in Java

Action	Code	Description
Declaration (using `ArrayDeque`)	`Queue<Node> q = new ArrayDeque<>();`	`Queue` is the interface; `ArrayDeque` is a common concrete class implementing it.
Adding an element	`q.add(node);`	Inserts the element at the **end** (rear) of the Queue.
Getting/Removing the front	`Node front = q.remove();`	Retrieves and **removes** the element at the **front**. Throws an exception if the queue is empty.
Checking the size	`q.size()`	Returns the number of elements currently in the Queue.

Step-by-Step Algorithm (Corrected)

1. **Declare the Queue:** `Queue<Node> q = new ArrayDeque<>();`
2. **Add the first node (root):** `q.add(node);`
3. **Run Loop:** `while(q.size() > 0)` or `while(!q.isEmpty())`
4. **Get the front element and process it:**
   - `Node front = q.remove();`
   - `System.out.println(front.data + " ");`
5. **Add all the children of that node:**
   - `for(Node child : **front**.children){` (***This is the critical fix!***)
   - `q.add(child);`
   - `}`

```java
public void levelOrder(Node node) {
    Queue<Node> q = new ArrayDeque<>();
    //or Queue<Node> q = new LinkedList<>();

    // 2. Correct Initial Push: Use add()
    q.add(node);

    while (q.size()>0) {

        // 4. Correct Dequeue: Use remove()
        Node front = q.remove();

        System.out.print(front.data + " ");

        // 5. Iterate over the children of the 'front' node
        for (Node child : front.children) {
            // Add the child to the queue
            q.add(child);
        }
    }
    // Print a newline or separator after traversal is complete
    System.out.println(x: ".");
}
```

```
10
20
30
40
50
60
```

Kindly do the dry run and meditate hard

Pehle level ka banda nikla, to dusre level ke dal gaye
Dusre level ka nikla to usne teesre level ke bande dale

Lekin sare teesre level ke bande, dusre level ke baad hi honge

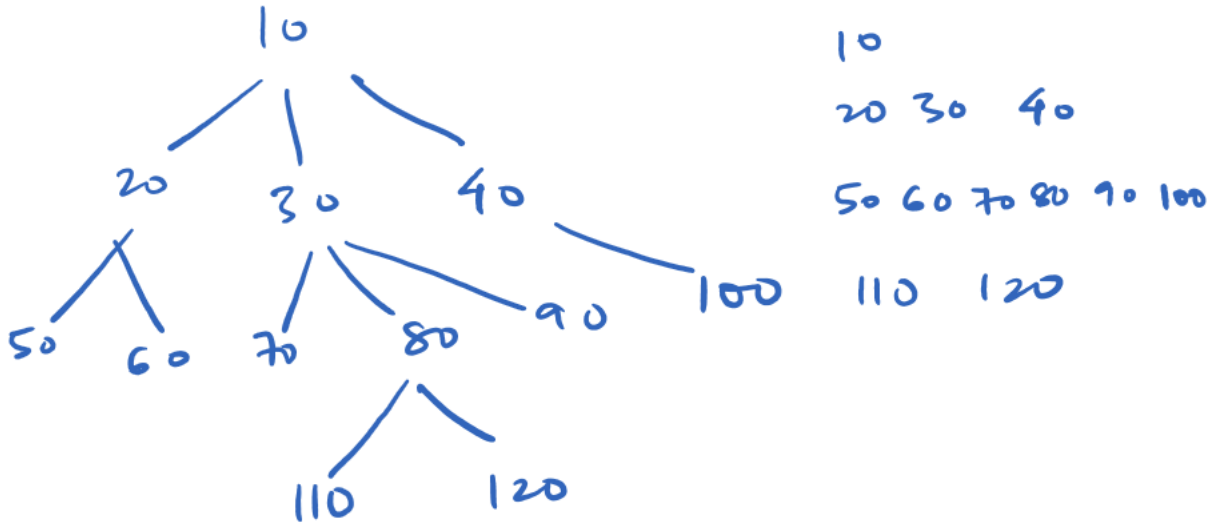
Add last me hota hai

Ache se meditate karke soche ki ye algorithm kaise chala

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeLevelOrder.java

Level order linewise

23 June 2022 20:02

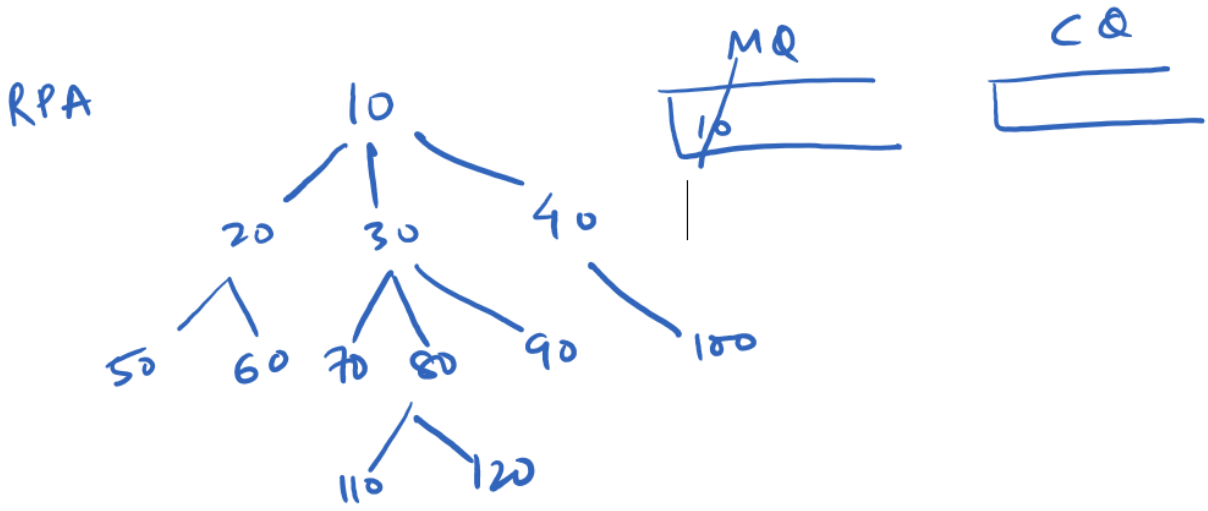


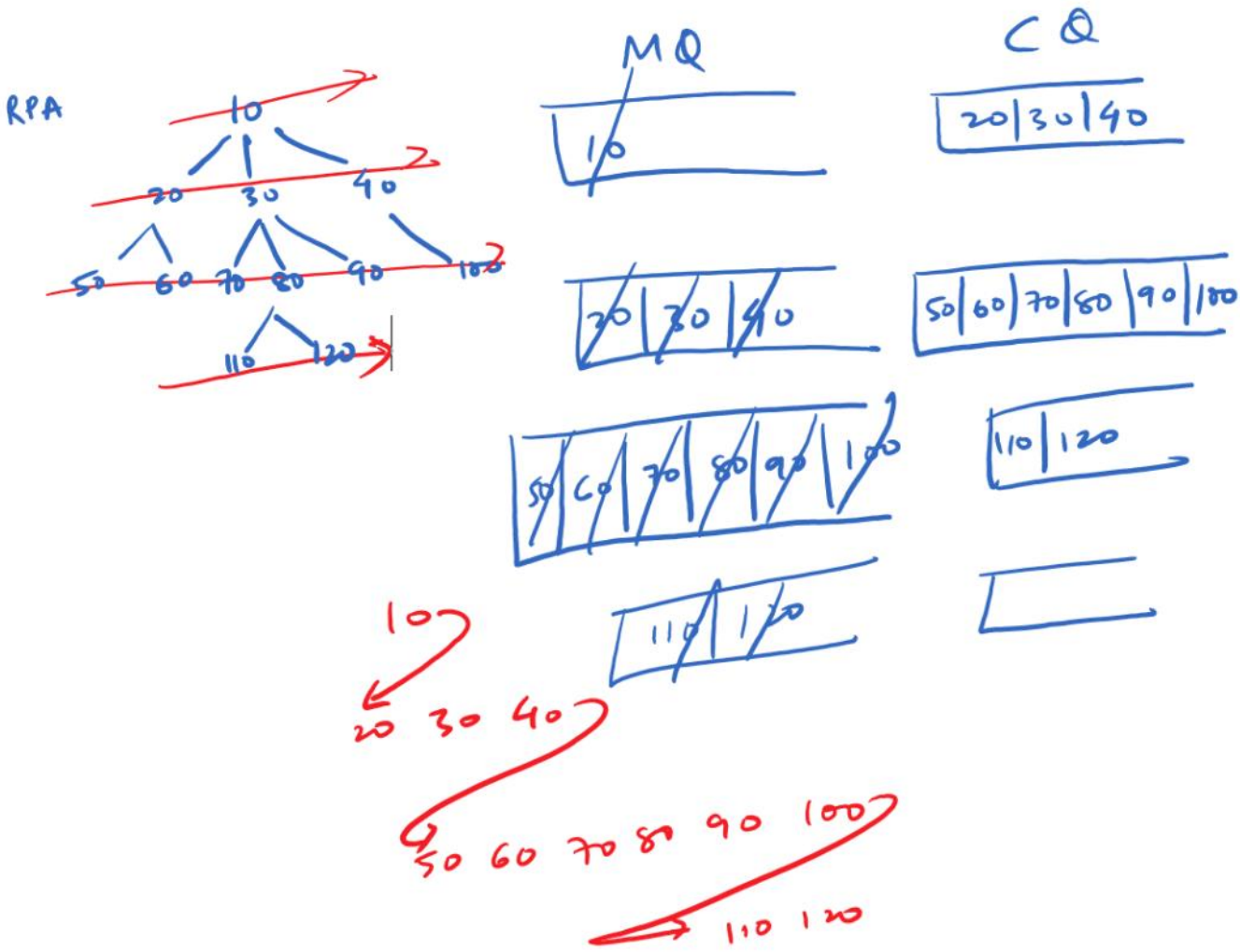
Har ek level alag alag line me print hoga aur
Sare level left se right me visit honge

Untitled

23 June 2022 20:03

We will take two queue
Main queue
Child queue





Don't see the code, just write your own by understanding the algo

Level order linewise algo

24 November 2025 21:48

Level Order Traversal (Line Wise) Algorithm

The method works by ensuring that at any point, the `mq` holds all the nodes of the **current level** being processed.

1. Initialization

- `Queue<Node> mq = new ArrayDeque<>();` : **Main Queue (mq)**, holds nodes of the current level.
- `Queue<Node> cq = new ArrayDeque<>();` : **Child Queue (cq)**, holds children of the current level (i.e., the nodes of the next level).
- `mq.add(node);` : Start by adding the **root node** to the main queue.

2. The Traversal Loop (`while(mq.size() > 0)`)

The loop runs as long as there are nodes to process in the current level (`mq`).

- **Process Current Node:**
  - `Node front = mq.remove();` : Dequeue and get the node at the front of the current level.
  - `System.out.println(front.data + " ");` : Process/print the node's data.
- **Enqueue Children:**
  - `for(Node child: front.children){ cq.add(child); }` : **All** children of the currently processed node (`front`) are added to the **child queue (`cq`)**.

3. Level Completion and Swap (The Key Logic)

This is the crucial step that separates levels and prints the new line:

- `if(mq.size() == 0){ ... }` : This condition is met **only when the main queue becomes empty**, meaning **all nodes of the current level have been processed**.
  - `mq = cq;` : The `mq` (now empty) is **swapped/replaced** with the `cq` (which holds all the nodes of the *next* level).
  - `cq = new ArrayDeque<>();` : A **new, empty** child queue is created to collect the nodes for the *subsequent* level.
  - `System.out.println();` : This prints a **new line**, ensuring the output for the next level starts on a fresh line, achieving the "Level Wise" output format.

Untitled

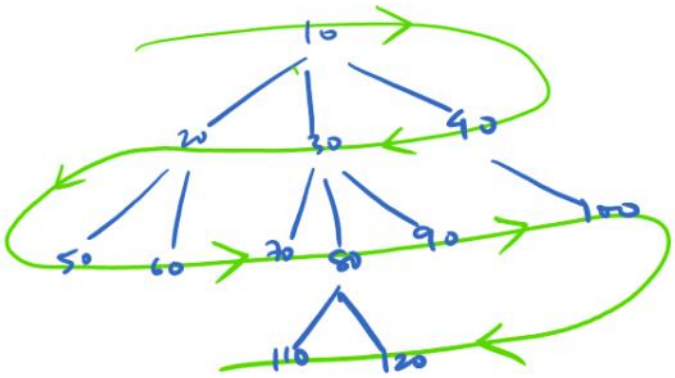
23 June 2022 20:05

```java
public void levelOrderLW(Node node) {
    // 1. Declare two queues: mq for current level, cq for next level's children
    Queue<Node> mq = new ArrayDeque<>();
    Queue<Node> cq = new ArrayDeque<>();


    mq.add(node); // Add the root node to start

    // Loop continues as long as there are levels to process
    while(!mq.isEmpty()){

        // A. Dequeue the node from the current level (mq)
        Node front = mq.remove();

        // B. Print the data. Use print() to keep level nodes on the same line.
        System.out.print(front.data + " ");

        // C. Enqueue all children of the processed node into the child queue (cq)
        for(Node child : front.children){
            cq.add(child);
        }

        // D. Level Complete Check & Swap
        if(mq.isEmpty()){
            // The current level (mq) is fully processed.

            mq = cq; // Swap: The children (cq) become the new current level (mq)
            cq = new ArrayDeque<>(); // Reset cq to collect the *next* level's children

            System.out.println(); // Print newline to separate the levels in the output
        }
    }
}
```

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/genericTreeLevelOrderLW.java

Level order linewise - zig zag

23 June 2022 20:05



10

40 30 20

50 60 70 80 90 100

120 110

Bas is baar zig zag hoga

Untitled

23 June 2022 20:06

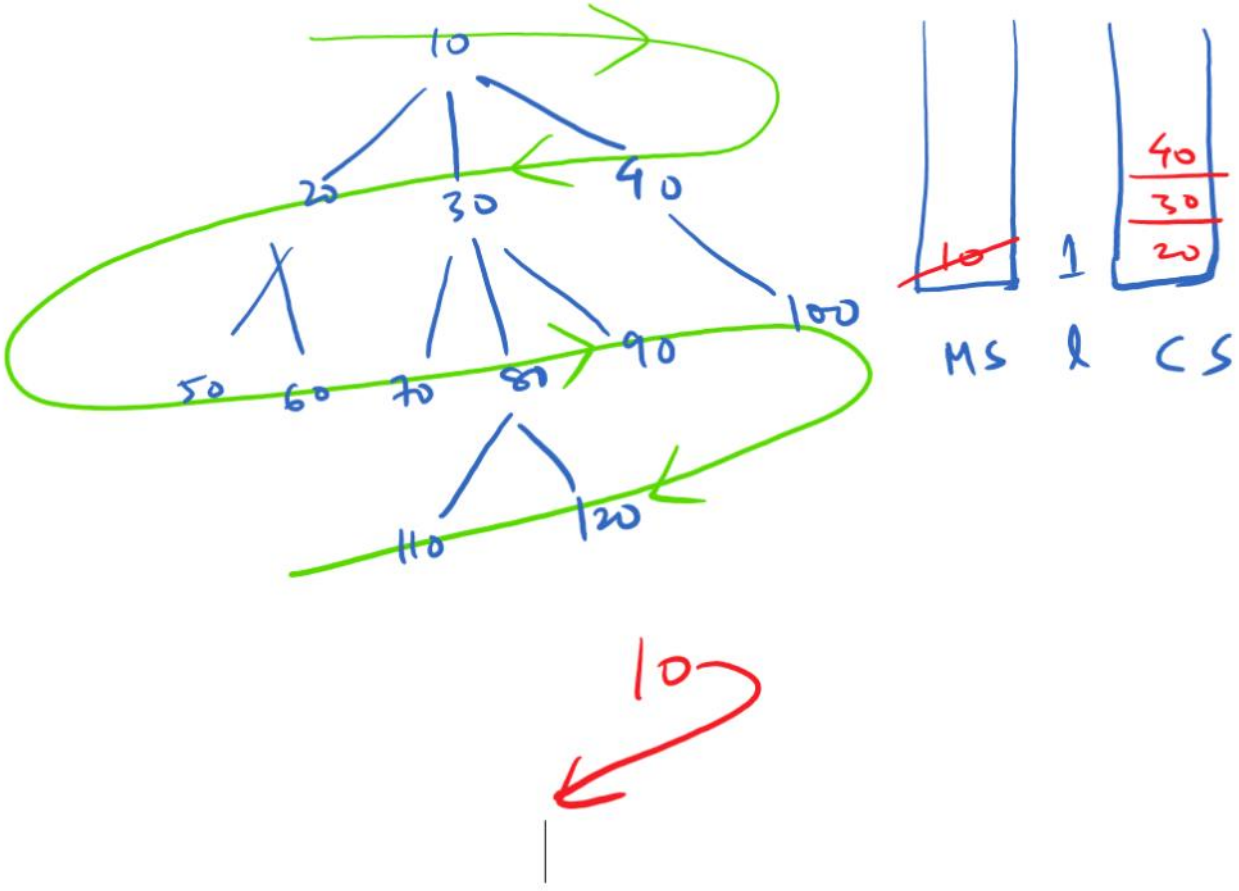
Understand What and how ?

Yaha queue ki jagah hum stack use karenge

Main stack, child stack
Or a flag called as level (l)

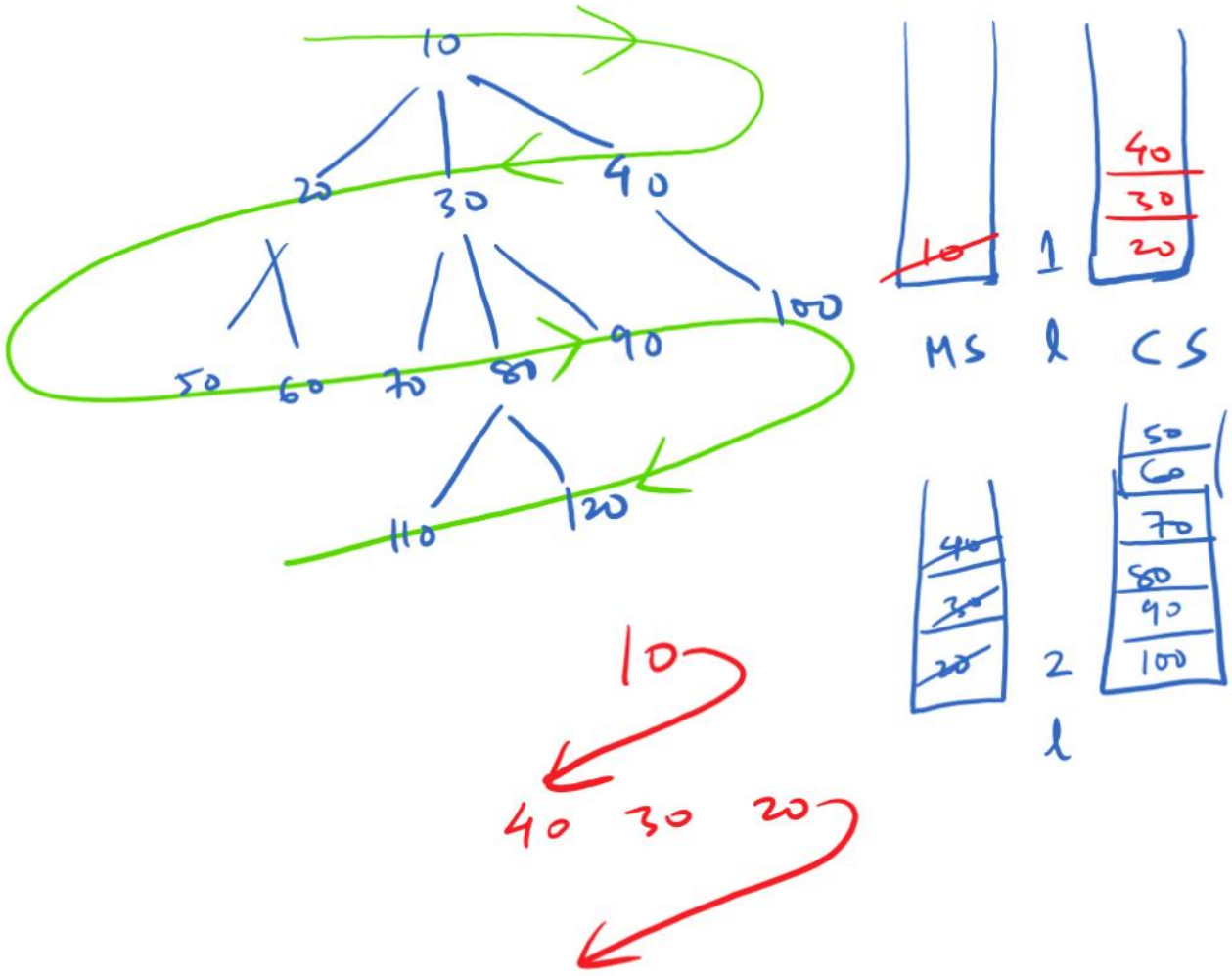
Remove print add

Shuruaat me hamara level 1 hai



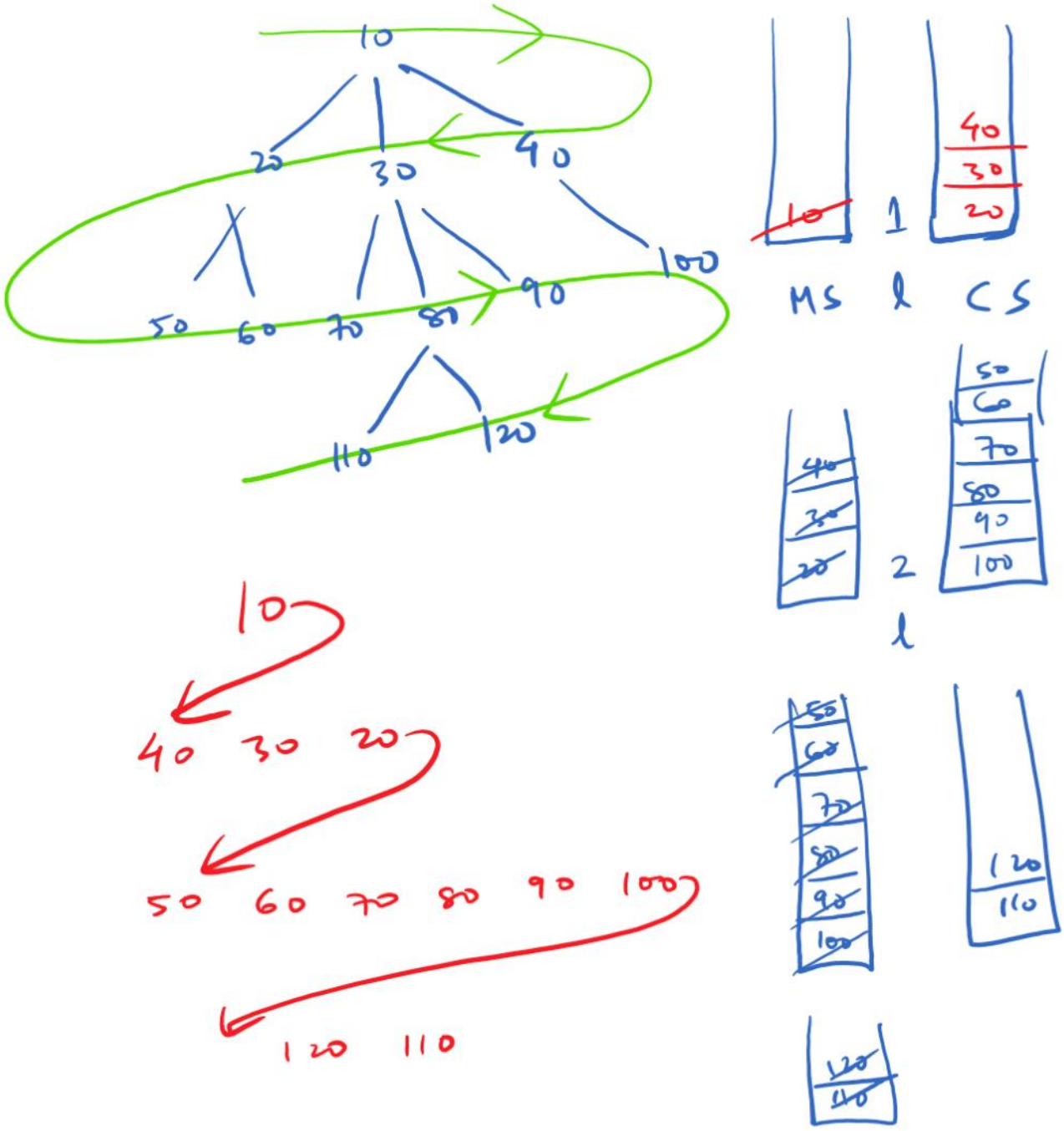
10 ke case me hum left se right ja rhe hai, to CS me hum children left se right hi add karenge

Level badal ke 2 kar de



40, 30, 20 me hum right to left ja rhe hai, to children bhi CS me right se left hi add karenge.

Aru level badal ke 3 kar dijiye



Note : level keeps on changing (increasing)

[only change is that based on the value of level , we are pushing the children from either left to right or right to left in the child stack.]

Code niche bina dekhe likhne ki kosis karo

```
void levelOrderLinewise(Node* node)
{
    //1. create a main stack and child stack to store the nodes
    stack <Node* > ms;
    stack <Node* > cs;

    //level flag to add the children
    int level=1;
    //2. push the first node in main stack
    ms.push(node);

    //3. Run the loop till the stack is not empty
    while(ms.size() > 0)
    {
        //take the top node
        Node* top = ms.top();
        // 1. REMOVE
        ms.pop();

        // 2. PRINT the data of top node
        cout<<top->data <<" ";

        // 3. ADD the children of the front node
        //based on value of level

        if(level%2 !=0){
            //start to end
            for(int i=0; i< top->children.size(); i++)
            {
                Node* child = top->children[i];
                cs.push(child);
            }
        }
        else{
            //end to start
            for(int i=top->children.size() -1; i >=0; i--)
            {
                Node* child = top->children[i];
                cs.push(child);
            }
        }

        if(ms.size() == 0)
        {
            ms=cs;

            while(!cs.empty())
            {
                cs.pop();
            }
            level++;
            cout<<endl;
        }
    }
}
```

```
void levelOrderLinewise(Node* node)
{
    //1. create a main stack and child stack to store the nodes
    stack <Node* > ms;
    stack <Node* > cs;

    //level flag to add the children
    int level=1;
    //2. push the first node in main stack
    ms.push(node);

    //3. Run the loop till the stack is not empty
    while(ms.size() > 0)
    {
        //take the top node
        Node* top = ms.top();
        // 1. REMOVE
        ms.pop();

        // 2. PRINT the data of top node
        cout<<top->data <<" ";

        // 3. ADD the children of the front node
        //based on value of level

        if(level%2 !=0){
            //start to end
            for(int i=0; i< top->children.size(); i++)
            {
                Node* child = top->children[i];
                cs.push(child);
            }
        }
        else{
            //end to start
            for(int i=top->children.size() -1; i >=0; i--)
            {
                Node* child = top->children[i];
                cs.push(child);
            }
        }

        if(ms.size() == 0)
        {
            swap(ms, cs);

            level++;
            cout<<endl;
        }
    }
}
```

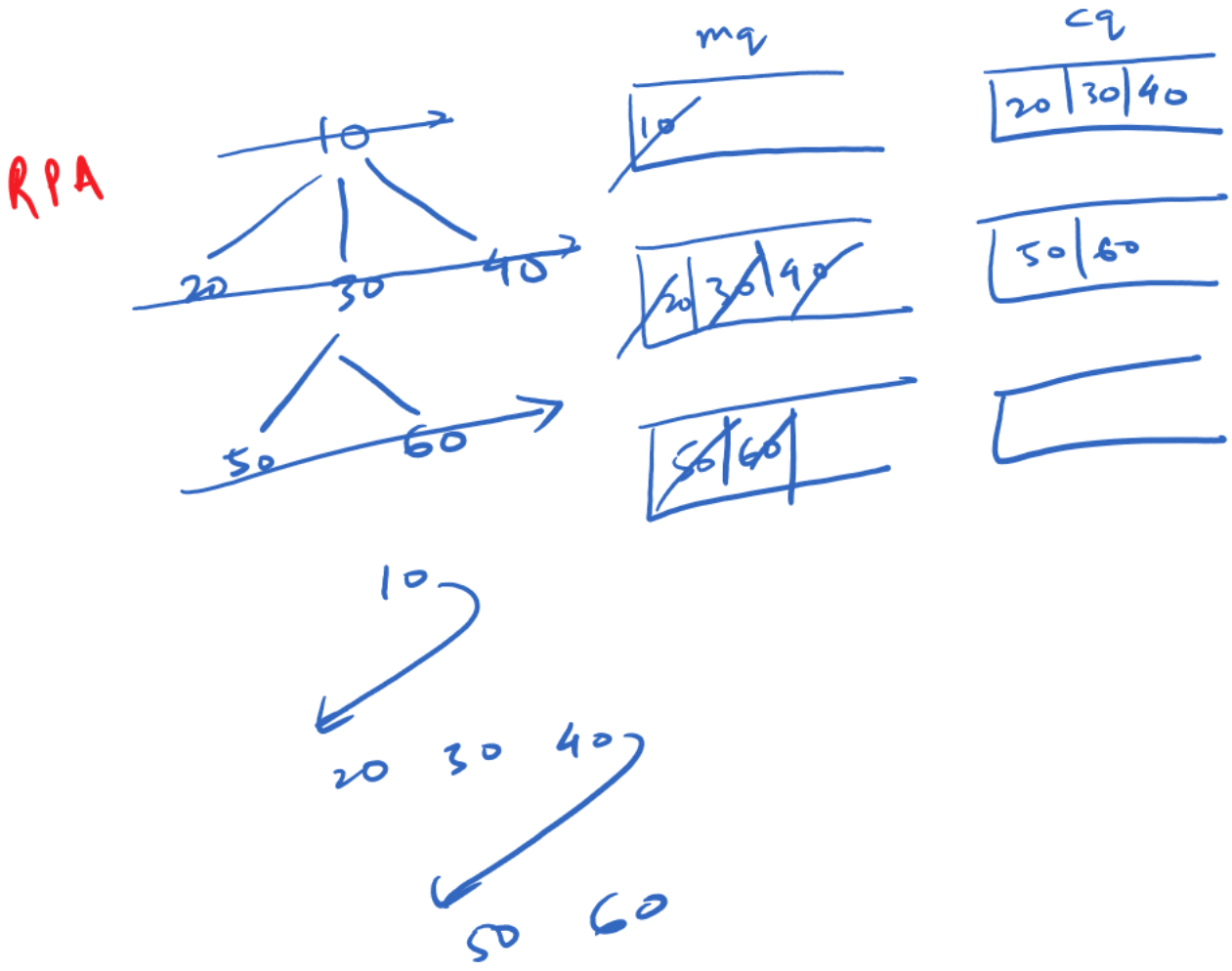
```
void levelOrderLinewise(Node* node)
{
    //1. create a main stack and child stack to store
the nodes
    stack <Node* > ms;
```

```
    stack <Node* > cs;
    //level flag to add the children
    int level=1;
    //2. push the first node in main stack
    ms.push(node);
    //3. Run the loop till the stack is not empty
    while(ms.size() > 0)
    {
        //take the top node
        Node* top = ms.top();
        // 1. REMOVE
        ms.pop();
        // 2. PRINT the data of top node
        cout<<top->data <<" ";
        // 3. ADD the children of the front node
        //based on value of level

        if(level%2 !=0){
            //start to end
            for(int i=0; i< top->children.size(); i++)
            {
                Node* child = top->children[i];
                cs.push(child);
            }
        }
        else{
            //end to start
            for(int i=top->children.size() -1; i >=0;
i--)
            {
                Node* child = top->children[i];
                cs.push(child);
            }
        }
        if(ms.size() == 0)
        {
            swap(ms, cs);

            level++;
            cout<<endl;
        }
    }
}
```

```
[Running] cd "c:\Users\Ankit\Documen
10
40 30 20
50 60 70 80 90 100
120 110
```

Level order alternate approaches

24 June 2022 09:20



First approach is using two queue, main queue and child queue.

```
void levelOrder(Node* node)
{
    queue<Node*> mq, cq;
    mq.push(node);

    while(mq.size()>0)
    {
        Node* front=mq.front();
        mq.pop();
        cout<< front->data<<" ";

        for(Node* child: front->children)
        {
            cq.push(child);
        }
        if(mq.size()==0)
        {
            swap(mq, cq);
        }
    }
}
```

We have seen this method already, but there are other ways to do the same.

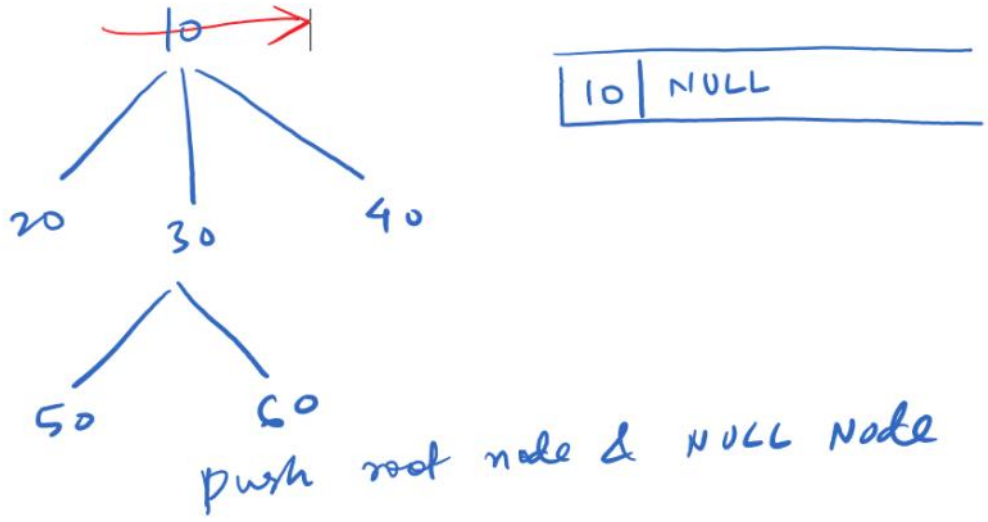
Delimiter approach

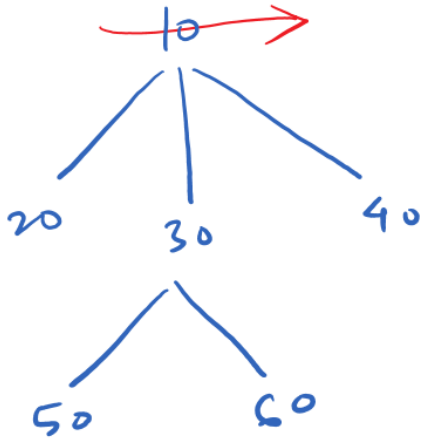
24 June 2022 09:21

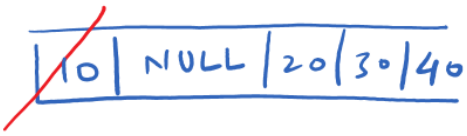
You don't require two queues,

This approach is with the delimiter approach

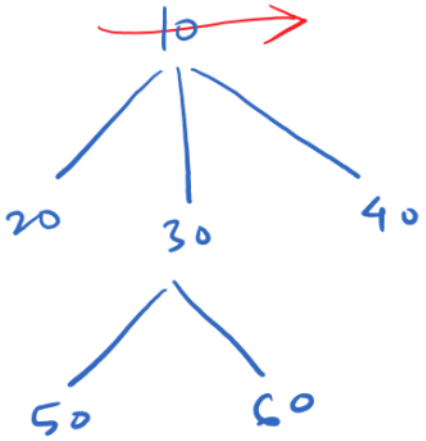
You can do the same using the null(n)



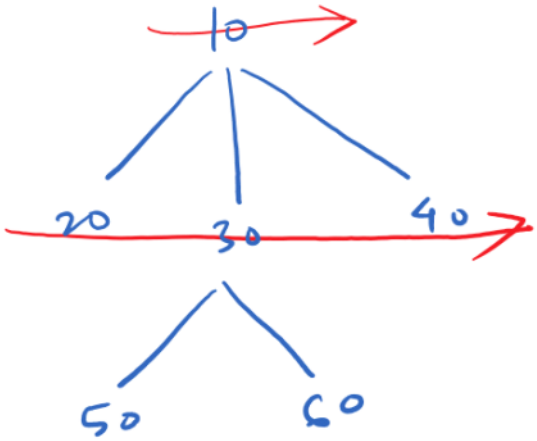




10 removed, & as it is not NULL node, so add its children

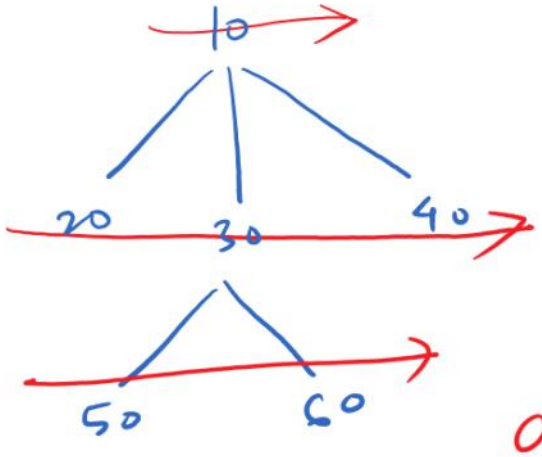


Remove NULL, change line &
Add one more NULL as
size of Queue > 0



10 | NULL | 20 | 30 | 40 | NULL | 50 | 60 | NULL

Remove NULL, change line As size of Queue > 0, so Add NULL



10 | NULL | 20 | 30 | 40 | NULL | 50 | 60 | NULL

For Last NULL, change line but As Queue size = 0, SO Don't Add NULL in only this last case

Add null with the node

If you remove null, and if queue is not empty, then add null in the end again, i.e. push null again. And change line

```
void levelOrder(Node* node)
{
    //1. create just one queue
    queue<Node*> mq;

    //2. push the node and the NULL node
    mq.push(node);
    mq.push(NULL);

    //3. Run the loop till the size of queue > 0
    while(mq.size()>0)
    {
        //4. REMOVE
        Node* rem= mq.front();
        mq.pop();

        //5. If the front node is NULL, then do the change line
        if(rem==NULL)
        {
            //if the queue size is greater than 0, add one more NULL node
            if(mq.size()>0)
            {
                mq.push(NULL);
                cout<<endl;
            }
        }
        else
        {
            // PRINT the data
            cout<< rem->data<<" ";

            // ADD the childrens
            for(Node* child: rem->children)
                mq.push(child);
        }
    }
}
```

Did dry run of the code

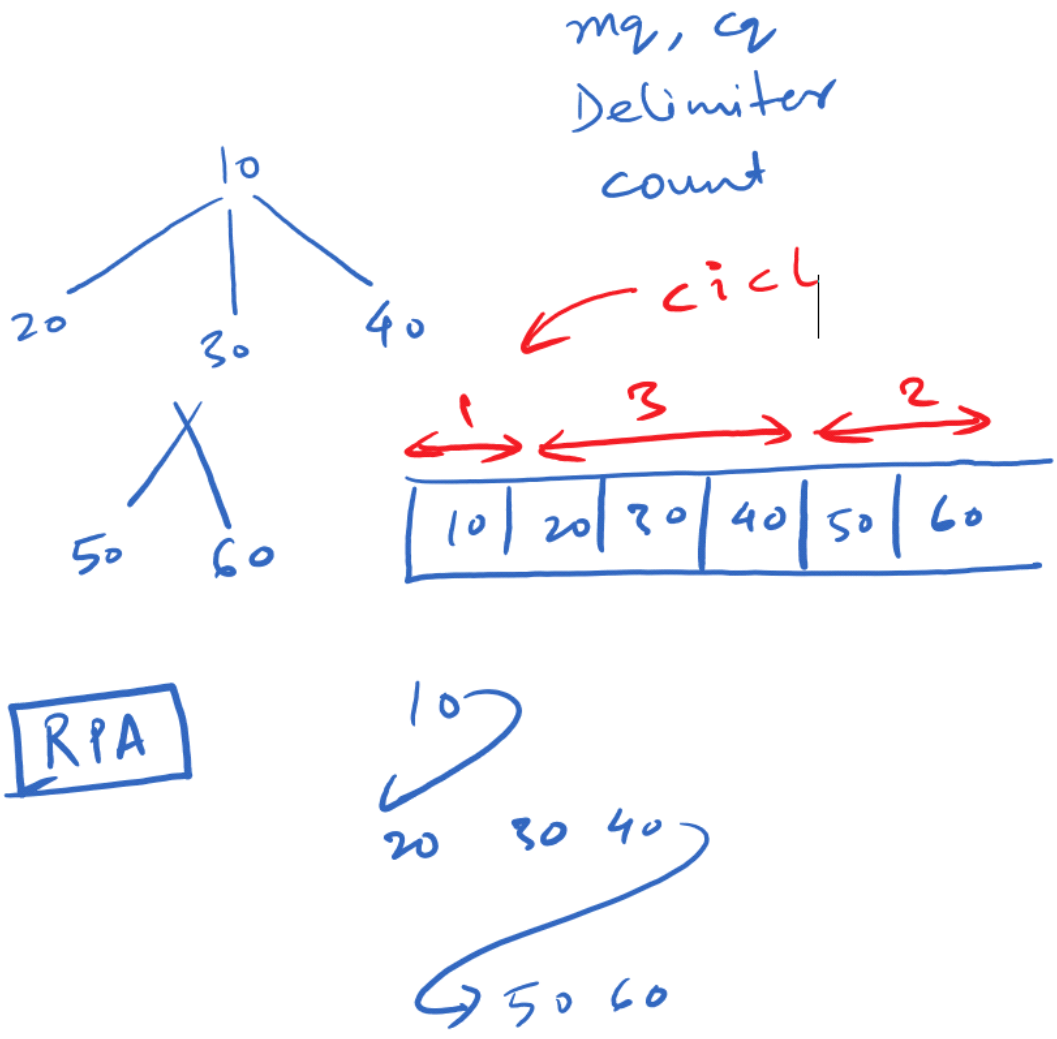
```
    public void levelOrderLW_NULL(Node node){
            // Check for null root/node before
attempting to add it to the ArrayDeque.
            if (node == null) {
                return;
            }
            // Initialize queue and add the root node.
            Queue<Node> q = new ArrayDeque<>();
            q.add(node);
            // Add null as a level separator after the
root.
            q.add(null);
            while(q.size() > 0){
                node = q.remove();

                // If the removed element is a node,
print it and enqueue its children.
                if(node != null){
                    System.out.print(node.data + " ");
                    for(Node child: node.children){
                        q.add(child);
                    }
                }
                // If the removed element is null, it
marks the end of a level.
                else {
                    // Check if the queue is not empty
(i.e., there are nodes for the next level).
                    if(q.size() > 0){
                        System.out.println(); // Start
a new line for the next level's output.
                        q.add(null); // Add a new null
marker for the end of the next level.
                    }
                }
            }
        }
```

Count approach

24 June 2022 09:29

Third approach using count variable.

Pehle queue ki size pata kar lo, aur utne baar loop chalao aur Remove print add karo



Itne approaches se mai tumhare tarqas bhar rha hu, kya pata kab kya kaam aa jaye
More weapon in your armoury.

05 Anjney Singh 1 year ago
Tarkas m teere dene k liyee apko sat sat naman. Hm jang jrur jeetenge

Write code by yourself.

Cicl: children in current level.

```
void levelOrder(Node* node)
{
    //1. create just one queue
    queue<Node*> mq;

    //2. push the node
    mq.push(node);

    //3. Run the loop till the size of queue > 0
    while(mq.size()>0)
    {
        int cicl = mq.size(); //cicl: children in current level

        for(int i=0; i< cicl; i++)
        {
            //4. REMOVE
            Node* rem= mq.front();
            mq.pop();

            //5. PRINT
            cout<< rem->data<< " ";

            //6. ADD
            for(Node* child: rem->children)
            {
                mq.push(child);
            }
        }
        cout<<endl;
    }
}
```

Kindly do the dry run

```
void levelOrder(Node* node)
{
    //1. create just one queue
    queue<Node*> mq;
    //2. push the node
```

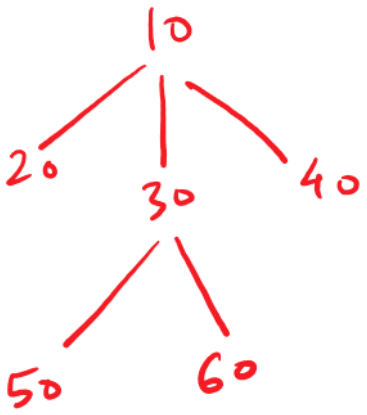
```
    mq.push(node);
    //3. Run the loop till the size of queue > 0
    while(mq.size()>0)
    {
        int cicl = mq.size(); //cicl: children in current level
        for(int i=0; i< cicl; i++)
        {
            //4. REMOVE
            Node* rem= mq.front();
            mq.pop();
            //5. PRINT
            cout<< rem->data<< " ";
            //6. ADD
            for(Node* child: rem->children)
            {
                mq.push(child);
            }
        }
        cout<<endl;
    }
}
```

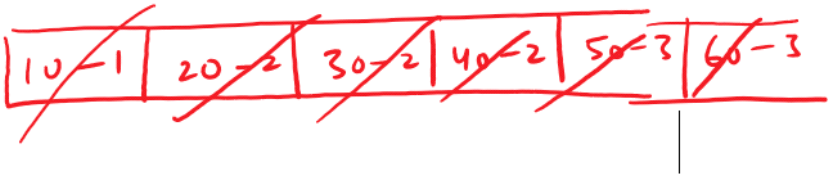
Pair class approach

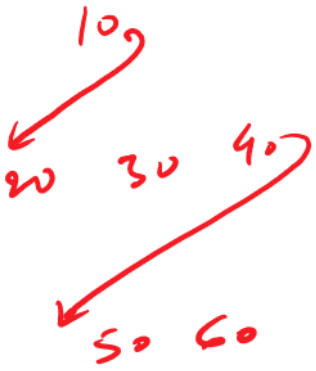
24 June 2022 12:54

Using pair class.

Is baar queue nodes ki nahi bani hai, pair ki bani hai








```
class Pair{
    public:
    Node* node;
    int level;
    Pair(Node* node, int level)
    {
        this->node= node;
        this->level= level;
    }
};
```

Write code by yourself

```
void levelOrder(Node* node)
{
    //1. create just one queue
    queue<Pair> q;
    // push the root with level 1
    q.push(Pair(node, 1));
    // create a variable level to keep track of level
    int level=1;

    // loop till the queue is not empty
    while(q.size()>0)
    {
        // REMOVE
        Pair rem= q.front();
        q.pop();

        //UPDATE level
        if(rem.level > level)
        {
            cout<<endl;
            level= rem.level;
        }
        //PRINT
        cout<<rem.node-> data<< " ";

        //ADD with increased level
        for(Node* child: rem.node-> children)
            q.push(Pair(child, rem.level+1));

    }
}
```

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    vector<Node*> children;
};
Node* newNode(int val)
```

```
{
    Node* node = new Node;
    node->data= val;
    return node;
}
class Pair{
    public:
    Node* node;
    int level;
    Pair(Node* node, int level)
    {
        this->node= node;
        this->level= level;
    }
};

Node* constructor(vector<int> &arr, int n)
{
    //1. create a root pointer and point it to NULL
    Node* root=NULL;
    //2. create a stack for the operation
    stack< Node*> st;
    //3. run the loop on all the elements of array
    for(int i=0; i<n; i++)
    {
        //4. if you encounter number other than -1
        if(arr[i]!= -1)
        {
            //4.11 Create a new node and fill the
value
            Node* t= newNode(arr[i]);
            //5.1 if stack empty, then make it root
            if(st.empty())
            {
                root=t;
            }
            else
            {
                //5.2 If not empty, then add to
childrens of node on top of stack
                st.top()->children.push_back(t);
            }
            //4.12 Push the node in the stack
            st.push(t);
        }
```

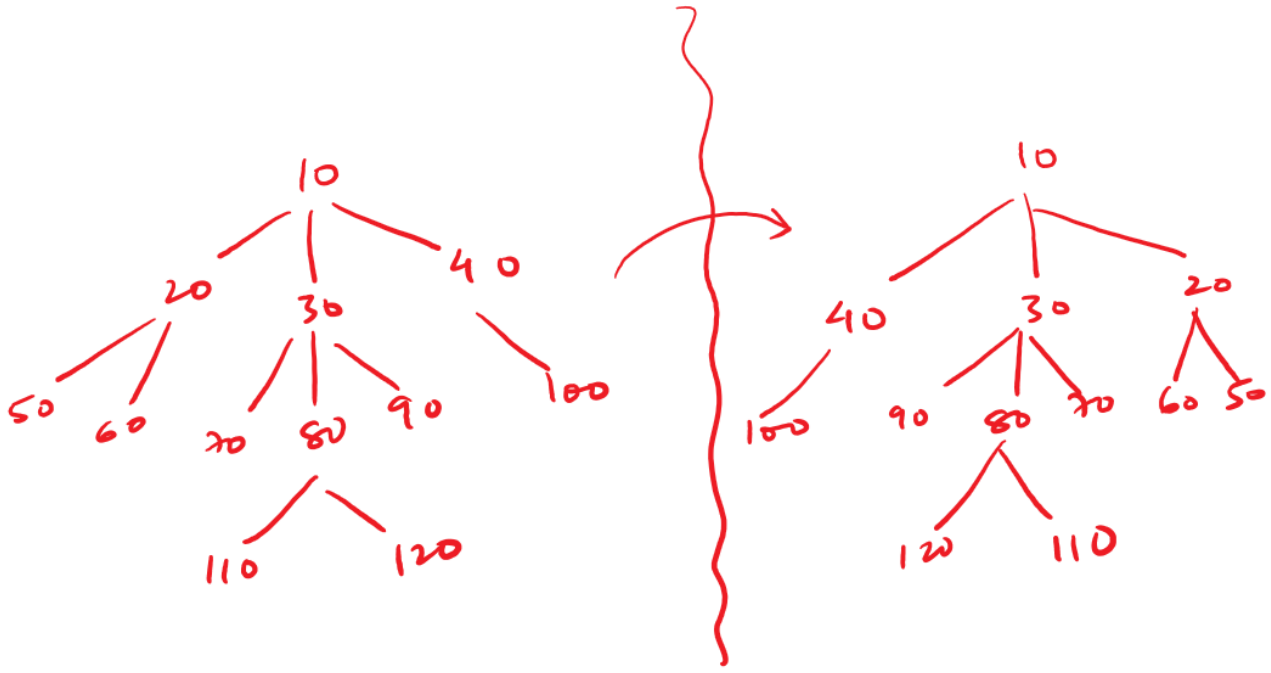
```
        else
        {
            //4.2 if you encounter -1, then pop from
stack
            st.pop();
        }
    }
    //return the root value
    return root;
}
void levelOrder(Node* node)
{
    //1. create just one queue
    queue<Pair> q;
    // push the root with level 1
    q.push(Pair(node, 1));
    // create a variable level to keep track of level
    int level=1;
    // loop till the queue is not empty
    while(q.size()>0)
    {
        // REMOVE
        Pair rem= q.front();
        q.pop();
        //UPDATE level
        if(rem.level > level)
        {
            cout<<endl;
            level= rem.level;
        }
        //PRINT
        cout<<rem.node-> data<< " ";
        //ADD with increased level
        for(Node* child: rem.node-> children)
            q.push(Pair(child, rem.level+1));
    }
}

int main()
{
    vector<int> arr;
    arr.assign({10, 20, 50, -1, 60, -1, -1, 30,
70, -1, 80, 110, -1, 120, -1, -1, 90, -1, -1, 40,
100, -1, -1, -1});
    int n= arr.size();
```

```
    Node* root=NULL;
    root=constructor(arr,n);
    levelOrder(root);
    return 0;
}
```

Mirror of a generic tree.

24 June 2022 15:58

Tree transformation

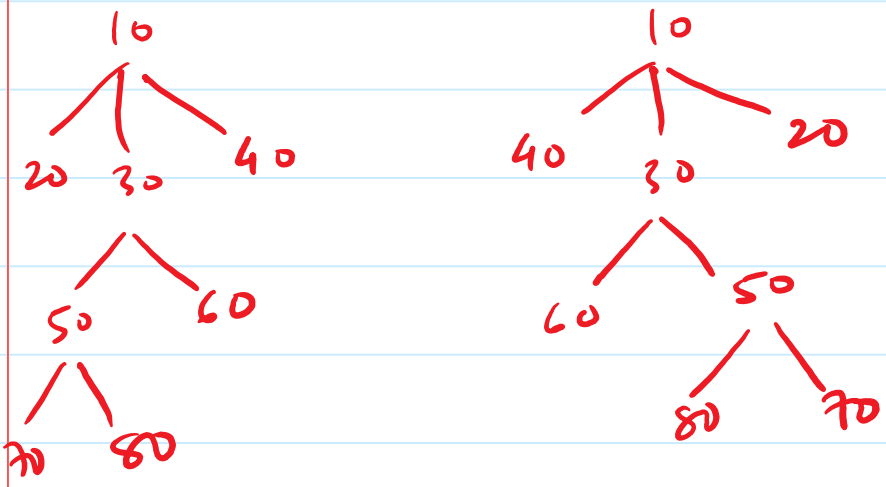


We are actually reversing the children at each level.

Untitled

05 July 2022 17:10

Reversing the children array list.



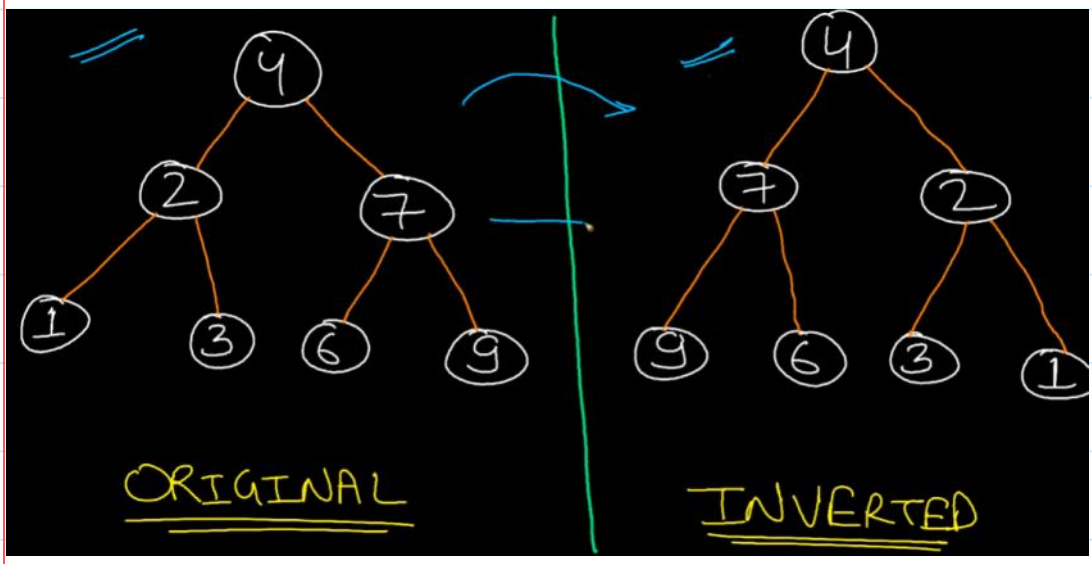
Faith: My children has already got their children's reversed

All I need to do is reverse my children as a root node and my mirror tree will be ready.

```java
public static void mirror(Node node){

    for(Node child: node.children){
        mirror(child);
    }
    //Collections.reverse(node.children);
    int left = 0;
    int right = node.children.size()-1;

    while(left < right){
        Node temp;
        temp = node.children.get(left);
        node.children.set(left, node.children.get(right)
        node.children.set(right, temp);
        left++;
        right--;
    }
}
```

Invert binary tree

24 June 2022 16:41

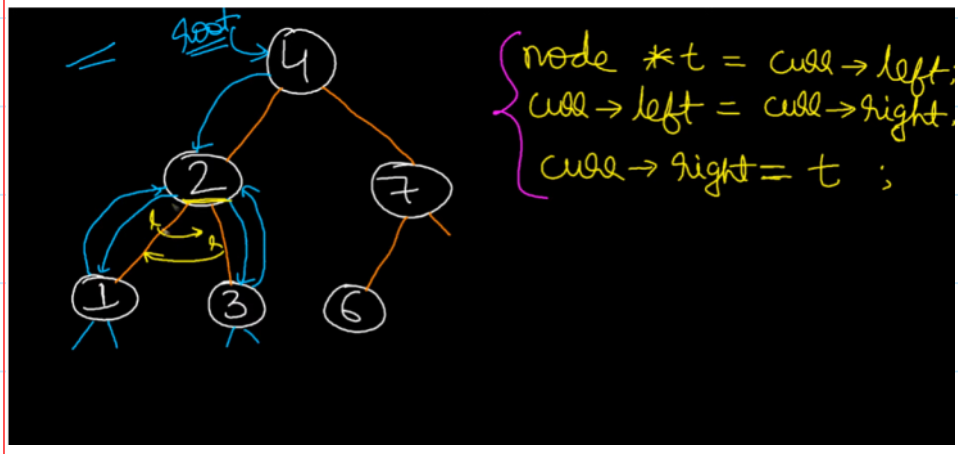


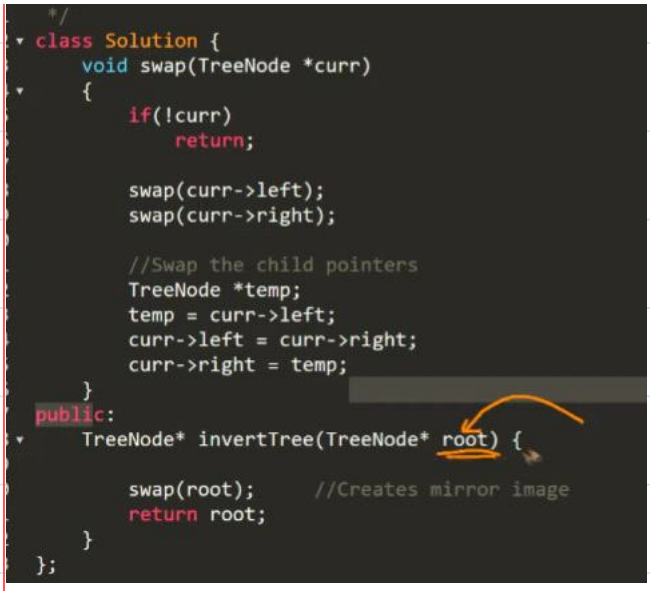
We can play with left and right pointer.

Typical recursion problem:

We will start from root and will post order processing.

After processing left and right side, we will process the current node.





```
*/
class Solution {
    void swap(TreeNode *curr)
    {
        if(!curr)
            return;

        swap(curr->left);
        swap(curr->right);

        //Swap the child pointers
        TreeNode *temp;
        temp = curr->left;
        curr->left = curr->right;
        curr->right = temp;
    }
public:
    TreeNode* invertTree(TreeNode* root) {

        swap(root);     //Creates mirror image
        return root;
    }
};
```

Untitled

10 July 2022 13:29

Removal of element from Vector or dynamic array should happen from right to left.

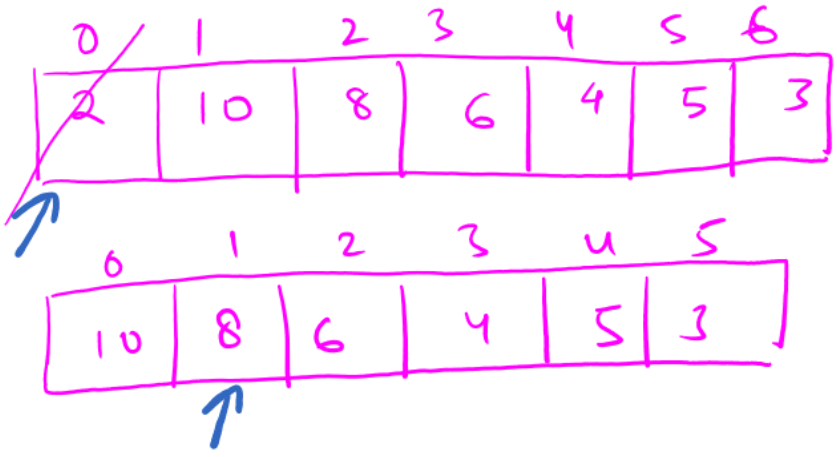
How to remove elements from vector or dynamic array which changes its size with removal?

We should remove from right to left, not from left to right.

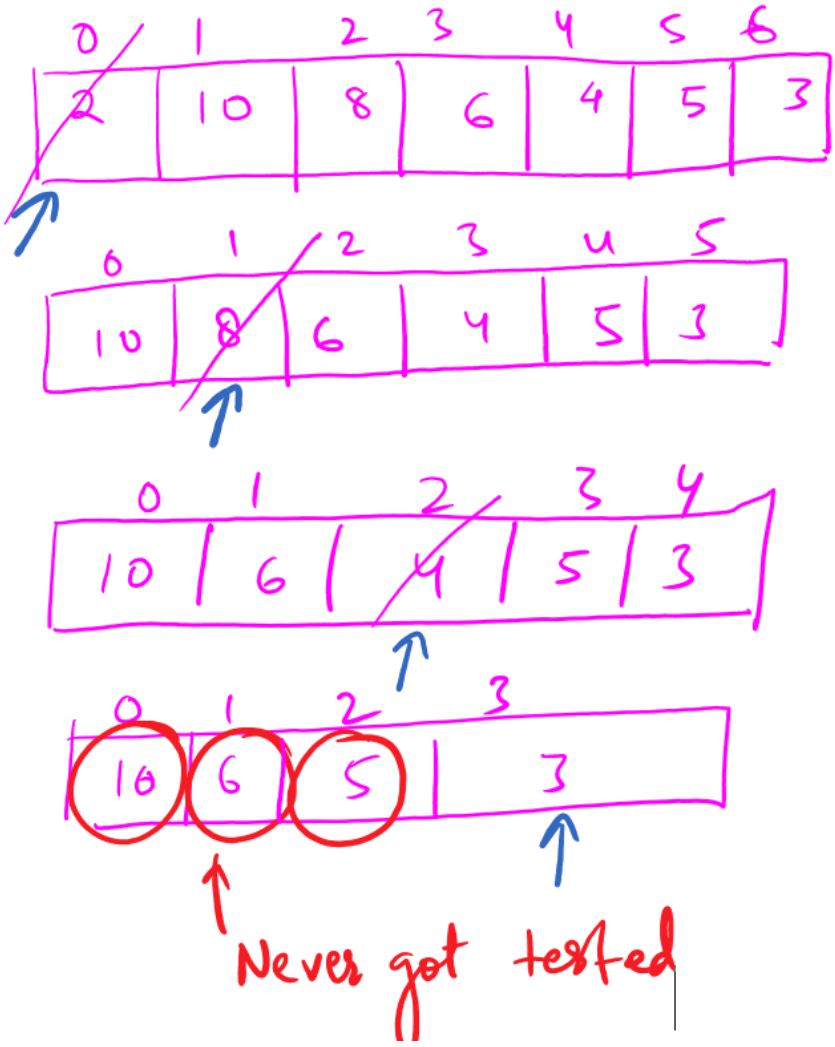
Very very important logical error to understand.

We have to remove all the even elements from the array list

When you removed 2, (i) increased , and 10 never got tested

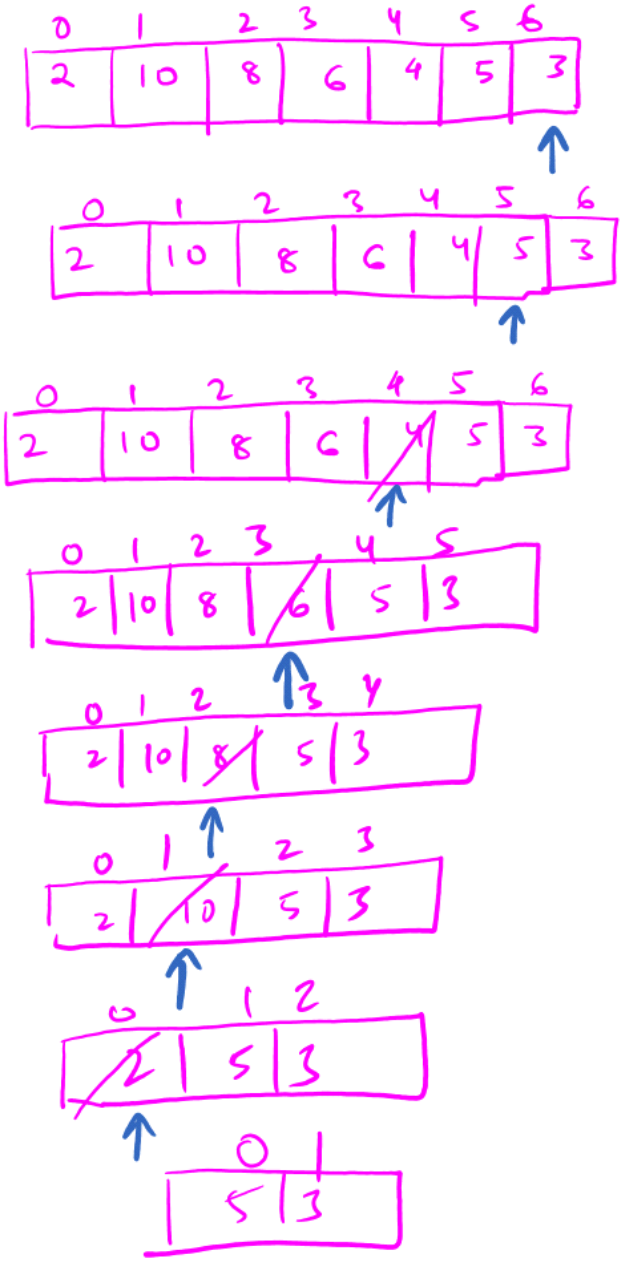


Removal se ye sab aage shift ho jayenge



Removal se cheeze aage badhti rehti hai, to kuch index test hi nahi ho pate, to logical error hai

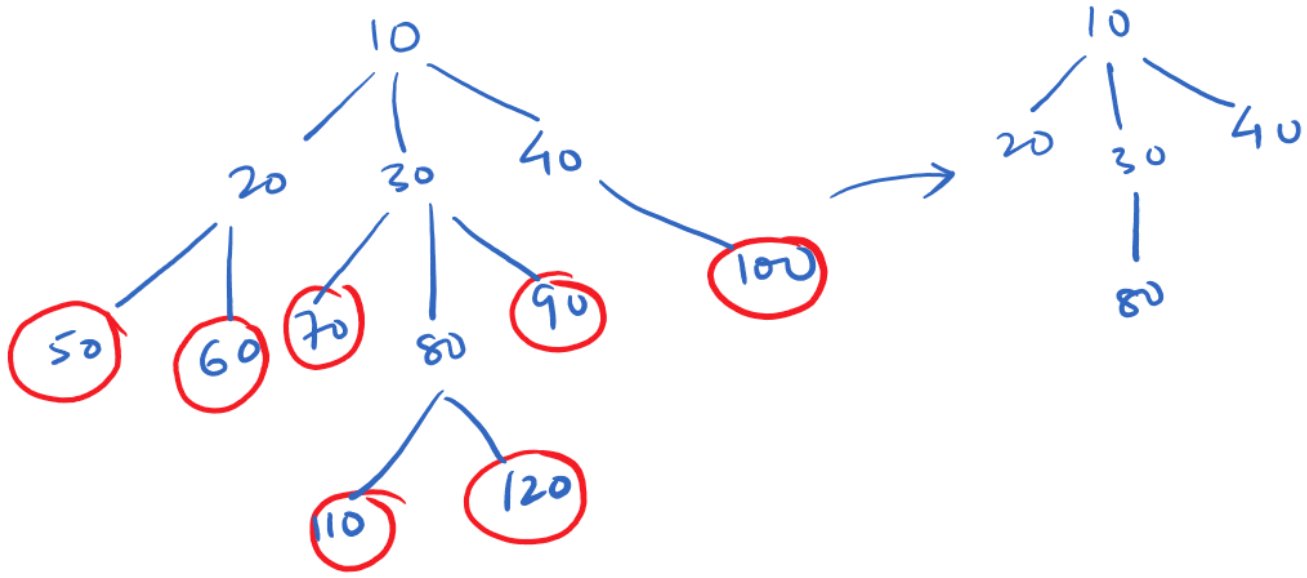
So let's do the same problem from iterating from end to beginning.


0 1 2 3 4 5 6
2 10 8 6 4 5 3
0 1 2 3 4 5 6
2 10 8 6 4 5 3
0 1 2 3 4 5 6
2 10 8 6 4 5 3
0 1 2 3 4 5
2 10 8 6 5 3
0 1 2 3 4
2 10 8 5 3
0 1 2 3
2 10 5 3
0 1 2
2 5 3
0 1
5 3

Remove leaves of the generic tree

26 June 2022 13:23

It is the transformation of tree.



I'll keep the faith that if I make a call on 20, then 20 will shed its leaves.

30 pe call pe, 30 ke leaves udd jayenge.

```
void removeLeaves(Node* node)
{
    for(Node* child: node->children){
        removeLeaves(child);
    }

    for(int i= 0; i< node->children.size(); i++){
        Node* child=node->children[i];
        if(child->children.size()==0){
            node->children.erase(node->children.begin()+i);
        }
    }

}
```

But this one will not give us error, but this is a logical error

Removal of element from Vector or dynamic array should happen from right to left.

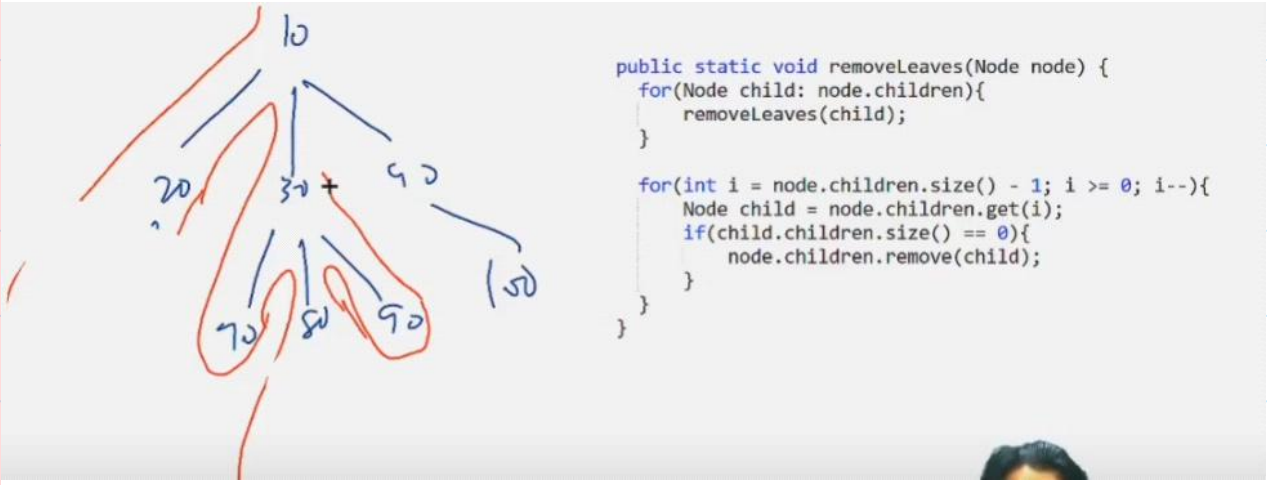
Now we wrote loop in opposite direction

```
void removeLeaves(Node* node)
{
    for(Node* child: node->children){
        removeLeaves(child);
    }

    for(int i= node->children.size()-1; i>=0; --i){
        Node* child=node->children[i];
        if(child->children.size()==0){
            node->children.erase(node->children.begin()+i);
        }
    }

}
```

But still we will not get desirable output

The removal is happening in post order, then we are facing the problem.

Jo leaf the bhi nahi, wo bhi leaf jaise nazaar aa sakte hai



```
public static void removeLeaves(Node node) {
  for(Node child: node.children){
      removeLeaves(child);
  }

  for(int i = node.children.size() - 1; i >= 0; i--){
      Node child = node.children.get(i);
      if(child.children.size() == 0){
          node.children.remove(child);
      }
  }
}
```

80 ko bhi remove kar denge kyunki wo bhi leaf ban chuka hai

To pehle children uda le, aur fir bache hue children pe chala le

```
void removeLeaves(Node* node)
{
    for(int i= node->children.size()-1; i>=0; --i){
        Node* child=node->children[i];
        if(child->children.size()==0){
            node->children.erase(node->children.begin()+i);
        }
    }

    for(Node* child: node->children){
        removeLeaves(child);
    }
}
```

Ab hum preorder me hata rhe hain

Note: depending on the requirement of question, we choose to do the job in preorder or post order.

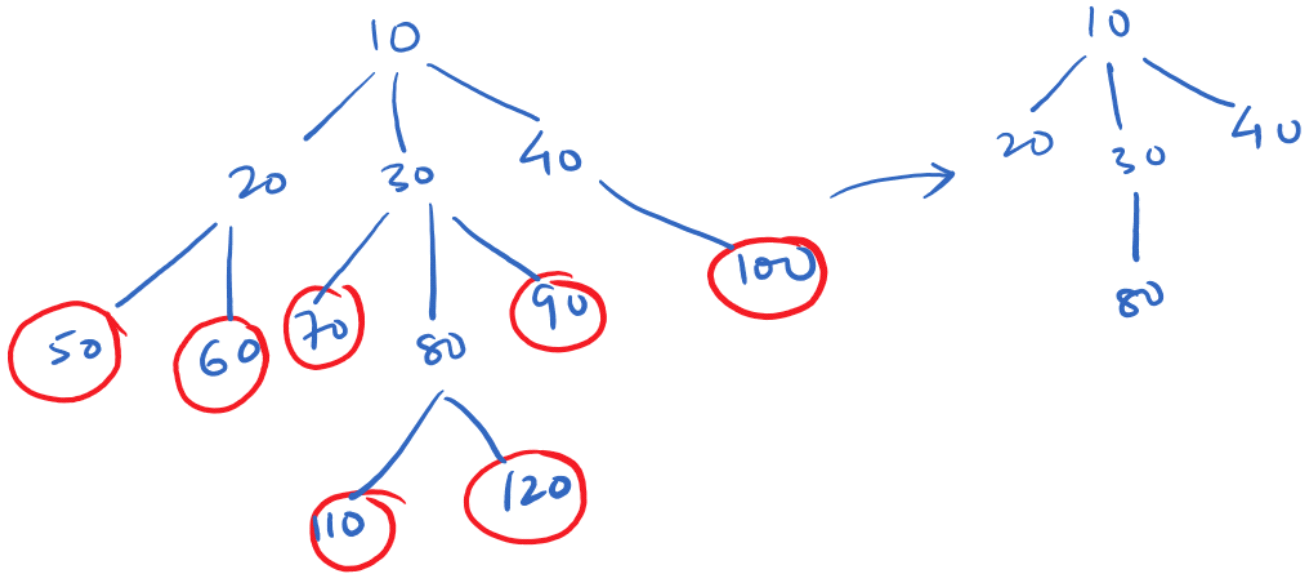
Agar aap child preorder me remove karenge to child remove ho jayenge, agar aap postorder me karenge to sab kuch remove ho jayega, kyunki aap child remove karte hue aa rhe hai, so jo leaf tha bhi nahi, to wo to ud jayega.

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/removeLeafNodesOfTree.java

Untitled

10 July 2022 13:32

Learning is :



Expectation: if I call on 10, then I'll get the one in the right after transformation

Faith: if I call on 20, 20 will shed its leaves,
If I call on 30, 30 will shed its leaves,
If I call on 40, 40 will shed its leaves

But 10 might also have the leaf nodes which need to be removed before making the call on 20, 30 and 40 because after we make a call on 20, 30 and 40, there might be few nodes which will shed its leaves and will appear like a leaf node for 10.

So now Expectation + Faith

removeLeaf ≡ rL

rL(10) = remove leaf nodes of 10 as a node

rL(20)

rL(30)

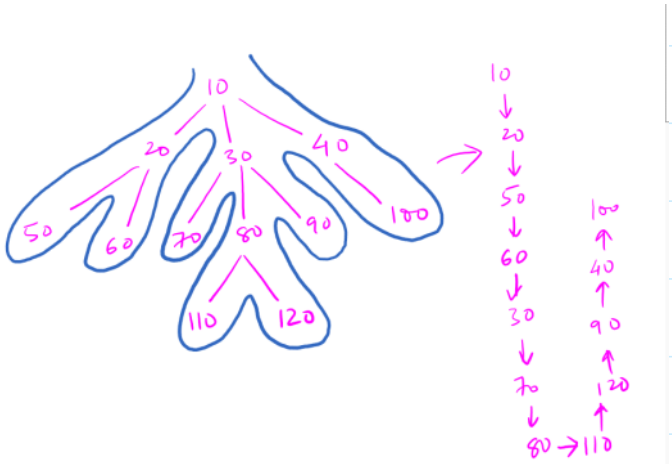
rL(40)

```
void removeLeaves(Node* node)
{
    for(int i= node->children.size()-1; i>=0; --i){
        Node* child=node->children[i];
        if(child->children.size()==0){
            node->children.erase(node->children.begin()+i);
        }
    }

    for(Node* child: node->children){
        removeLeaves(child);
    }
}
```

Untitled

05 July 2022 18:03

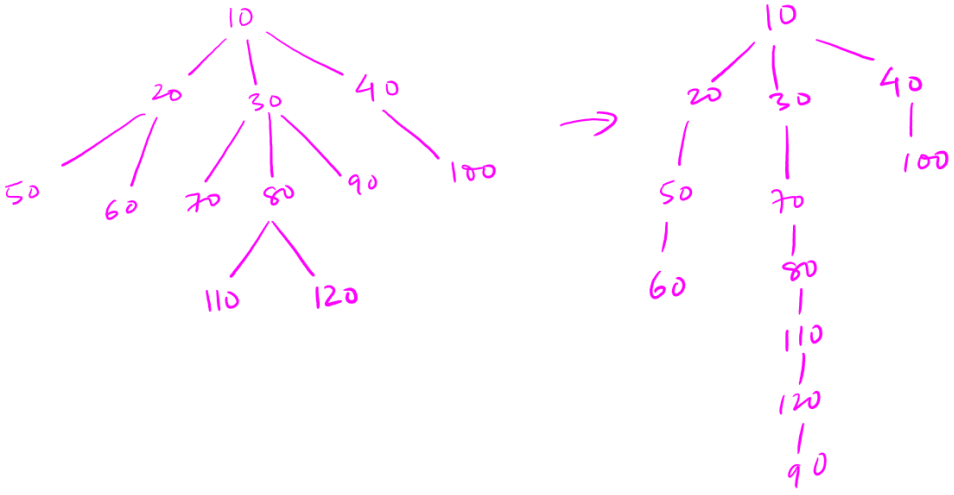
Linearize a generic tree



Linearize in the preorder fashion.

Faith in recursion helps us to think, we only think about our own problem and rest of the problem is supposed to be solved by faith.

20, 30, 40 linear hona janta hai



Agar 20, 30 aur 40 ko call karenge to wo subtree linear hona janta hai, ye

hamara faith hai.

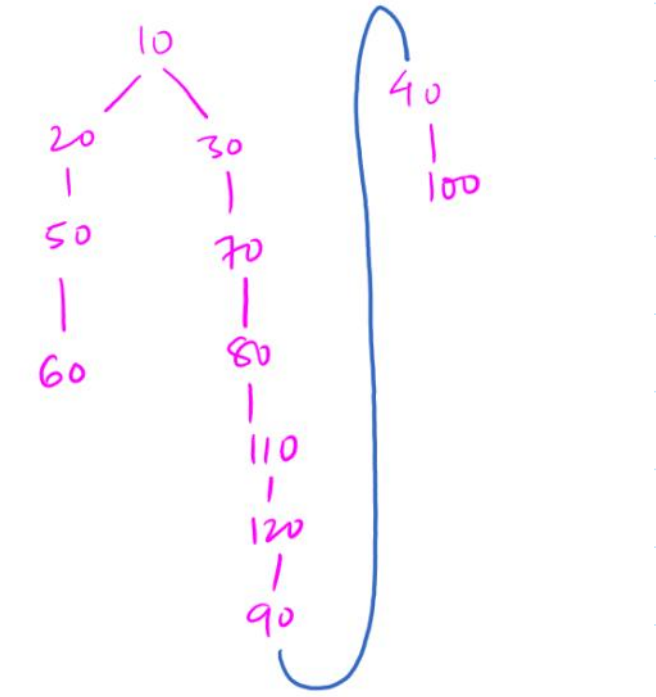
```
void linearize(Node* node)
{
    for(Node* child: node->children)
        linearize(child);
```

Itne code se hum viswas kar sakte hai, ki wo upar diye gaye picture jaisa ho jayega.

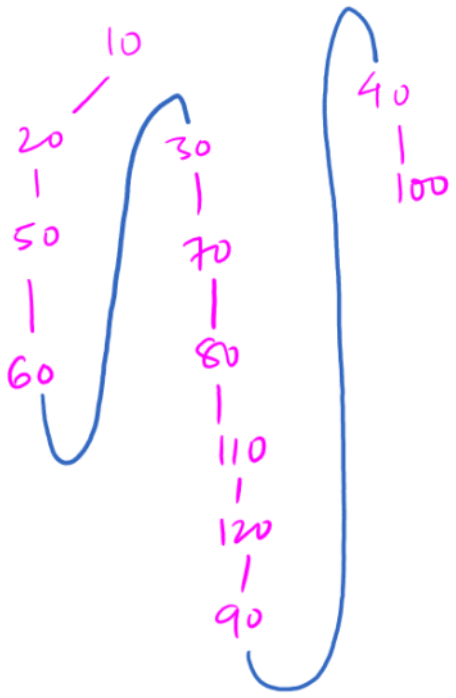
20, 30 aur 40 ne to apna kaam kar diya, par abhi 10 ne apna kaam nahi kiya hai

Ab 10 ko kya karna hai,

1. Akhri ko udaye aur second last ki tail me add kar de



2. Last ko remove kare aur second last ki tail me add kar de



Aisa wo tab tak kare jab tak 10 ka sirf ek hi child na back jaye

```
void linearize(Node* node)
{
    for(Node* child: node->children)
        linearize(child);

    while(node->children.size() > 1)
    {
        Node* lc= node->children.back();
        node->children.pop_back();
        Node* slc= node->children.back();
        Node* tail= getTail(slc);
        tail->children.push_back(lc);
    }
}
```

Lc: last child
Slc: second last
tail: second last tail

Get tail function will help us get the second last tail

Now we will write getTail code

```
Node* getTail(Node * node)
{
    while(node->children.size() > 0)
        node=node->children[0];

    return node;
}

void linearize(Node* node)
{
    for(Node* child: node->children)
        linearize(child);

    while(node->children.size() > 1)
    {
        Node* lc= node->children.back();
        node->children.pop_back();
        Node* slc= node->children.back();
        Node* tail= getTail(slc);
        tail->children.push_back(lc);
    }
}
```

This method is in O(n^2) where n is the no. of node.

Untitled

09 July 2022 15:17

```
#include<bits/stdc++.h>
using namespace std;
struct Node{
    int data;
    vector<Node*> children;
};
Node* newNode(int val)
{
    Node* node = new Node;
    node->data= val;
    return node;
}
class Pair{
    public:
    Node* node;
    int level;
    Pair(Node* node, int level)
    {
        this->node= node;
        this->level= level;
    }
};

Node* constructor(vector<int> &arr, int n)
{
    //1. create a root pointer and point it to NULL
    Node* root=NULL;
    //2. create a stack for the operation
    stack< Node*> st;
    //3. run the loop on all the elements of array
    for(int i=0; i<n; i++)
    {
        //4. if you encounter number other than -1
        if(arr[i]!= -1)
        {
            //4.11 Create a new node and fill the
value
            Node* t= newNode(arr[i]);
```

```
            //5.1 if stack empty, then make it root
            if(st.empty())
            {
                root=t;
            }
            else
            {
                //5.2 If not empty, then add to
childrens of node on top of stack
                st.top()->children.push_back(t);
            }
            //4.12 Push the node in the stack
            st.push(t);
        }
        else
        {
            //4.2 if you encounter -1, then pop from
stack
            st.pop();
        }
    }
    //return the root value
    return root;
}
void levelOrder(Node* node)
{
    //1. create just one queue
    queue<Pair> q;
    // push the root with level 1
    q.push(Pair(node, 1));
    // create a variable level to keep track of level
    int level=1;
    // loop till the queue is not empty
    while(q.size()>0)
    {
        // REMOVE
        Pair rem= q.front();
        q.pop();
        //UPDATE level
        if(rem.level > level)
        {
            cout<<endl;
            level= rem.level;
```

```
        }
        //PRINT
        cout<<rem.node-> data<< " ";
        //ADD with increased level
        for(Node* child: rem.node-> children)
            q.push(Pair(child, rem.level+1));
    }
}
Node* getTail(Node * node)
{
    while(node->children.size() > 0)
        node=node->children[0];
    return node;
}
void linearize(Node* node)
{
    for(Node* child: node->children)
        linearize(child);
    while(node->children.size() > 1)
    {
        Node* lc= node->children.back();
        node->children.pop_back();
        Node* slc= node->children.back();
        Node* tail= getTail(slc);
        tail->children.push_back(lc);
    }
}


int main()
{
    vector<int> arr;
    arr.assign({10, 20, 50, -1, 60, -1, -1, 30,
70, -1, 80, 110, -1, 120, -1, -1, 90, -1, -1, 40,
100, -1, -1, -1});
    int n= arr.size();
    Node* root=NULL;
    root=constructor(arr,n);
    levelOrder(root);
    linearize(root);
    cout<<endl<<"mirror created"<<endl;
    levelOrder(root);
    return 0;
}
```

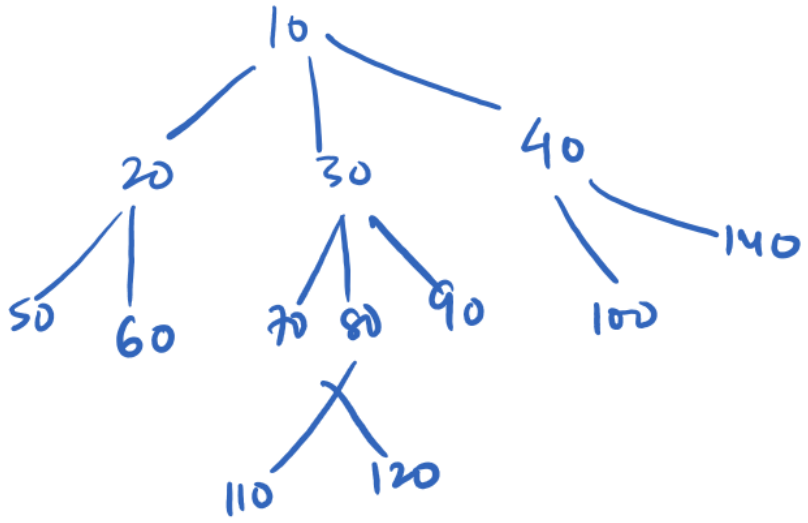
Untitled

06 July 2022 15:16

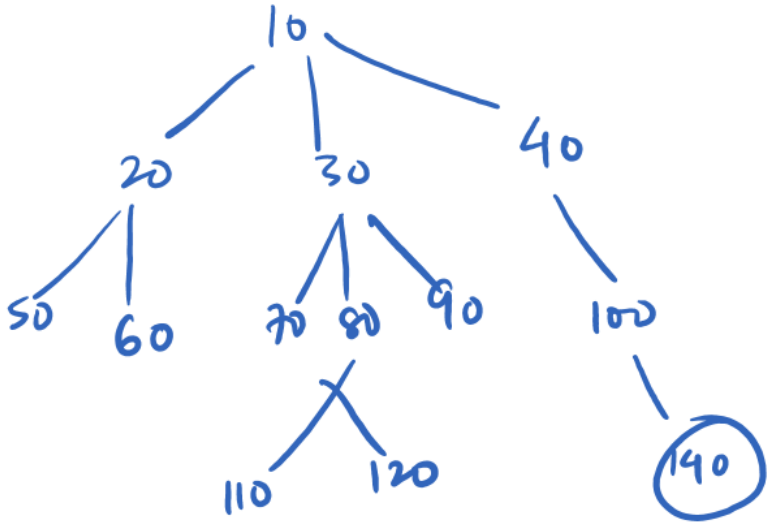
Har node ke right side me hum ek loop lagate hai to get the tail in the last approach, n nodes me n nodes ko chalaoge to complexity ho jayegi O(n^2)

Faith badal rha hu efficient banane ke lie.

Linearize na-kewal children ko linearize kar deta hai, balki hame tail bhi return karega.



Akhri node pe pehle linearize call karenge, wo apko tail de dega

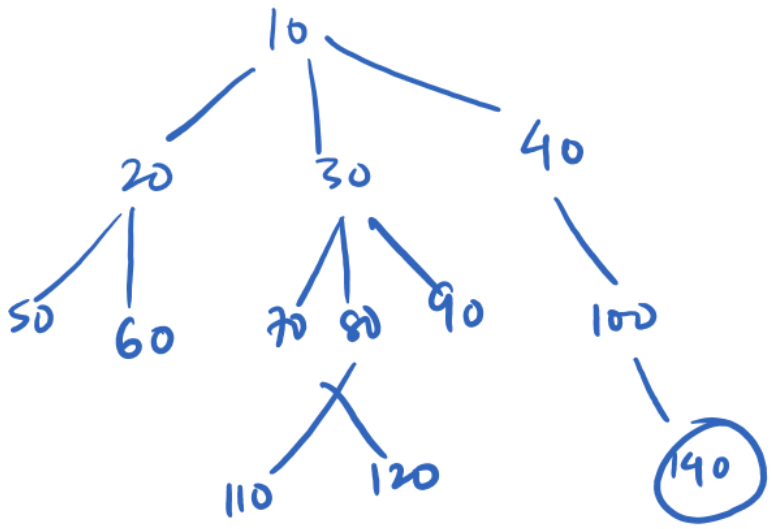


Tail is 140

```
Node* linearize2(Node* node)
{
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->children.size() -1]);
```

Is line se last wala seedha bhi ho gya, aur hame tail bhi mil gayi … i.e. 140

Ab hum second last ko linearise karenge aur uski tail milegi



30 wale ko karenge linearise ab hum log (which is second last)

Lekin usse pehle last wale ko remove kar lenge

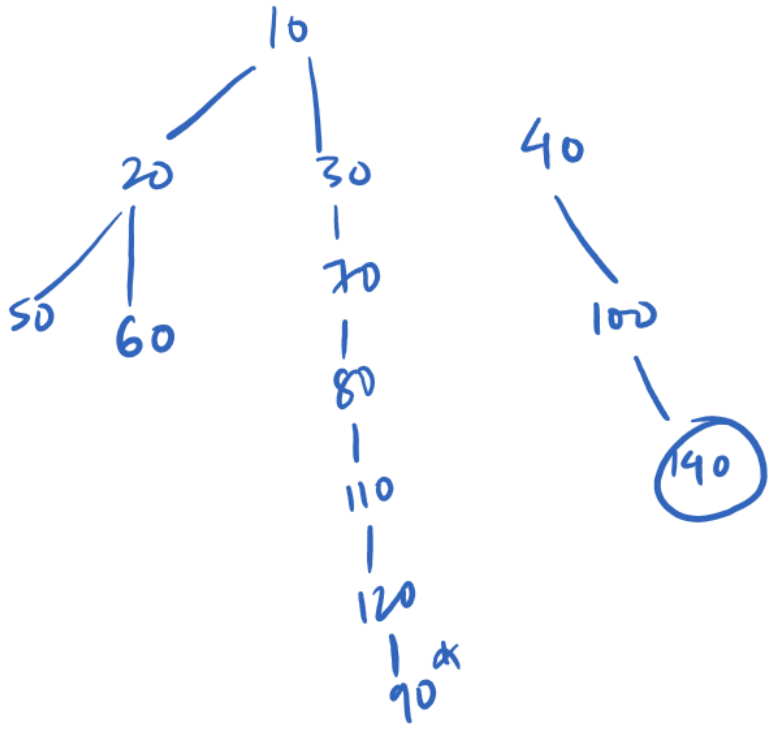
```
Node* linearize2(Node* node)
{
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->children.size() -1]);

    //run loop jab tak node ke children ka size 1 se bara hai
    while(node->children.size()>1)
    {   //last ko remove kar liya
        Node* last= node->children.back();
        node->children.pop_back();
```

Ab second last ko linearize karke uski tail mangayenge

```
Node* linearize2(Node* node)
{
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->children.size() -1]);

    //run loop jab tak node ke children ka size 1 se bara hai
    while(node->children.size()>1)
    {   //last ko remove kar liya
        Node* last= node->children.back();
        node->children.pop_back();

        Node* secondLast=node->children.back();
        Node* sklt= linearize2(secondLast); //sklt: secondLast ki tail
```

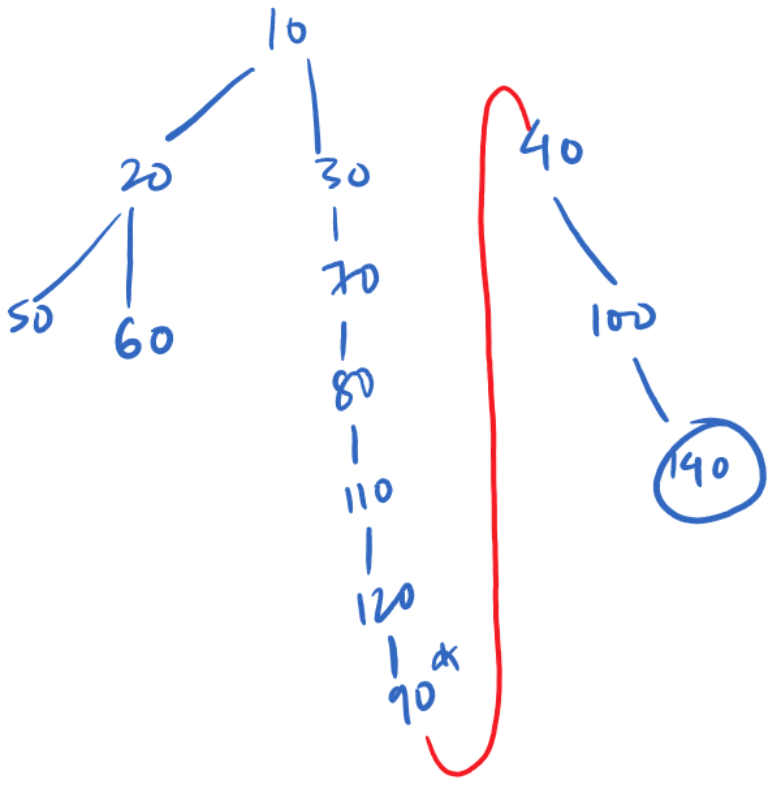
Last wale ko remove kardiya to ab second last wala back me aa jayega.

Ab hamare faith se na ki wo sirf linear ho gya, balki usse mujhe tail bhi mil gayi



Ab second last ki tail me 40 ko add karna hai

```
Node* linearize2(Node* node)
{
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->children.size() -1]);

    //run loop jab tak node ke children ka size 1 se bara hai
    while(node->children.size()>1)
    {   //last ko remove kar liya
        Node* last= node->children.back();
        node->children.pop_back();

        Node* secondLast=node->children.back();
        Node* sklt= linearize2(secondLast); //sklt: secondLast ki tail

        //secondLast ki tail me add karde last ko
        sklt->children.push_back(last);
    }
```



Ab hame return kya karna hai??

Overall ki tail hai, lkt: last ki tail jo hamne nikali thi which is at 140

So we will return it.

```
Node* linearize2(Node* node)
{
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->children.size() -1]);

    //run loop jab tak node ke children ka size 1 se bara hai
    while(node->children.size()>1)
    {   //last ko remove kar liya
        Node* last= node->children.back();
        node->children.pop_back();

        Node* secondLast=node->children.back();
        Node* sklt= linearize2(secondLast); //sklt: secondLast ki tail

        //secondLast ki tail me add karde last ko
        sklt->children.push_back(last);
    }
    return lkt;
}
```

Ab hame ek base case chahie hoga, kyunki agar children ka size 0 ho gya, to lkt calculate karte time, hame segmentation fault aa jayega

As node->children[0 - 1] will be nothing

```
//lkt: last ki tail
Node* lkt = linearize2(node->children[node->children.size() -1]);
```

Ye line me -1 ho jayega, to problem aa jayegi

```
    //leaf node is base case, then return itself as it is tail of it's
    //linear version. [already linear, single node]
    if(node->children.size())
    {
        return node;
    }
```

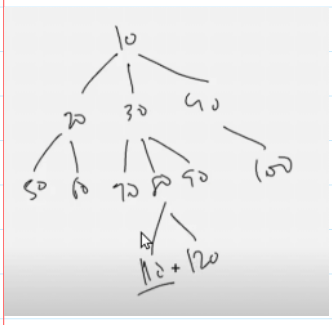
So finally the code is

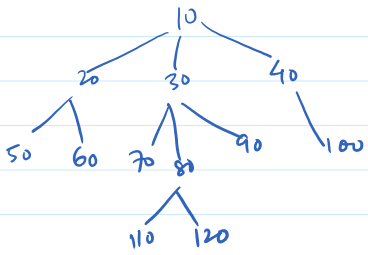
```
Node* linearize2(Node* node)
{
    //leaf node is base case, then return itself as it is tail of it's
    //linear version. [already linear, single node]
    if(node->children.size()==0)
    {
        return node;
    }
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->children.size() -1]);

    //run loop jab tak node ke children ka size 1 se bara hai
    while(node->children.size()>1)
    {   //last ko remove kar liya
        Node* last= node->children.back();
        node->children.pop_back();

        Node* secondLast=node->children.back();
        Node* sklt= linearize2(secondLast); //sklt: secondLast ki tail

        //secondLast ki tail me add karde last ko
        sklt->children.push_back(last);
    }
    return lkt;
}
```

```
Node* linearize2(Node* node)
{
    //leaf node is base case, then return itself as it
is tail of it's
    //linear version. [already linear, single node]
    if(node->children.size()==0)
    {
        return node;
    }
    //lkt: last ki tail
    Node* lkt = linearize2(node->children[node->
children.size() -1]);

    //run loop jab tak node ke children ka size 1 se
bara hai
    while(node->children.size()>1)
    {   //last ko remove kar liya
        Node* last= node->children.back();
        node->children.pop_back();
        Node* secondLast=node->children.back();
        Node* sklt= linearize2(secondLast); //sklt:
secondLast ki tail
        //secondLast ki tail me add karde last ko
        sklt->children.push_back(last);
    }
    return lkt;
}
```

Find in generic tree

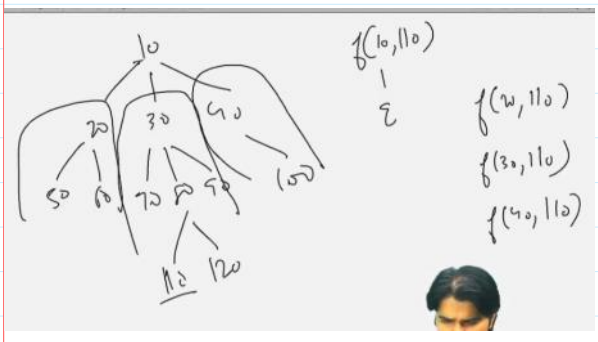
09 July 2022 15:41

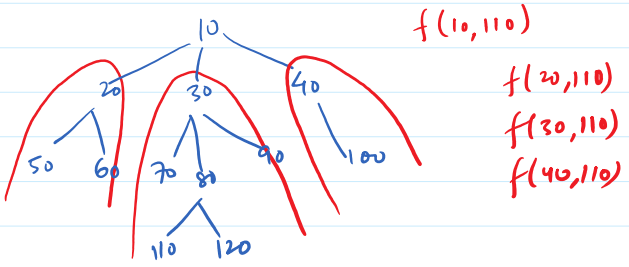




Data mil gya to true return karenge nahi to false return karenge

Faith ye hai ki 20, 30 aur 40 apne subtree me dundhna jante hai,





Agar khud hi 110 ke barabar hai to return true karega,
Nahi mila to 20 se puchega, agar mil gya to true return karega, nahi to

```
//agar true mil jata hai to aage loop ko abort kar deta hai
```

fir 30 se puchega, agar mil gya to true return karega, nahi to

```
//agar true mil jata hai to aage loop ko abort kar deta hai
```

fir 40 se puchega

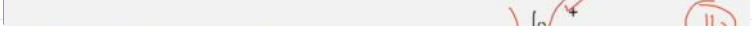
In this case, euler adha hi chalta hai

```
bool find(Node* node, int data)
{
    if(node->data == data)
        return true;

    for(Node* child: node->children)
    {
        bool fic= find(child, data);
        //agar true mil jata hai to aage loop ko abort kar deta hai
        if(fic == true)
            return true;
    }
    return false;
}
```

Adha euler chalega. [half euler]

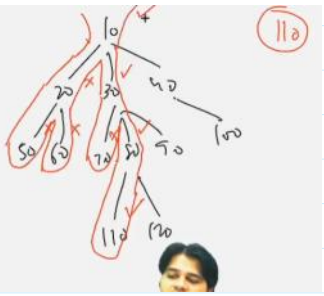
Jaise hi ise answer mil jata hai, ye seedha ghar ki taraf bhagta hai
aage loop ko abort karte hue

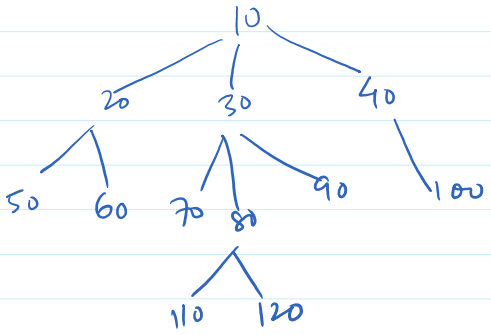
Please do the dry run



Please do the dry run

```
public static boolean find(Node node, int data) {
    if(node.data == data){
        return true;
    }

    for(Node child: node.children){
        boolean fic = find(child, data);
        if(fic){
            return true;
        }
    }

    return false;
}
```



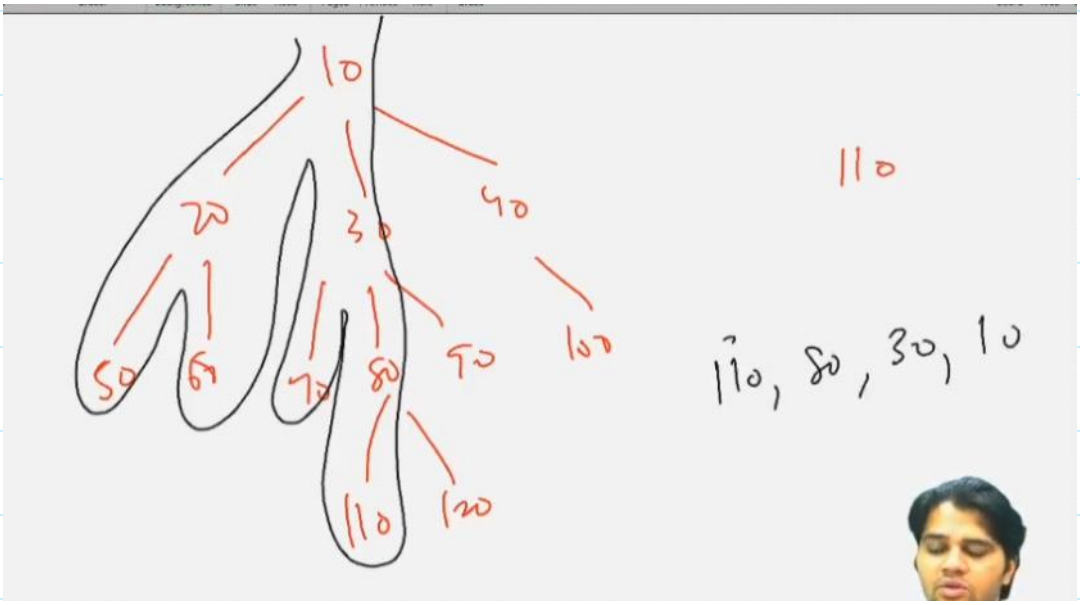


Node to root path

09 July 2022 15:55

Adha euler tree chalta hai aur wo loop ko abort karke seedha ghar ko bhagta hai

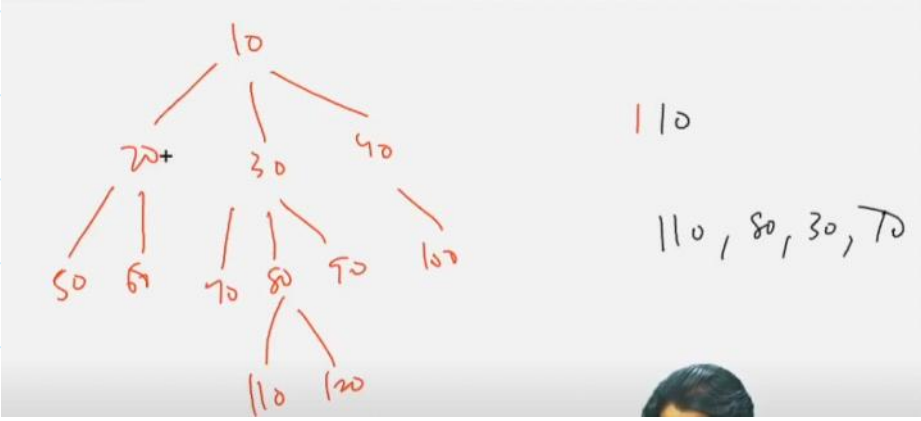
Use the same concept here.



False ke barabar hogi hamari khali array list
True ke barabar hogi hamari bhari hue array list

It's just like find an element in generic tree.

Hamari expectation 10 se ye hai ki agar 110 mil gaya to hame wo 110 se 10 tak ka path return karega

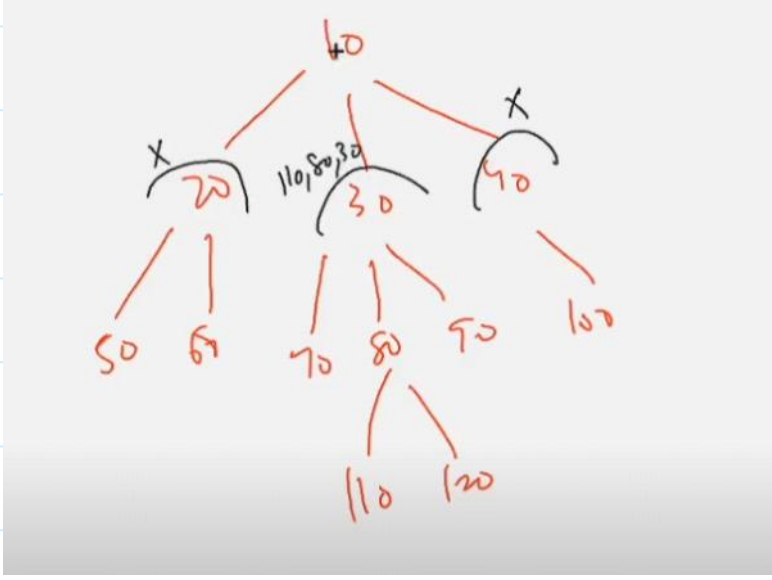


Faith ye hai ki agar wo 20 me mil gya, to 110 se 20 tak ka path return karta

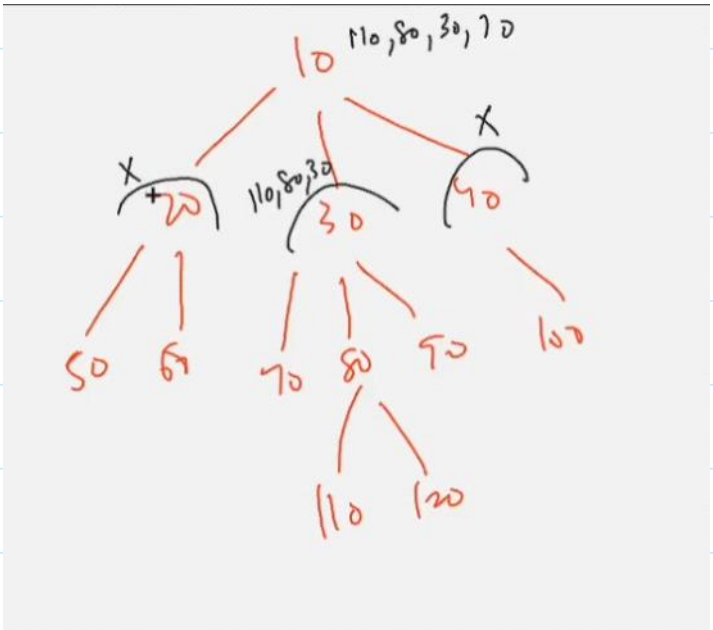
Agar 30 me mil jata to , 110 se 30 tak ka path return karta

Agar 40 me mil jata to , 110 se 40 tak ka path return karta

20 me to milega nahi , 30 me milega (110, 80, 30) , aur 40 me milega nahi



Ab 10 kya karega, jiske yaha se rasta aya, array list ayi, uske raste me khud ko add karke return kar dega



```cpp
vector<int> nodeToRoot(Node* node, int data)
{
    if(node->data == data)
    {
        //agar mil gya to vector bana ke, usme daal ke return kar do
        vector<int> vec;
        vec.push_back(node->data);
        return vec;
    }
    for(Node* child: node->children)
    {
        vector<int> ptc = nodeToRoot(child, data); //ptc: path till child
        if(ptc.size() > 0)
        {   //agar mil gya to node ke data ko chipka ke return kar do
            ptc.push_back(node->data);
            return ptc;
        }
    }

    //nahi mila to khali vector bana ke return kar do
   vector<int> khali;
   return khali;
}
```

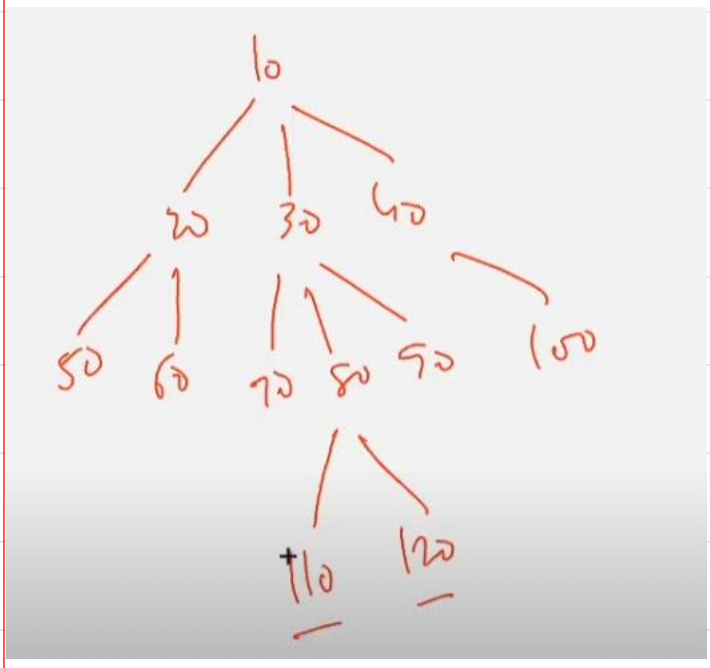
```cpp
vector<int> nodeToRoot(Node* node, int data)
{
```

```
    if(node->data == data)
    {
        //agar mil gya to vector bana ke, usme daal
ke return kar do
        vector<int> vec;
        vec.push_back(node->data);
        return vec;
    }
    for(Node* child: node->children)
    {
        vector<int> ptc = nodeToRoot(child, data);
//ptc: path till child
        if(ptc.size() > 0)
        {   //agar mil gya to node ke data ko chipka
ke return kar do
            ptc.push_back(node->data);
            return ptc;
        }
    }
    //nahi mila to khali vector bana ke return kar
do
   vector<int> khali;
   return khali;
}
```

Lowest common ancestor

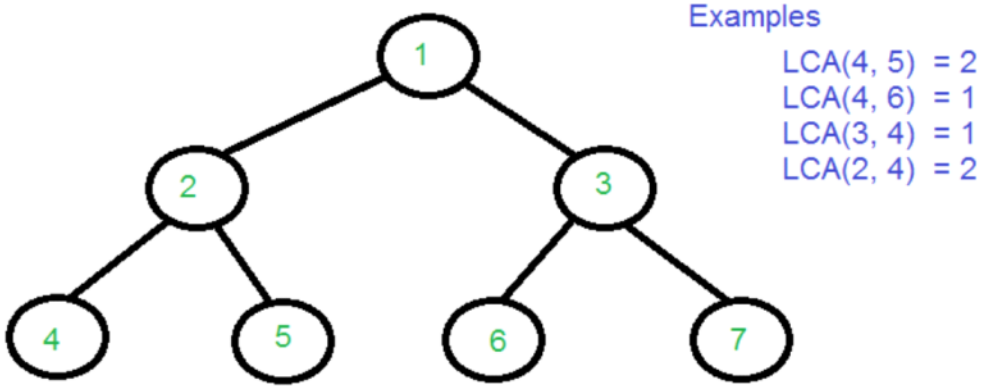
09 July 2022 16:25

A tree is given
Two values which exist in tree are given



Let T be a rooted tree. The lowest common ancestor between two nodes n1 and n2 is defined as the lowest node in T that has both n1 and n2 as descendants (where we allow a node to be a descendant of itself).

The LCA of n1 and n2 in T is the shared ancestor of n1 and n2 that is located farthest from the root. Computation of lowest common ancestors may be useful, for instance, as part of a procedure for determining the distance between pairs of nodes in a tree: the distance from n1 to n2 can be computed as the distance from the root to n1, plus the distance from the root to n2, minus twice the distance from the root to their lowest common ancestor. (Source Wiki)



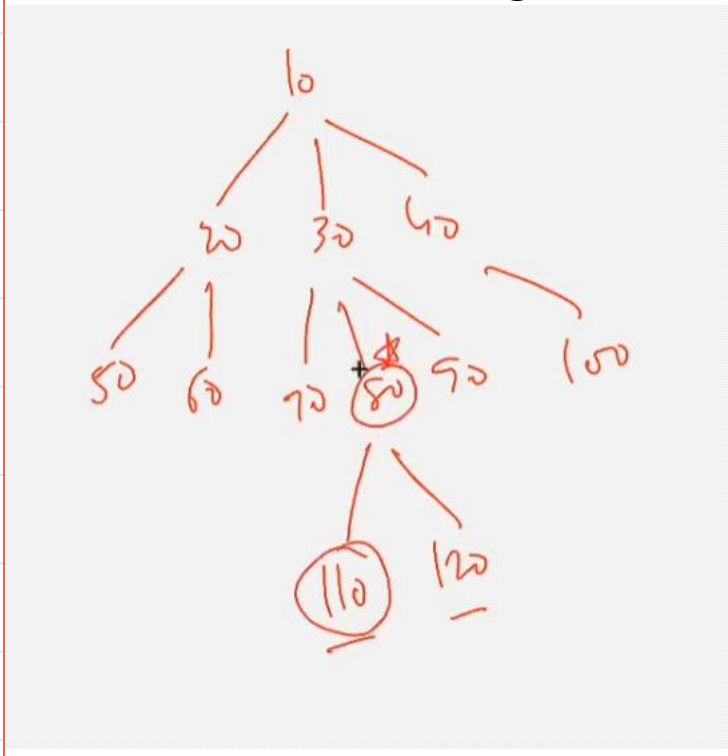
110 aur 120 ka lowest common ancestor

To answer hoga 80.

70 aur 110 ke case me 30 hoga

60 aur 100 ke case me 10 hoga

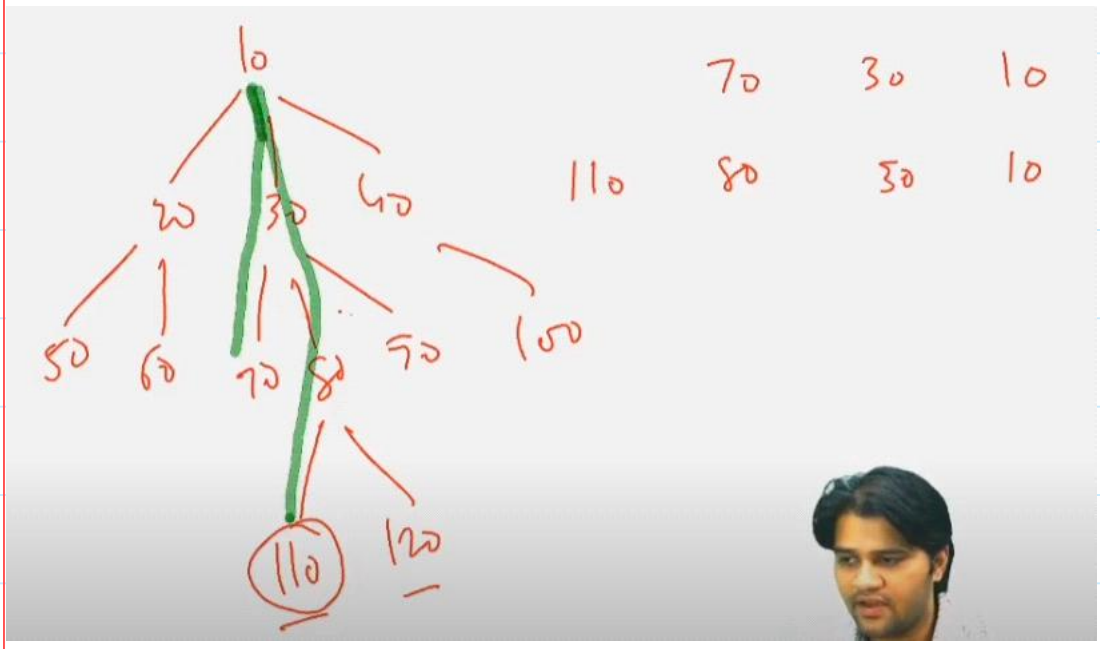
80 aur 110 me 80 hi hoga



Untitled

09 July 2022 16:30

70 aur 110 ka nikale



Node to root path likh lenge

Jaha apka pehla unequal element detect ho, usse piche wale ko return kar denge

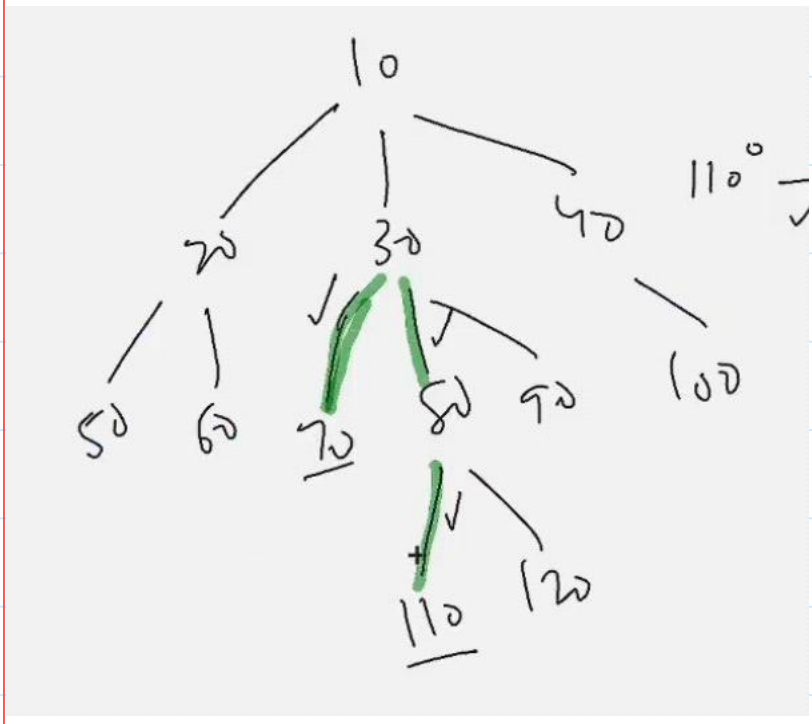
Ulta loop chaliyega node to root path me

```cpp
vector<int> nodeToRoot(Node* node, int data)
{
    if(node->data == data)
    {
        //agar mil gya to vector bana ke, usme daal ke return kar do
        vector<int> vec;
        vec.push_back(node->data);
        return vec;
    }
    for(Node* child: node->children)
    {
        vector<int> ptc = nodeToRoot(child, data); //ptc: path till child
        if(ptc.size() > 0)
        {   //agar mil gya to node ke data ko chipka ke return kar do
            ptc.push_back(node->data);
            return ptc;
        }
    }

    //nahi mila to khali vector bana ke return kar do
   vector<int> khali;
   return khali;
}


int lca(Node* node, int d1, int d2)
{
    vector<int> path1= nodeToRoot(node, d1);
    vector<int> path2= nodeToRoot(node, d2);

    int i = path1.size()-1;
    int j = path2.size()-1;
    while(i>=0 && j>=0 && path1[i] == path2[j]){
        i--;
        j--;
    }
    i++;
    j++;
    return path1[i];
}
```

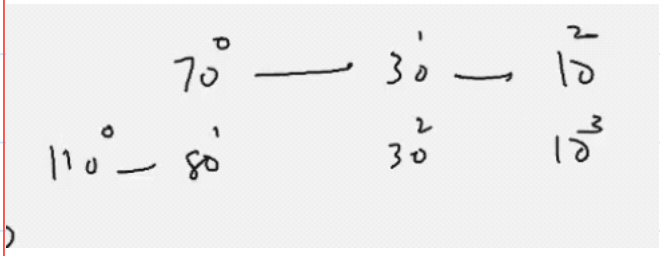
Distance between nodes

09 July 2022 16:35

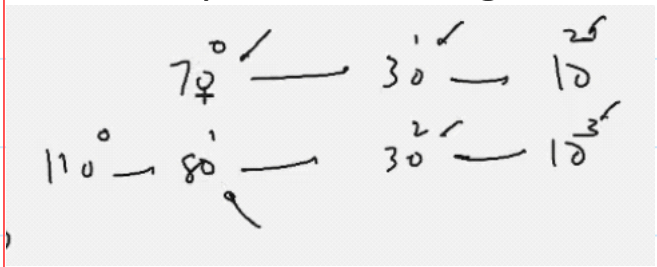
Distance between 70 and 110 in terms of edges
Solution will be 3



We will use node to root path concept.



10 se loop start karenge



Now 70 and 80 are not equal

Break karenge aur wapas I aur j ko badha denge aur akhri equal me chale jayenge

Hum waha pahuchna chahte hai jaha se rasta diverge ho jata hai, basically find LCA

```
int distanceBetweenNodes(Node* node, int d1, int d2)
{
    vector<int> path1= nodeToRoot(node, d1);
    vector<int> path2= nodeToRoot(node, d2);

    int i = path1.size()-1;
    int j = path2.size()-1;
    while(i>=0 && j>=0 && path1[i] == path2[j]){
        i--;
        j--;
    }
    i++;
    j++;


}
```

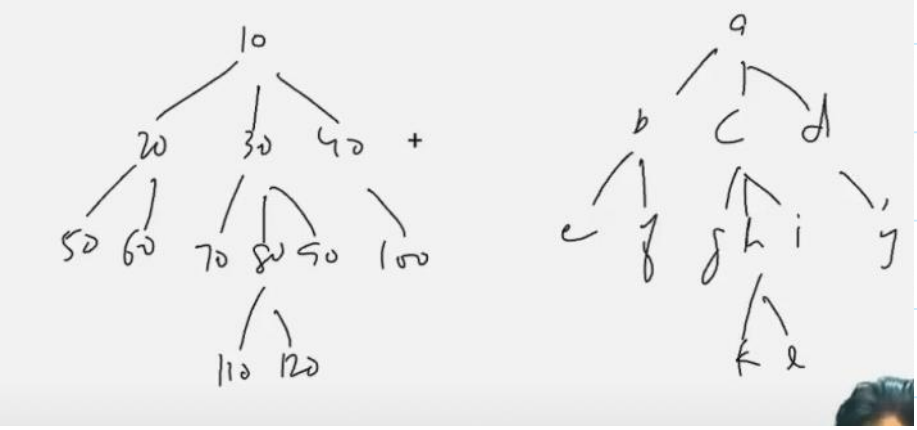
Yaha pe is stage me I bata rha hai ki 70 se kitna dur hai, aur j bata rha ki 110 ki kitna dur hai

So return i+j

```
int distanceBetweenNodes(Node* node, int d1, int d2)
{
    vector<int> path1= nodeToRoot(node, d1);
    vector<int> path2= nodeToRoot(node, d2);

    int i = path1.size()-1;
    int j = path2.size()-1;
    while(i>=0 && j>=0 && path1[i] == path2[j]){
        i--;
        j--;
    }
    i++;
    j++;

    return i+j;
}
```

Are trees similar in shape

09 July 2022 16:50



Kya dono tree ki shape same hai, data se koi faraq nahi padta hai

Are similar :::: aS

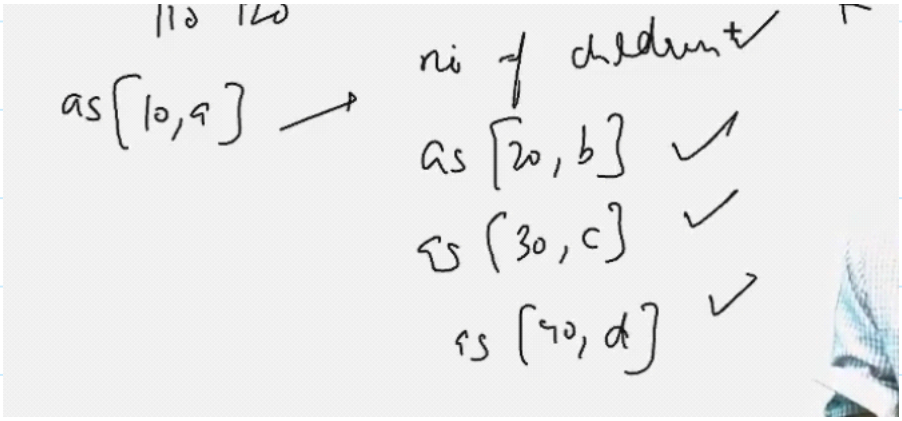
Root of both tree is given : 10 and a

areSimilar(10, a) ::: aS(10, a)

For this both of these should have same number of children
And
Their corresponding child should also be similar

So,areSimilar(20, b);
areSimilar(30, c);
areSimilar(40, d);

High level code



20 jo ho so b ke sath similar ho
30 jo ho so c ke sath similar ho
Aur 40 jo ho so d ke sath similar ho

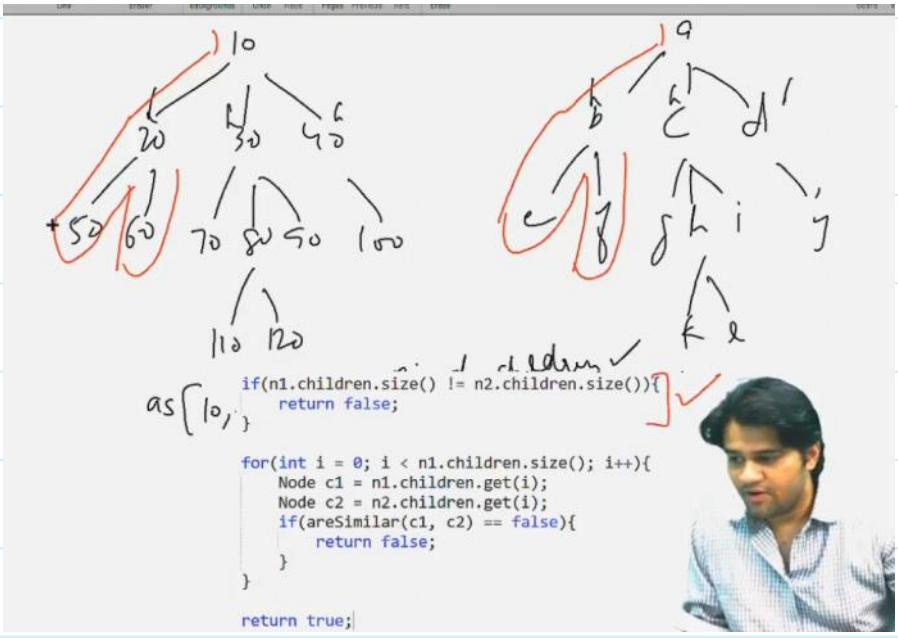
Aur no of children same ho 10 aur a ke

To 10 and a are of similar shape.

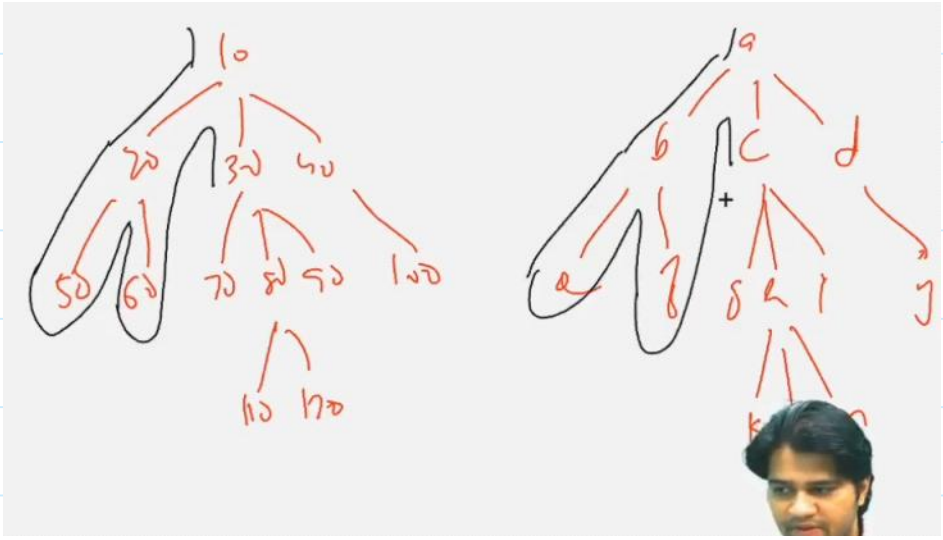
```cpp
bool areSimilar(Node* n1, Node* n2)
{
    if(n1->children.size() != n2->children.size())
    {
        return false;
    }

    for(int i=0; i< n1->children.size(); i++)
    {
        Node* c1 = n1->children[i];
        Node* c2 = n2->children[i];

        if(areSimilar(c1, c2) == false)
            return false;

    }
    return true;
}
```

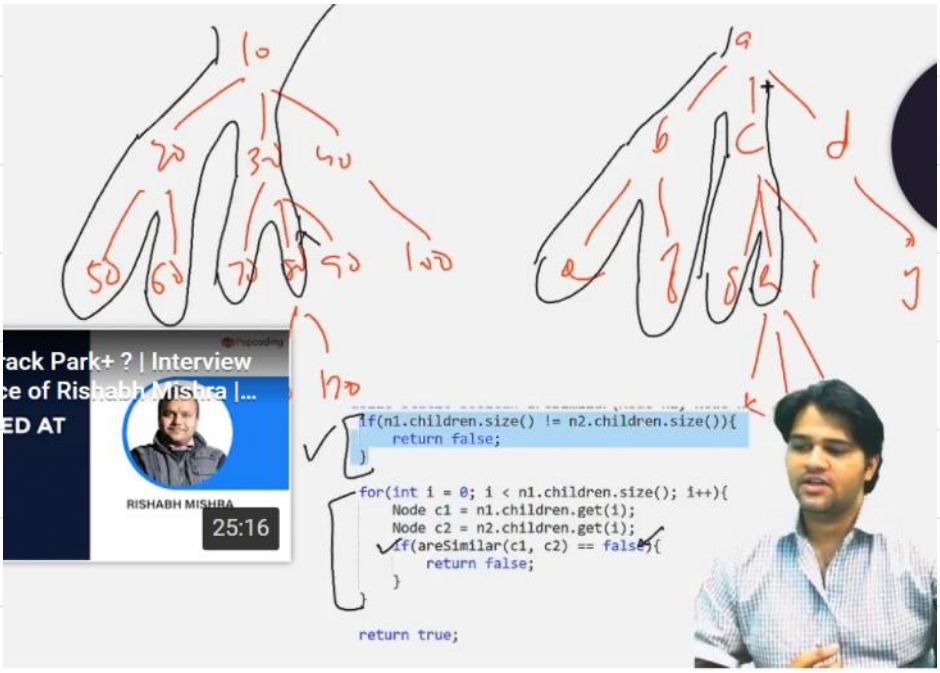
Kaafi find jaisa code hai ye

Now let's do the euler path and dry run



Ek baar ek false case bhi chala ke dekh lo

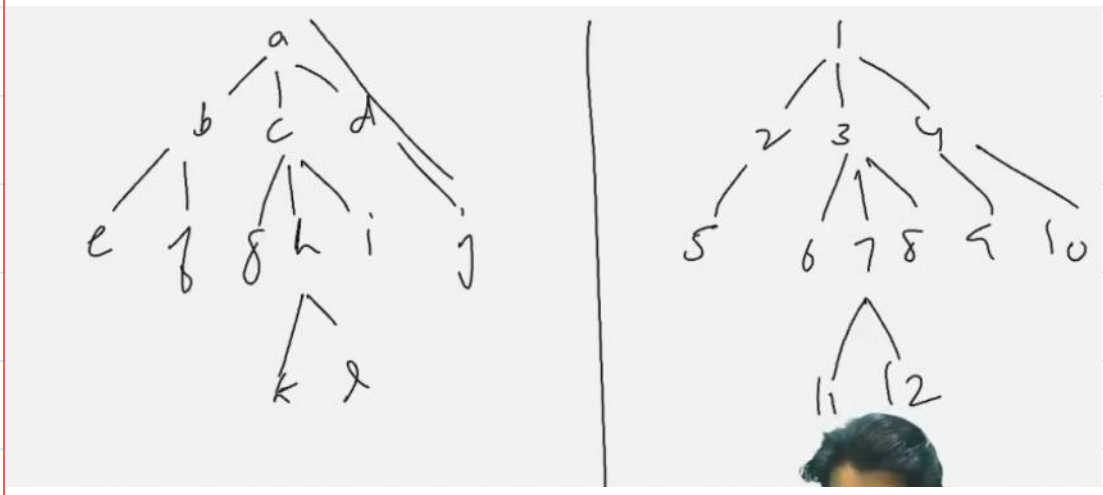




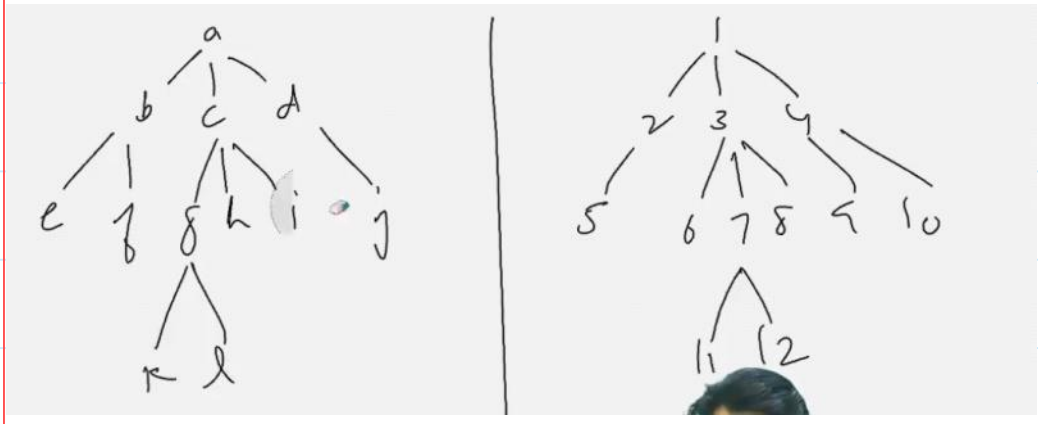
Adha euler chalega aur wahi se false return kar dega

Are tree mirror in shape

09 July 2022 17:12



Ye dono ek dusre ka mirror shape hai, ignoring the data



Ye dono ek dusre ki mirror image nahi hai

Untitled

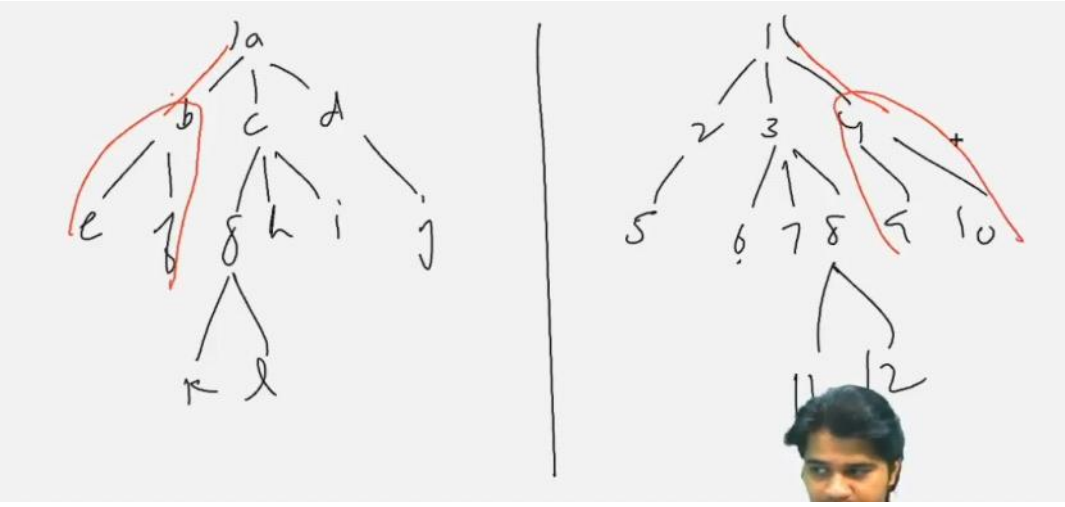
09 July 2022 17:14

High level socho

Agar a aur 1 ko call kare

To pehla to unke no of childrens same hone chahie

Aur a ka pehla child b wala subtree, should be mirror image of 1 ka last child 4 ka subtree



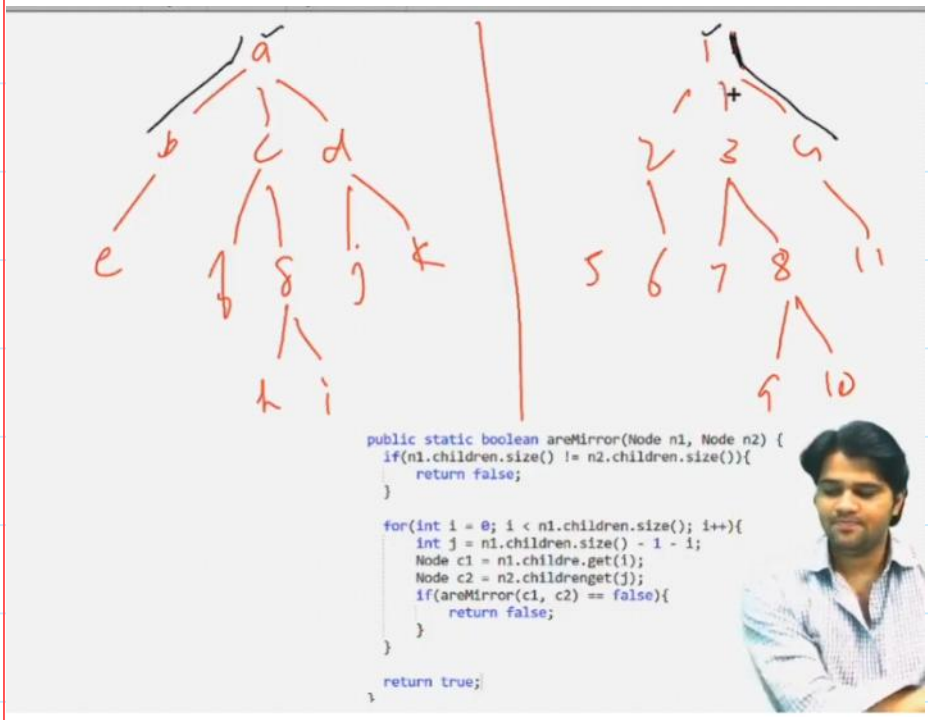
Fir uska dusra child, iske piche se dusre child ki mirror image honi chahie

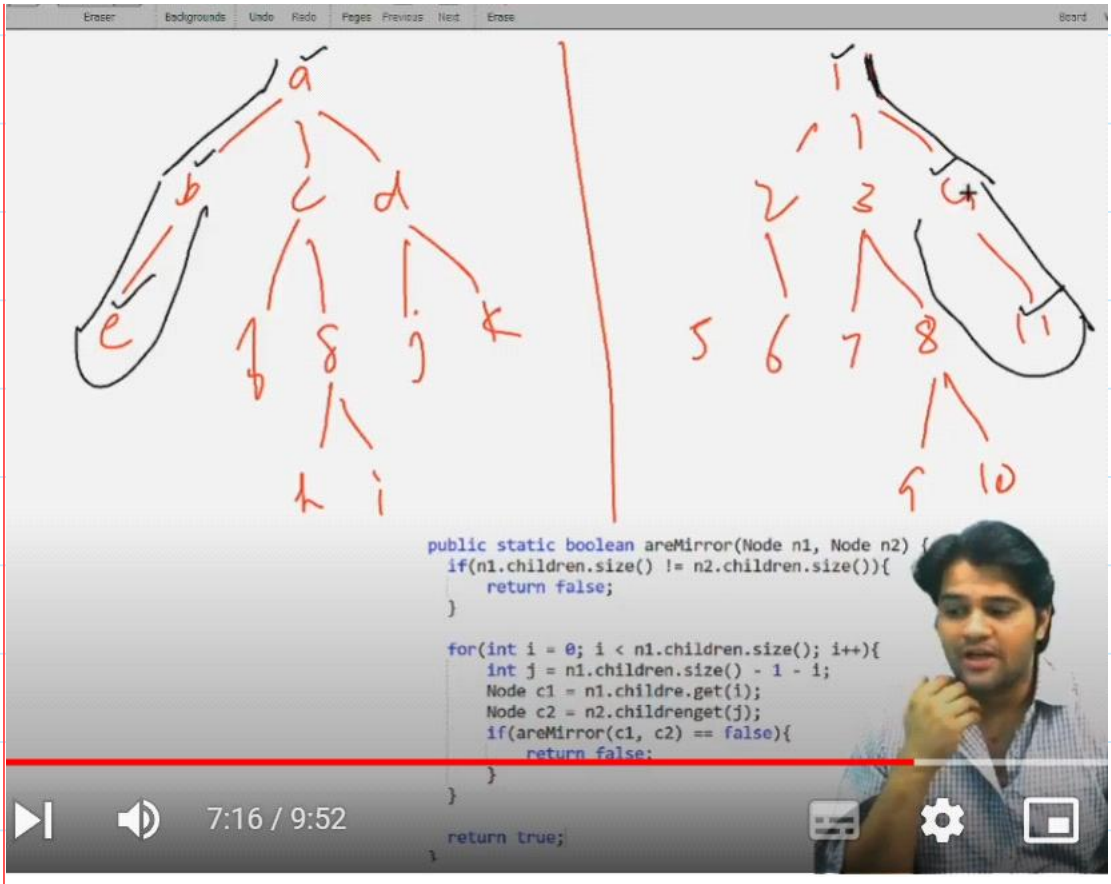
Aur iska akhri child, dusre ke pehle child ki mirror image honi chahie

So this question is very similar to areSimilar() but here inside the loop check one from beginning and another one from end.

```cpp
bool areMirror(Node* n1, Node* n2)
{
    if(n1->children.size() != n2->children.size())
    {
        return false;
    }

    for(int i=0; i< n1->children.size(); i++)
    {
        int j= n1->children.size()-1 - i;
        Node* c1 = n1->children[i];
        Node* c2 = n2->children[j];

        if(areMirror(c1, c2) == false)
            return false;

    }
    return true;
}
```

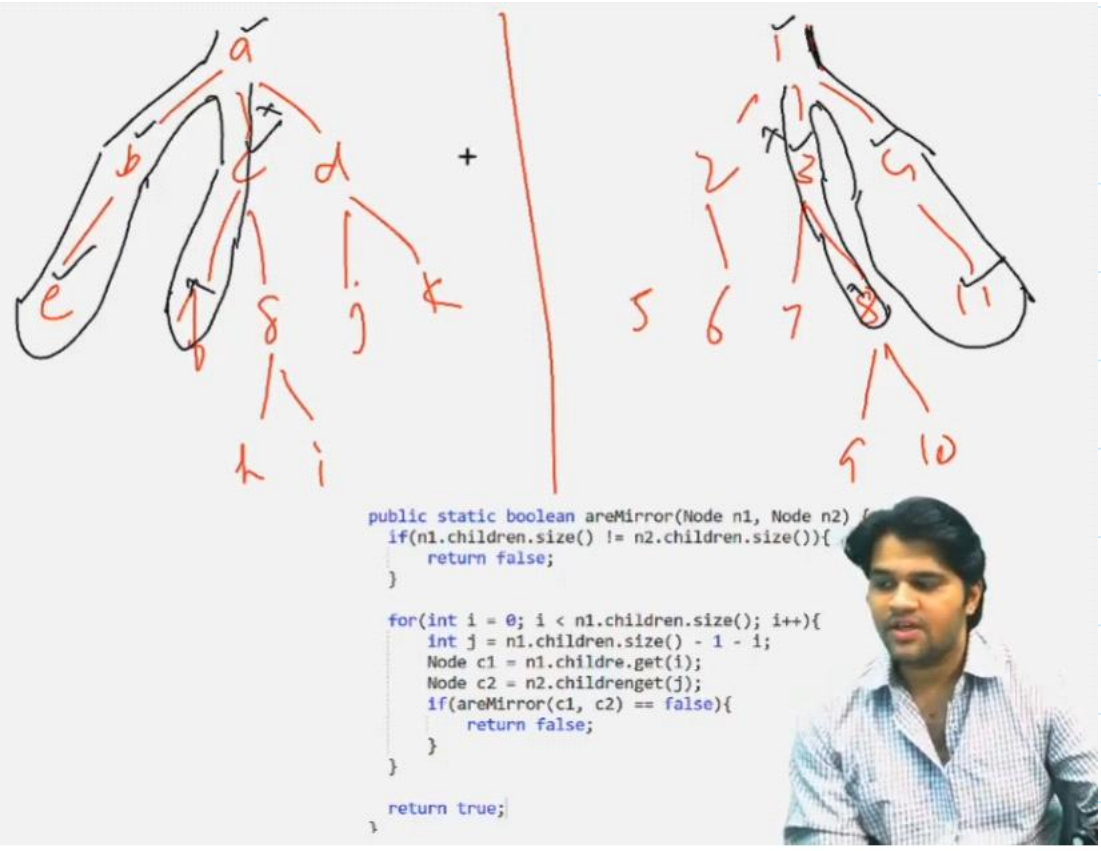
Now let's do the dry run of the code





Abhi apne dimag me b aur 4 ban jaye, jaha euler path pahuch jaye, hume wahi banke sochna chahie

That's how recursion becomes clear

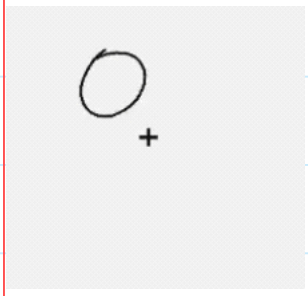


Jaise hi false ayega, wo seedha loop ko abort karke ghar ko bhagega

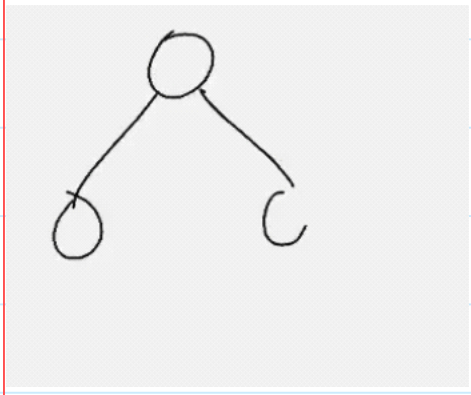
Is generic tree symmetric

09 July 2022 17:29

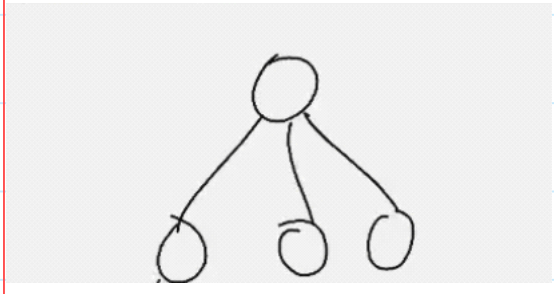
Hands are not symmetric
But face is symmetric.



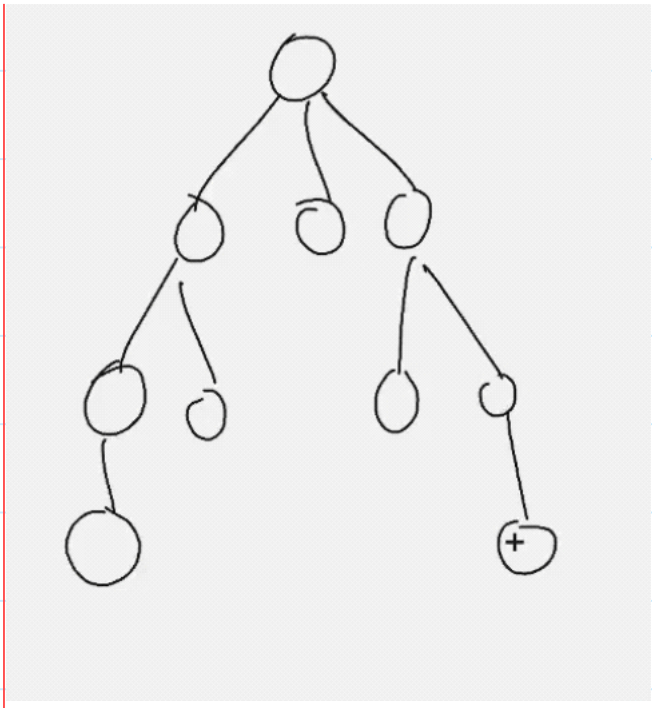
Single node is symmetric



This is also symmetric



This is also symmetric

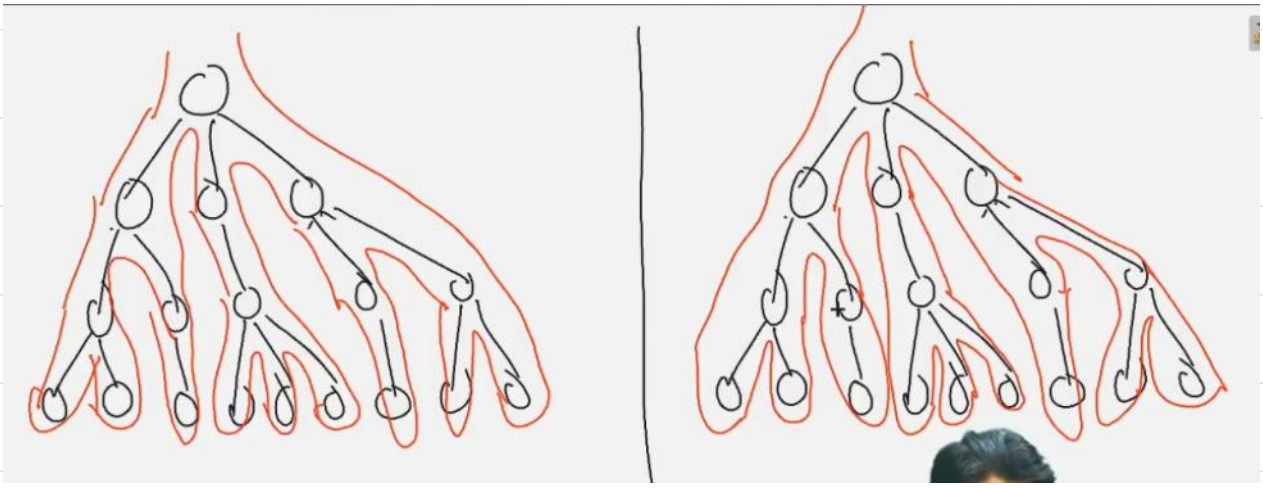


This is also symmetric

Untitled

09 July 2022 17:34

Symmetric tree are mirror image of itself in terms of shape, not in terms of data.



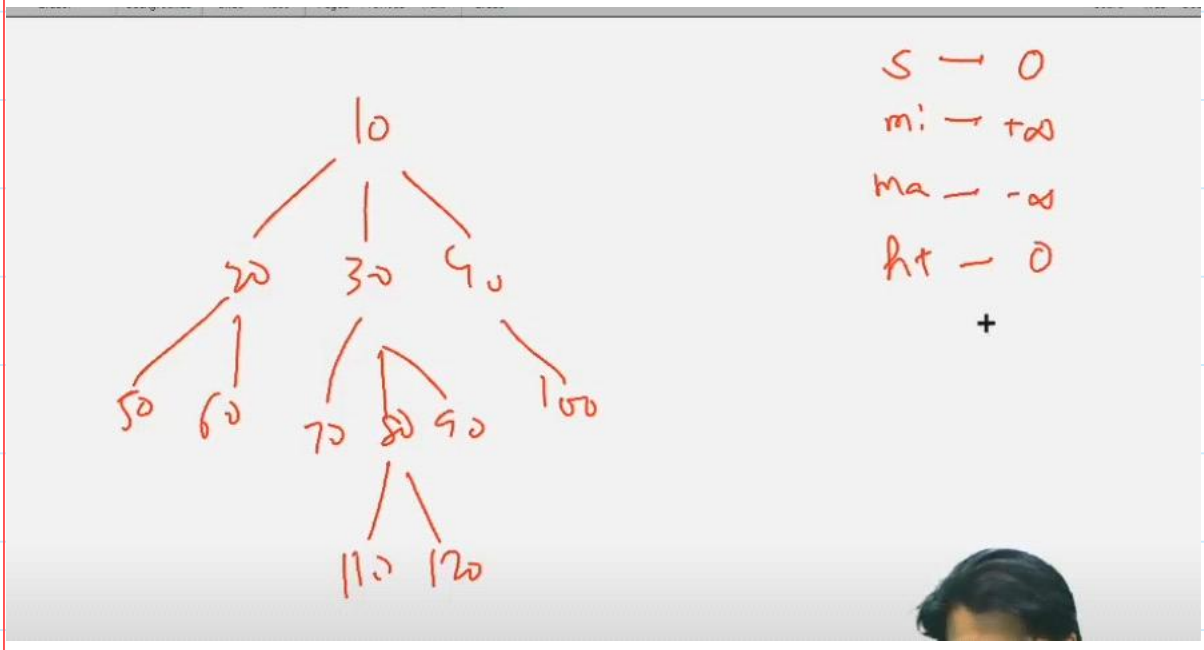
We will use areMirror() code

```cpp
bool areMirror(Node* n1, Node* n2)
{
    if(n1->children.size() != n2->children.size())
    {
        return false;
    }

    for(int i=0; i< n1->children.size(); i++)
    {
        int j= n1->children.size()-1 - i;
        Node* c1 = n1->children[i];
        Node* c2 = n2->children[j];

        if(areMirror(c1, c2) == false)
            return false;

    }
    return true;
}

bool isSymmetric(Node* node)
{
    return areMirror(node, node);
}
```

Generic tree multisolver.

09 July 2022 17:41

Size, maximum , height of a generic tree has been solved using recursion.



A function will help us to traverse in the tree and we will make changes to the data member i.e. size, min, max, ht during the traversal.

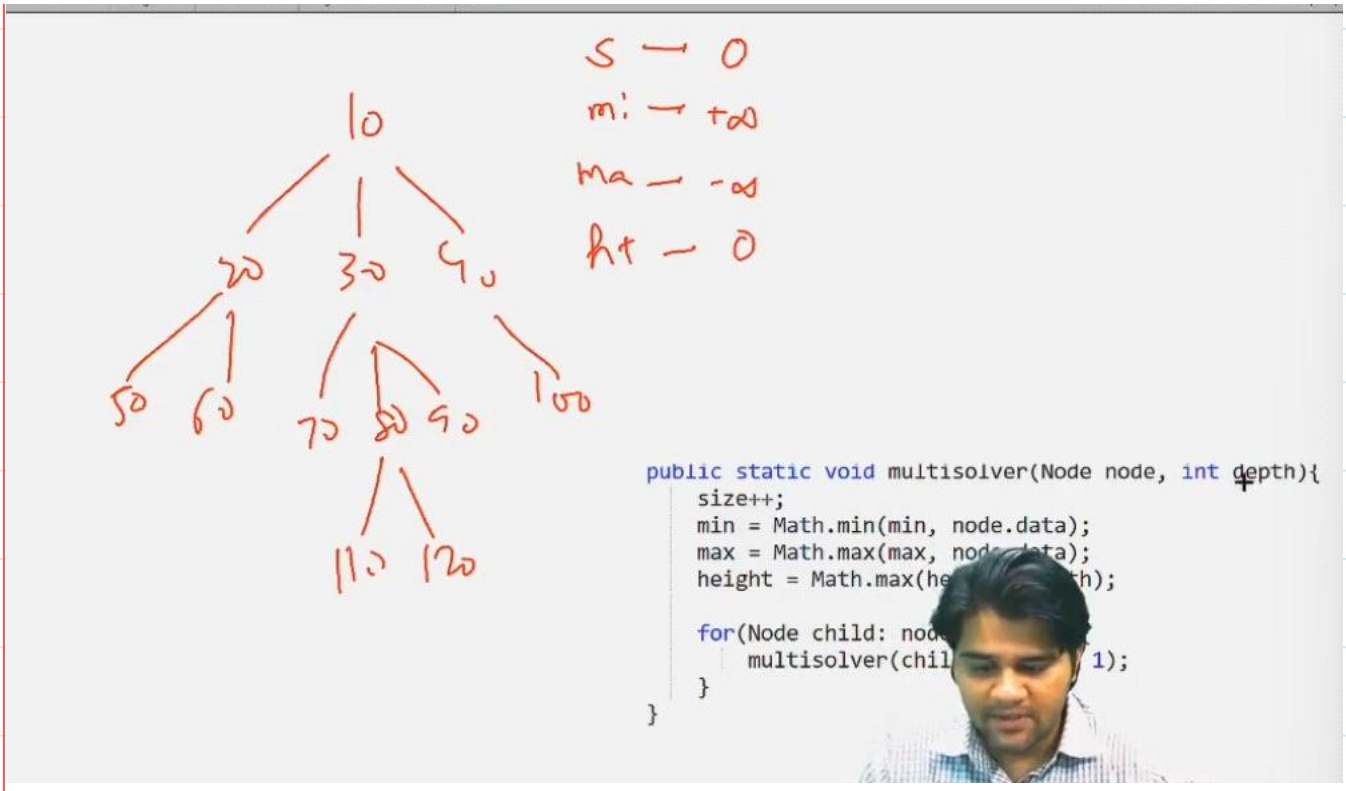
```
static int size;
static int mini;
static int maxi;
static int ht;
void multiSolver(Node* node, int depth)
{
    size++;
    mini= min(mini, node->data);
    maxi= max(maxi, node->data);

    ht= max(ht, depth);

    for(Node* child: node->children){
        multiSolver(child, depth+1);
    }


}
```

```
127     multiSolver(root,0);
128     levelOrder(root);
129     cout<<endl<<"size "<< size<<endl;
130     cout<<"min "<< mini<< endl;
131     cout<<"max "<< maxi<< endl;
132     cout<<"ht "<<ht<<endl;
133     return 0;
134 }
```

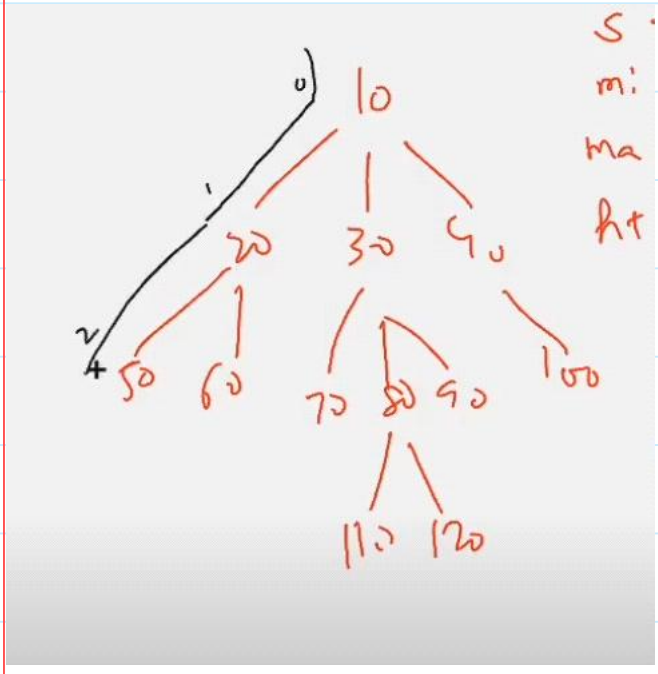
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
[Running] cd "c:\Users\ankit\Documents\Trees\"
"c:\Users\ankit\Documents\Trees\"gold
10
20 30 40
50 60 70 80 90 100
110 120
size 12
min 10
max 120
ht 3
```

Now do the dry run

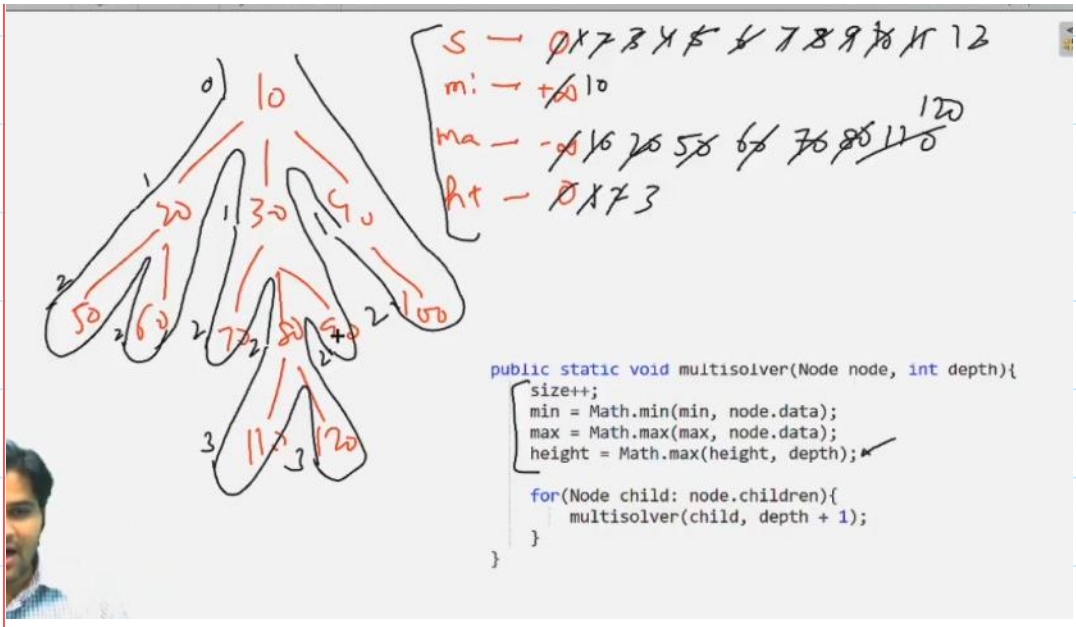


Recursion to euler me behti hai



Black one is the depth which has been passed, basically the level

Euler represent karta hai ki stack me kya state rahi hogi.



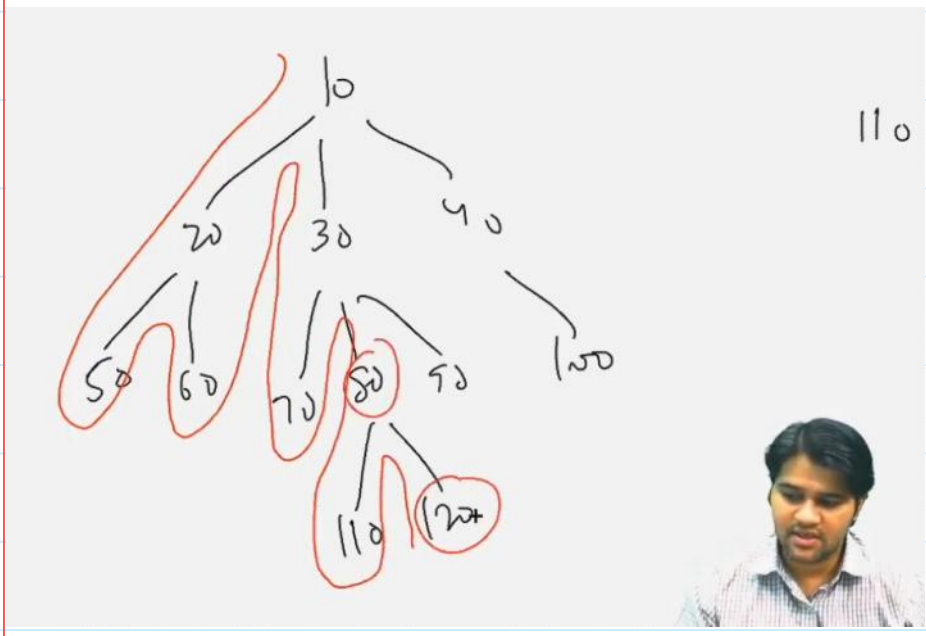
Is strategy ko kehte hai, traverse and change.

Predecessor and successor of an element

09 July 2022 17:57

You are given a tree and value

So in preorder find it's predecessor and successor



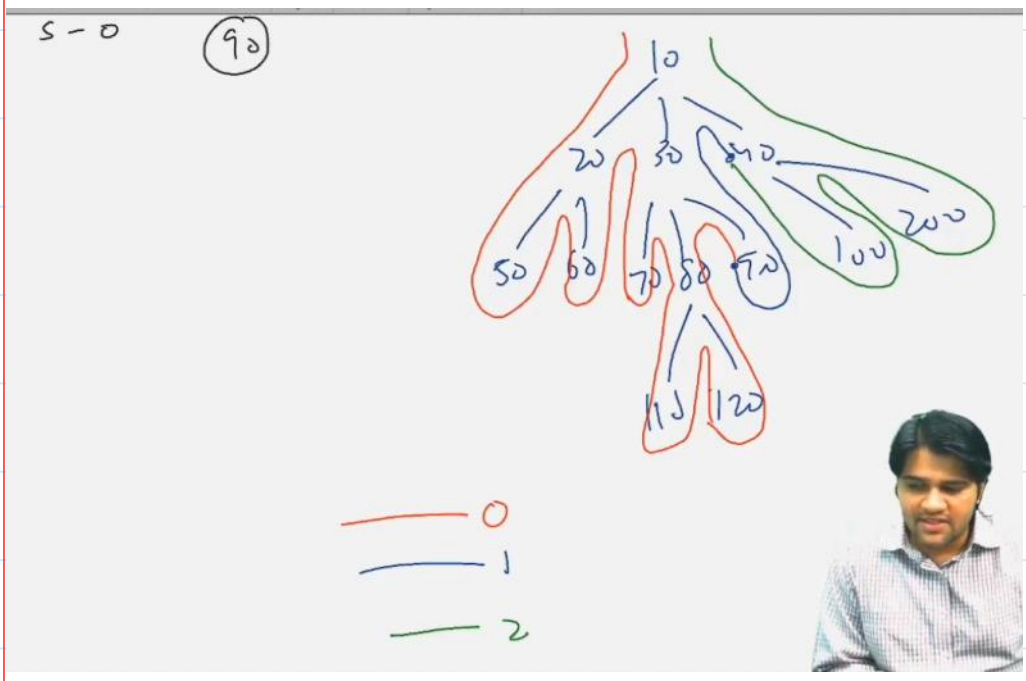
For 110, predecessor is 80 and successor is 120

Use strategy: Traverse and change.

Find predecessor and successor of 90.

Trick is to use state variable and keep it 0.

Jab tak 90 mil nahi jata, tab tak state variable 0 hi rhega
90 pe 1 hoga
Aur usse agle bande pe, state ko 1 se 2 ho jayega.

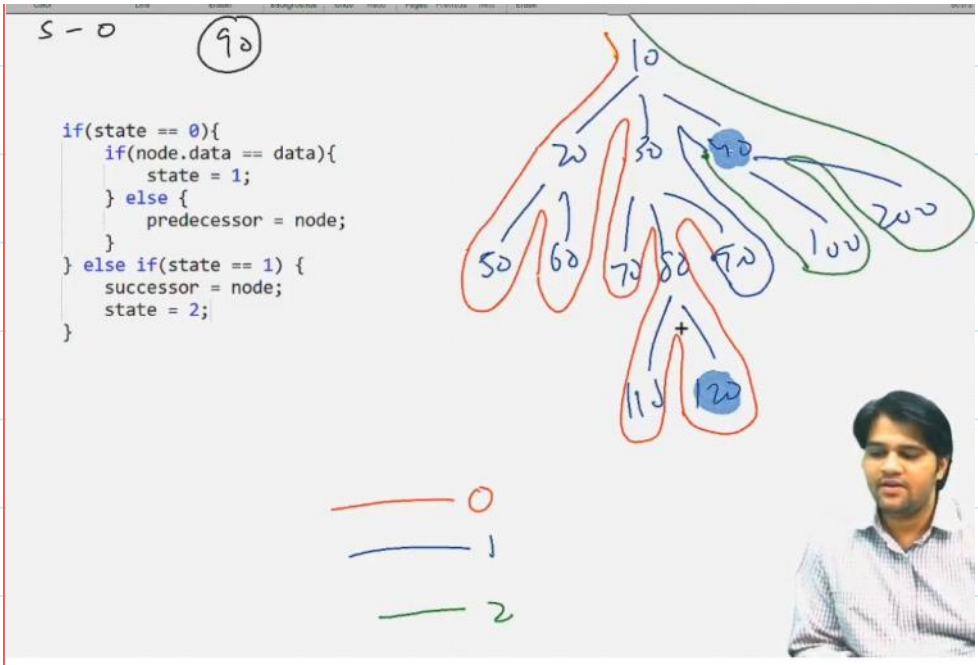


State 0 me jo akhri banda hai, wo predecessor hai

State 2 me jo pehla banda hai, wo successor hai

```
static Node* predecessor;
static Node* successor;
static int state;
void predecessorAndSuccessor(Node* node, int data)
{

    if(state ==0){
        if(node->data ==data){
            state=1;
        }
        else
        {
            predecessor =node;
        }
    }
    else if(state ==1){
        successor = node;
        state = 2;
    }


    for(Node* child: node->children){
        predecessorAndSuccessor(child, data);
    }
}
```

Do the dry run of this strategy.

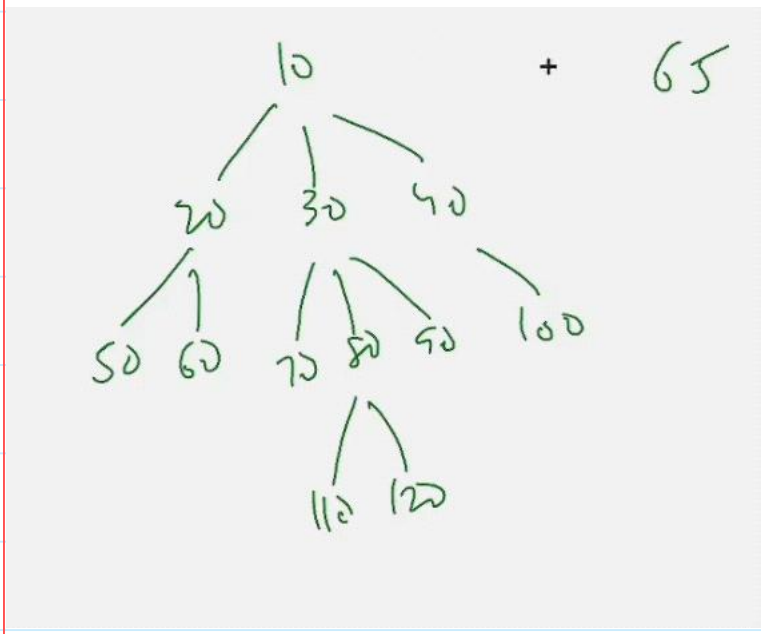

s - 0
90
if(state == 0){
if(node.data == data){
state = 1;
} else {
predecessor = node;
}
} else if(state == 1) {
successor = node;
state = 2;
}
10
20
50
70
50
60
70
80
90
100
200
110
120
0
1
2

Untitled

09 July 2022 18:20

Ceil and floor

Ceil: apse just bari value



65 se bare walo me sabse chota

Which is 70.

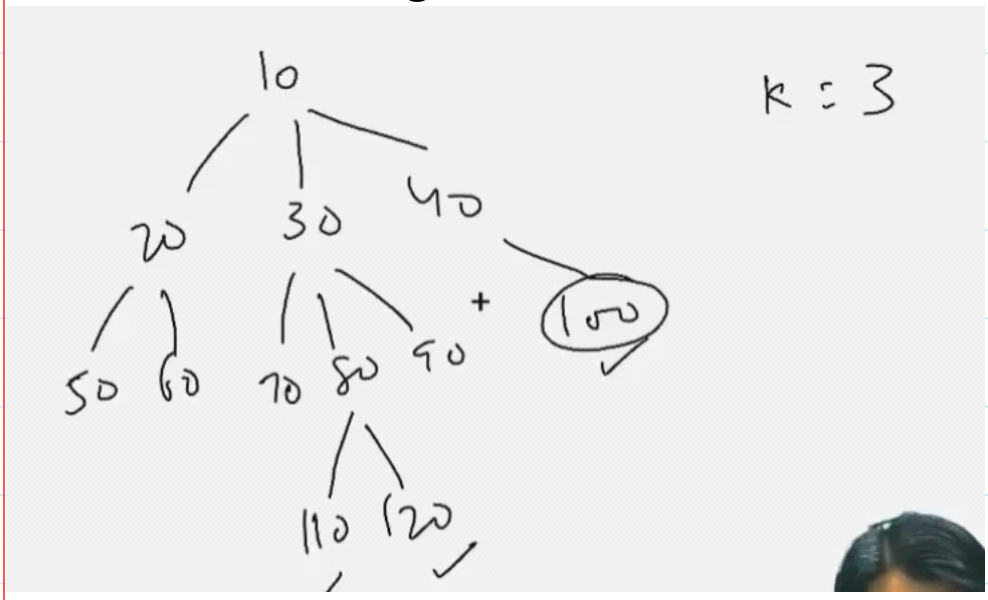
Floor: choto me sabse bara
Apse just choti value

```
}
static int ceili;
static int floori;

void ceilAndFloor(Node* node, int data){

    if(node->data< data)
    {
        floori= max(floori, node->data);
    }
    else
    {
        ceili=min(ceili, node->data);
    }
    for(Node* child: node->children){
        ceilAndFloor(child, data);
    }
}

int main()
{
    vector<int> arr;
    arr.assign({10, 20, 50, -1, 60, -1, -1, 30, 70, -1, 80, 110, -1

    int n= arr.size();
    Node* root=NULL;
    root=constructor(arr,n);
    levelOrder(root);
    ceili = INT_MAX;
    floori= INT_MIN;
    ceilAndFloor(root, 65);
    cout<<endl;

    cout<<"ceil of 65 "<<ceili<<endl;
    cout<<"floor of 65 "<<floori<<endl;

    return 0;
}
```

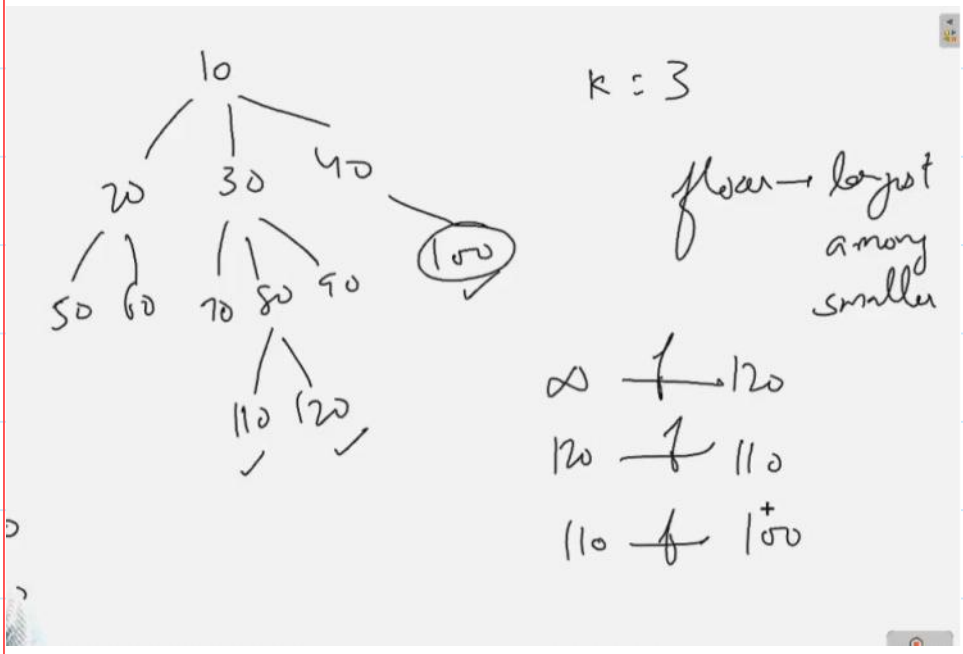
Kth largest element.

09 July 2022 18:32

If k=3, then 3rd largest value



We will use concept of ceil and floor



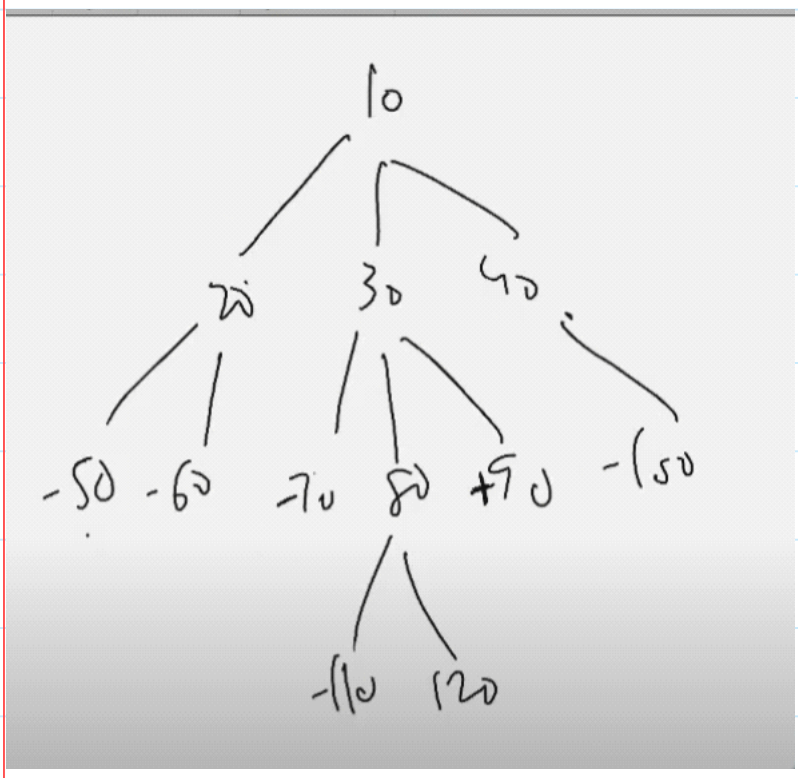
Isme k baar floor nikalunga, from infinity to the kth largest value

If you will find the floor of infinity for k no of times, then you will get the kth largest number.

```
}

public static int kthLargest(Node node, int k){
  floor = Integer.MIN_VALUE;
  int factor = Integer.MAX_VALUE;
  |
  for(int i = 0; i < k; i++){
      ceilAndFloor(node, factor); // will set floor
      factor = floor;
      floor = Integer.MIN_VALUE;
  }

  return factor;
}
```

Node with maximum subtree sum

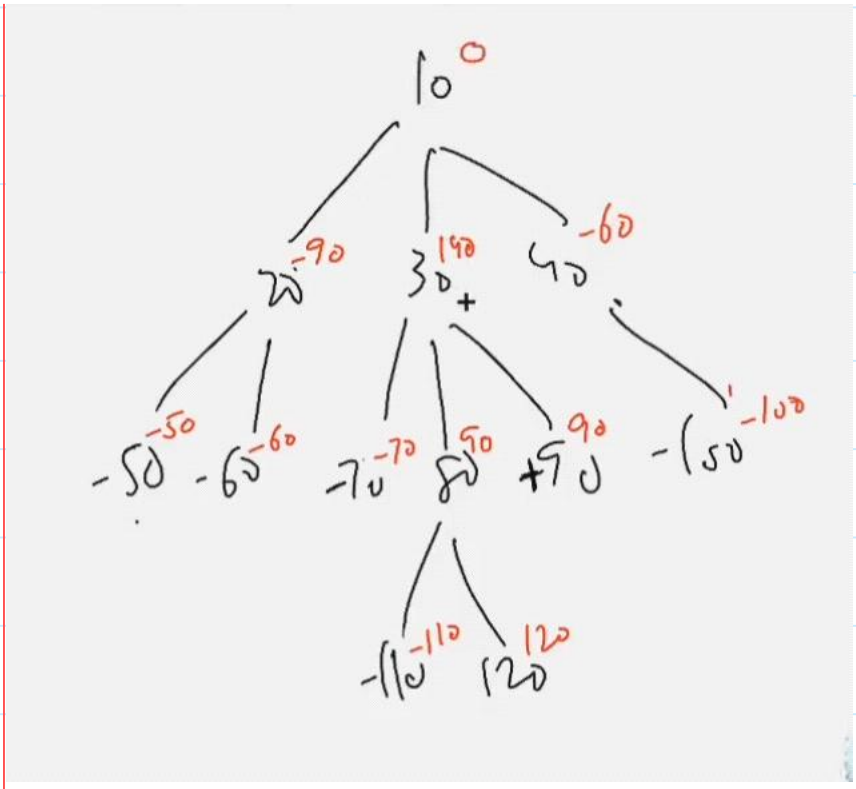
09 July 2022 18:38



Obviously negative numbers to honge hi

Hame wo node khojna hai, jiske subtree ka sum jyada hai

Har node pe calculate karna parega, which we are writing in red.



From here we can see the sum of all subtree

So answer is 30 wala node with the sum = 140

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/MaxSubtreeSum.java

Untitled

09 July 2022 18:40

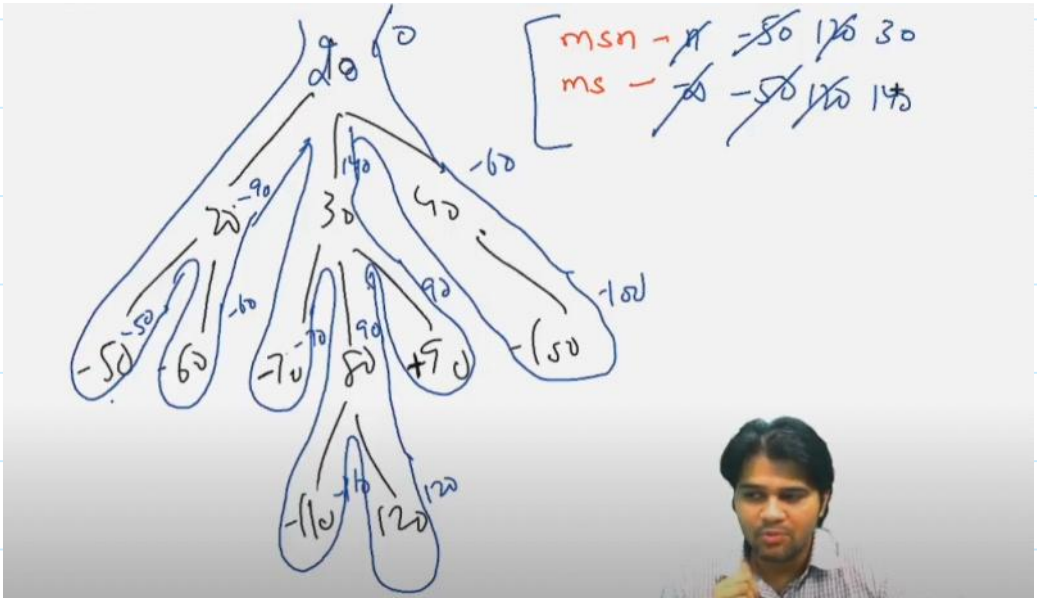
Travel and change strategy.

Size ke function me size hi pata karna hota hai, aur size hi return karte hai

Ht function me ht hi pata karna hota hai, aur ht his return karte hai

Lekin yaha pe hum return kuch aur karenge aur nikalna kuch aur hi hota hai

Return karenge sum, aur nikalna hame maxsum aur maxsum node.



Ye wala dry run samjhe aap

Recursion return kar rhi sum,
Lekin change aa rha hai msn and ms me.

Try writing the code by yourself;
Code on next page

Untitled

09 July 2022 18:59

```
static int msn=0;
static int ms=INT_MIN;

int travel(Node* node){
    int sum=0;

    for(Node* child: node->children){
        int csum = travel(child);
        sum+=csum;
    }
    sum+=node->data;

    if(sum> ms){
        msn=node->data;
        ms=sum;
    }
    return sum;
}
```

Ye us type of questions ko represent karta hai, jaha pe hum return kuch aur karate hai , aur change kuch aur karte hai

Ho rha hai yaha bhi travel and change

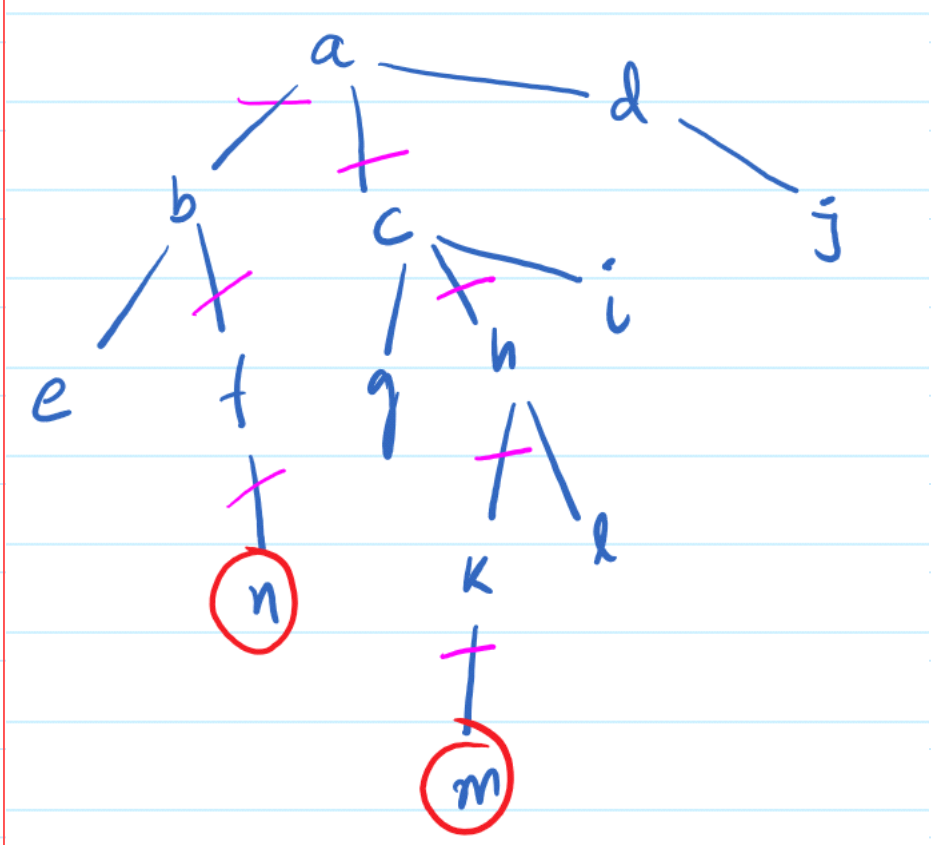
- Return something else, and calculate something else

Ye ek additional tool hai, travel and change karo, lekin return kuch aur karo, aur calculate kuch aur

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/MaxSubtreeSum.java

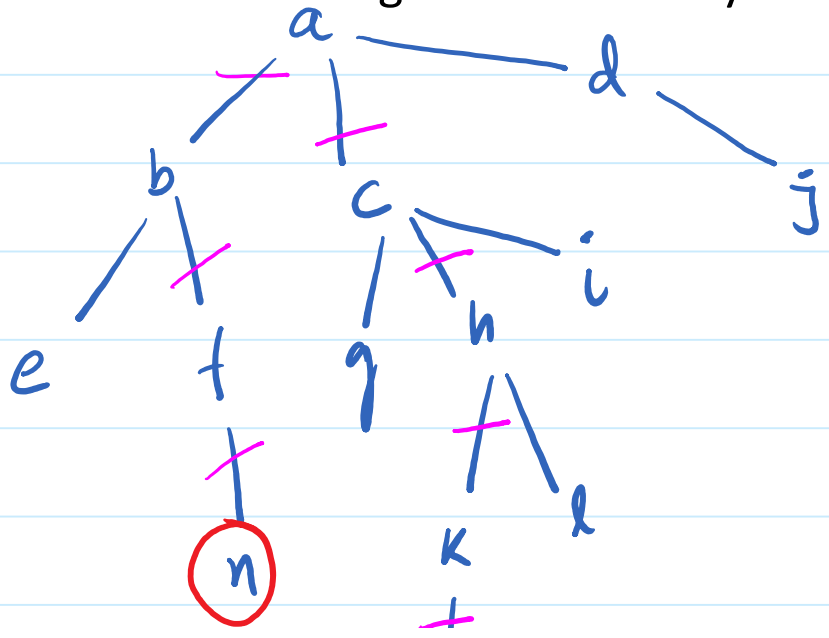
09 July 2022 18:59

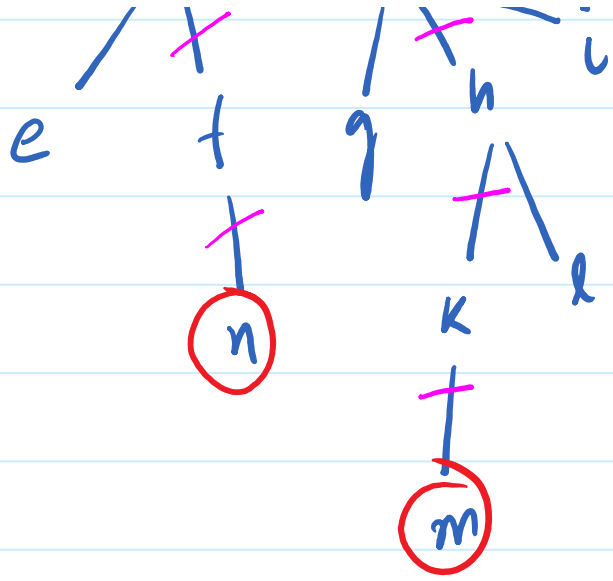
Diameter of a generic tree



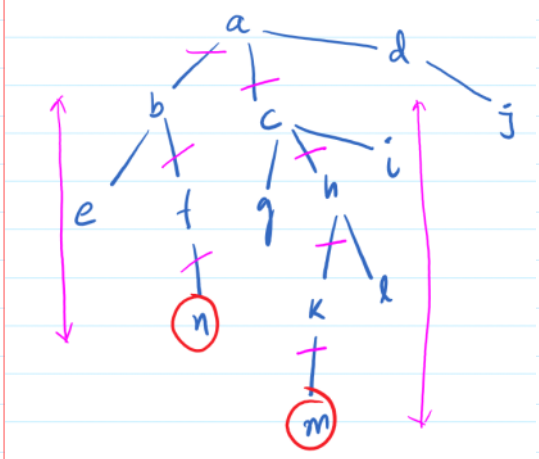
M se n tak 7 edges hai

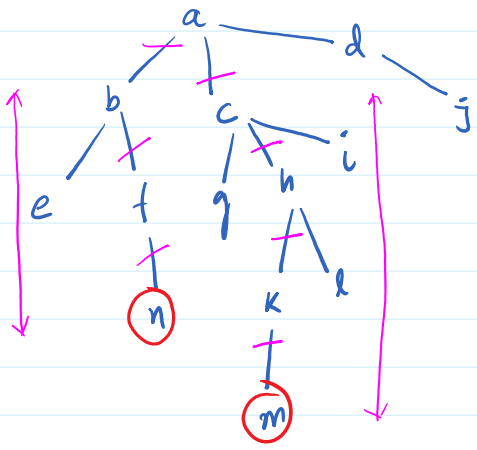
Maximum no of edges between any two node of the tree




e
f
g
h
i
k
l
n
m

09 July 2022 19:23



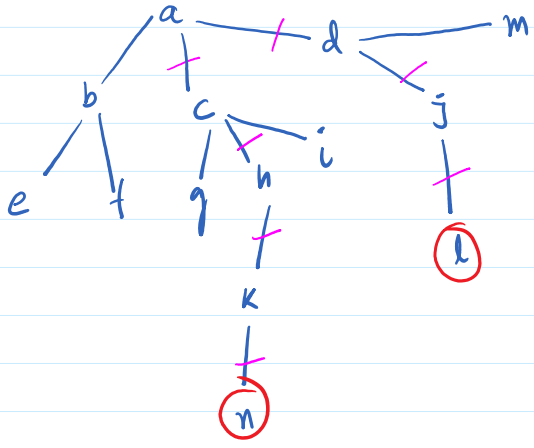


Tum aisa soch sakte ho:
Ye jo node "a" hai, uske sare child consider karo, usme sabse jyada ht wala aur second largest ht wala find karo

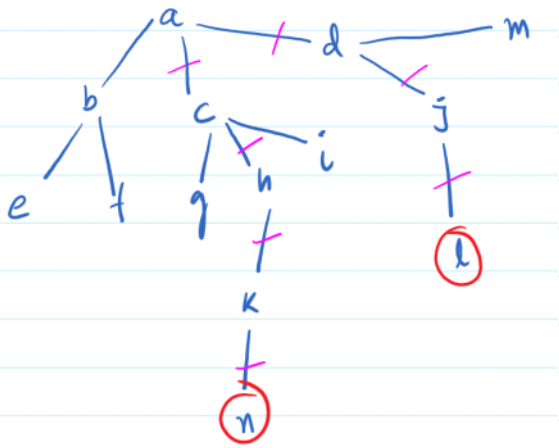
Deepest aur second deepest khojo

-> c hai jiska sabse deepest child hai =3
-> b hai jiska second deepest child hai = 2
Fir dono ko add karke usme do aur edges add kar do

Par aisa hamesha nahi hota hai

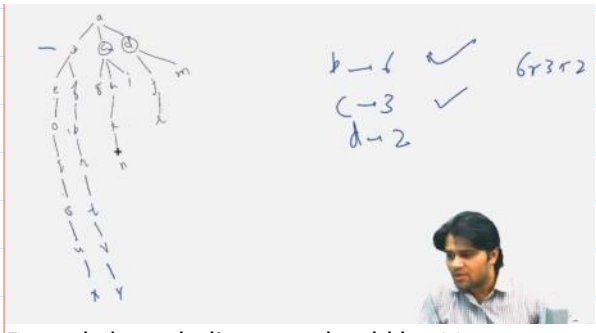


Yaha 7 diameter dikhega



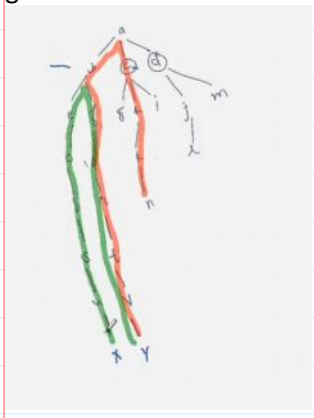
Lekin hamesha aisa nahi hoga

Lekin ab "a" ki terms me sochoge to



Formula ke sath diameter should be 11

Lekin there is something else available. This will be the answer in green



To jaruri nahi hai ki apka diameter "a" se guzre.
Aisa ho sakta hai ki bina "a" ke bhi diameter ban jaye

Ye baate yaad rakhna ki diameter root se hamesha nahi guzarega

Return kuch aur karte hai, calculate kuch aur karenge.

```
static int dia = 0;
static int calculateDiaReturnHeight(Node node){
    int dch = -1;
    int sdch = -1;

    for(Node child: node.children){
        int ch = calculateDiaReturnHeight(child);
        if(ch > dch){
            sdch = dch;
            dch = ch;
        } else if(ch > sdch){
            sdch = ch;
        }
    }
    int cand = dch + sdch + 2;

    if(cand > dia){
        dia = cand;
    }

    dch += 1;
    return dch;
}
```

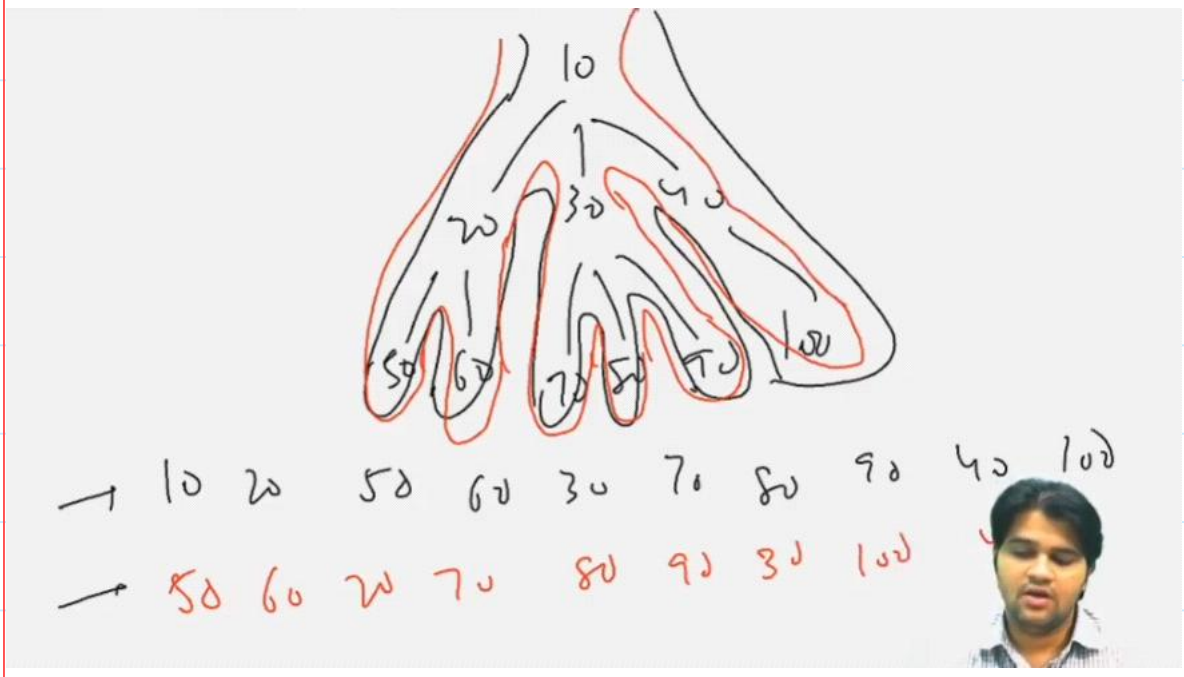
https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/TreeDiameterFinder.java

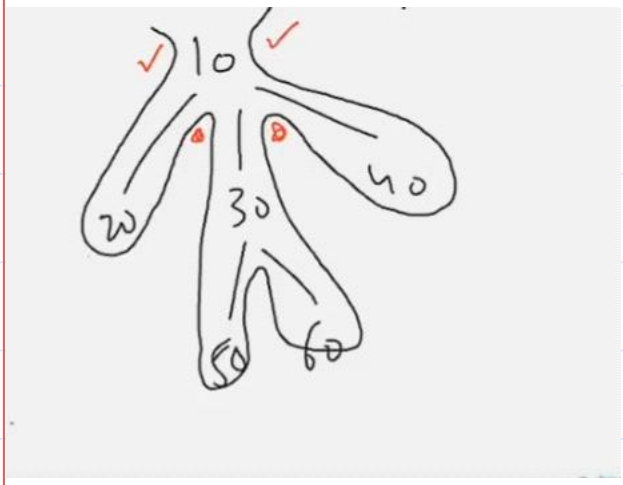
Untitled

11 July 2022 20:09

Iterative preorder and post order of generic tree.

https://github.com/AlgoMagnet/0or1/blob/main/GenericTree/TreeTraversal.java





10 chaar bar visit ho rha hai
Pehle preorder me visit hua

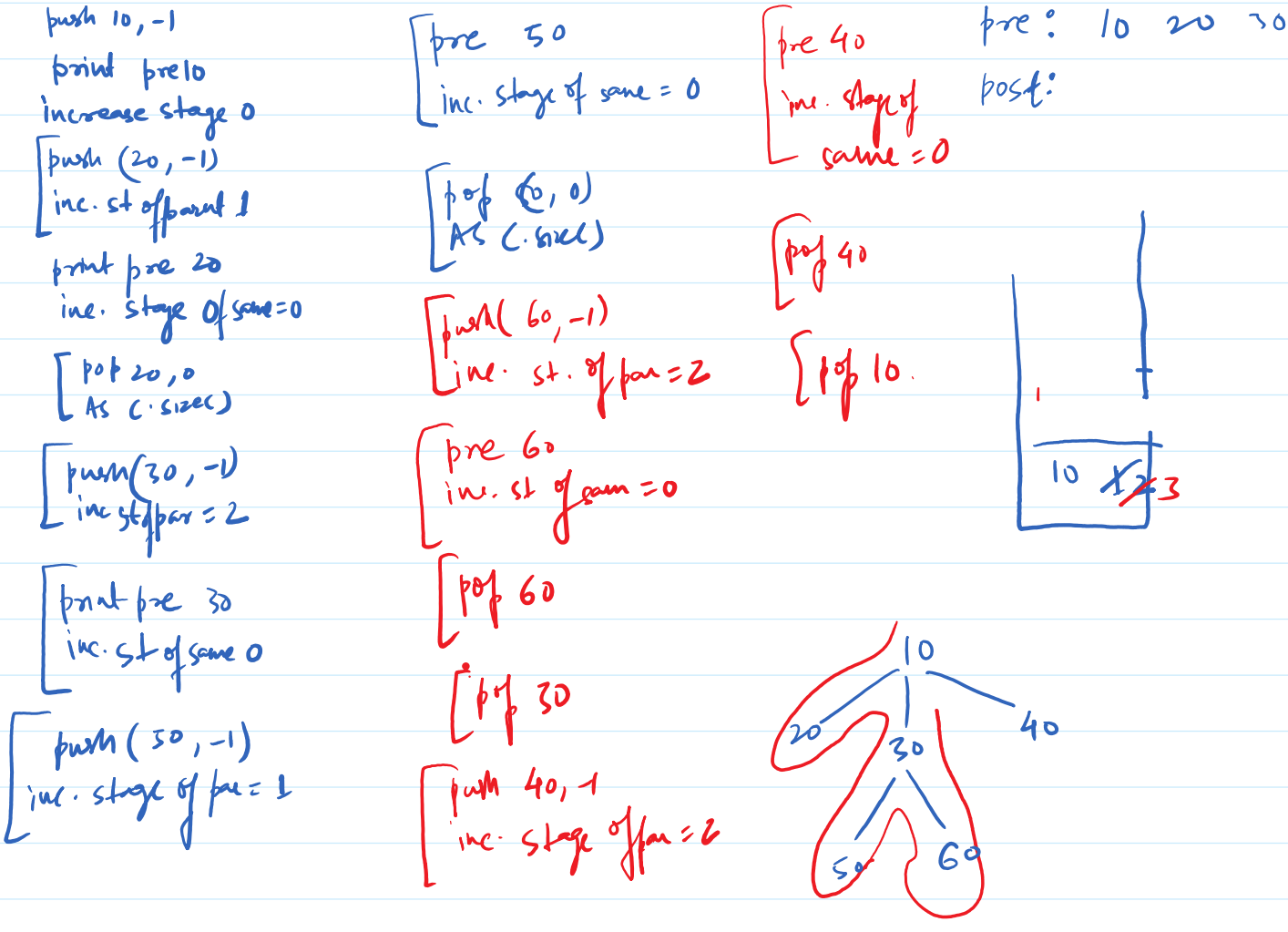
Fir first child visit hone ke baad visit hua
Fir second child visit hone ke baad visit hua
Fir third child visit hone ke baad, kyunki sare children visit ho gye to ab post order me visit ho gya

We will take advantage of this concept.

11 July 2022 20:27

Ek level ke baad apko sirf logic samjhna chahie, baar baar dekhna chahie, aur fir jab logic ache se samajh me aa jaye to code karne ki kosis karni chahie.

Agar code nahi bhi kar paye aur samay ho jindagi me to kosis karte rehna chahie aur logic and algorithm ko aur clearly samajna chahie

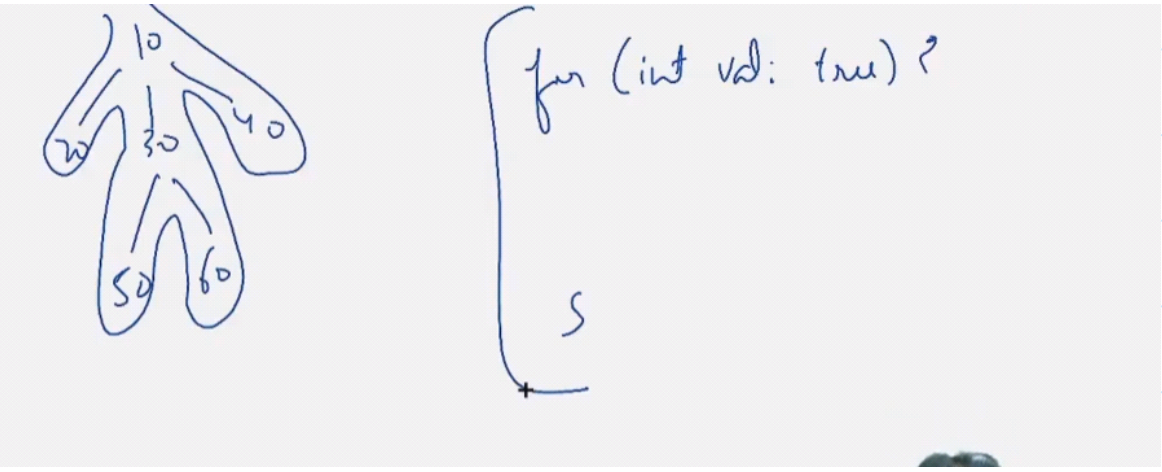


Mai chahta hu ki jo bhi mujhe ata hai, wo apko shikha paun.

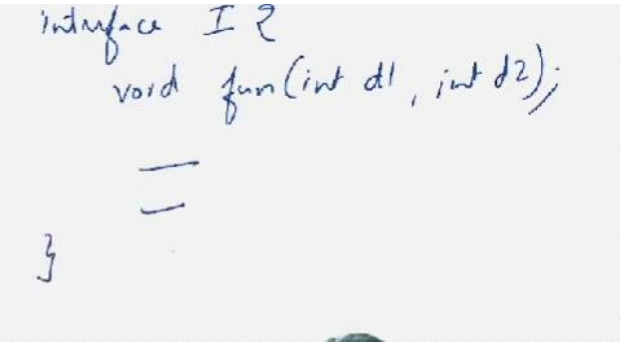
Untitled

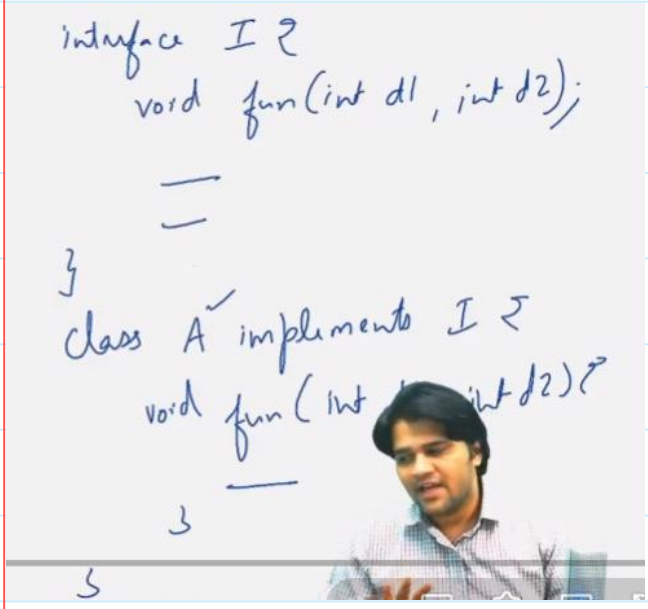
29 November 2025 07:52

Iterable and iterator



Interfaces to define signature of function





💡 What are Interfaces in Java?

An **interface** in Java is a blueprint of a class. It is a completely **abstract** type that is used to specify a contract for a class that implements it.

It defines a set of **methods** that a class must implement (or provide concrete definitions for) if it chooses to "implement" that interface. Think of it as a **protocol** or a **standard** that any implementing class must follow.

🔑 Key Characteristics

- **Abstract by Default:** Since Java 8, an interface can contain both abstract and concrete methods, but it fundamentally remains an abstract concept.
- **Method Signatures:** Traditionally, an interface contained only **method signatures** (method names, parameters, and return types) without any body.
- **Multiple Inheritance:** A class can implement multiple interfaces, which is how Java achieves a form of **multiple inheritance of behavior** (but not state), overcoming the restriction of inheriting from only one class.
- `implements` **Keyword:** A class uses the keyword `implements` to adopt an interface.
- `interface` **Keyword:** Interfaces are declared using the `interface` keyword.

🪜 Example and Use

1. Defining the Interface (The Contract)

Let's define a contract for anything that can be moved.

Java

```java
public interface Movable {
    // A constant field
    int MIN_SPEED = 1;

    // Abstract method (must be implemented)
    void move(int distance);

    // Default method (provides a default implementation)
    default void stop() {
        System.out.println("The object is stopping.");
    }
}
```

2. Implementing the Interface (Fulfilling the Contract)

A `Car` class must implement the `move` method, as required by the `Movable` interface.

Java

```java
public class Car implements Movable {
    @Override
    public void move(int distance) {
        System.out.println("Car moved " + distance + " units.");
    }

    // The Car class can optionally override the default stop() method,
    // but it doesn't have to.
}
```

3. Usage (Polymorphism)

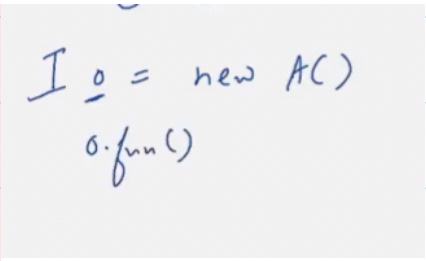
Interfaces are essential for **Polymorphism**, allowing you to treat objects of different types uniformly.

Java

```java
public class Main {
    public static void main(String[] args) {
        Car myCar = new Car();

        // Polymorphic reference:
        Movable object = myCar;

        object.move(50); // Calls Car's implementation
        object.stop();   // Calls the interface's default implementation

        // Output:
        // Car moved 50 units.
        // The object is stopping.
    }
}
```

How to create object of interface:

We use instance of class



❌ No, you cannot directly create an object (instance) of an interface in Java.

🔴 The Reason: Interfaces are Abstract

An interface is a purely **abstract** type. It defines a **contract** (a set of abstract methods) but provides no actual implementation for most of those methods (before Java 8, it provided *zero* implementation).

Since an interface specifies **what** an object should do but not **how** it does it, the Java Virtual Machine (JVM) cannot allocate memory and construct a fully functional object from it. You must rely on a concrete class to provide the necessary implementation.

🔑 Interface and Object Creation

While you can't use `new InterfaceName()`, interfaces are integral to object-oriented programming through two primary mechanisms:

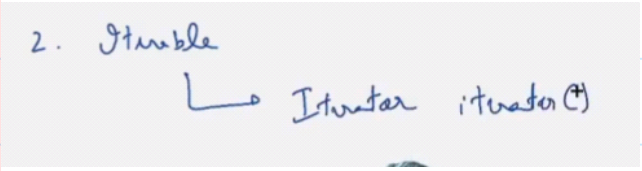
1. Implementation

You must create a **concrete class** that `implements` the interface and provides a body for all its abstract methods. You then create an object of that **implementing class**:

Java

```java
public interface Drawable {
    void draw();
}

public class Circle implements Drawable {
    @Override
    public void draw() {
        System.out.println("Drawing a circle.");
    }
}

// Creating an object of the implementing class
Drawable obj = new Circle(); // ✅ Valid
obj.draw();
```

Iterable interface?



💡 What is the `Iterable` Interface in Java?

The `Iterable` **interface** is a fundamental interface in the Java Collections Framework. Its primary purpose is to enable an object to be the target of the **"for-each loop"** (also known as the enhanced for loop).

If a class implements the `Iterable` interface, it signifies that the object's components can be **iterated** (or traversed) over, one by one.

🔧 Structure and Method

The `Iterable` interface is simple and has only one **abstract method** that all implementing classes must provide:

1. `iterator()`

   This method returns an instance of the `Iterator` interface.

   Java

   ```
   public interface Iterable<T> {
       Iterator<T> iterator();

       // Since Java 8, it also includes a default method:
       // default void forEach(Consumer<? super T> action) { ... }
   }
   ```

   - `T` stands for the type of elements that can be traversed.
   - The `Iterator` object returned is the actual workhorse; it is responsible for keeping track of the current position during the traversal.

Name of returned object is iterator and function name is iterater.

```
public class Main {
    public static class GenericTree implements Iterable<Integer> {
        Node root;
        GenericTree(Node root){
            this.root = root;
        }

        public Iterator<Integer> iterator(){
            Iterator<Integer> obj = new GTPreorderIterator();
            return obj;
        }
    }

    public static class GTPreorderIterator implements Iterator<Integer> {

    }
```

5. Iterator
   - hasNext() [whether new value is available]
   - next() [returns CV, sets up for next value]

```
        GenericTree gt = new GenericTree(root);
        for(int val: gt){ // syntactical sugar dependent on Iterable
            System.out.println(val);
        }

        Iterator<Integer> gti = gt.iterator();
        while(gti.hasNext() == true){
            System.out.println(gti.next());
        }
    }
```

That's correct! The `Iterator` interface in Java is defined by three core methods, with `hasNext()` and `next()` being the two most crucial for traversal:

⚙️ Core Methods of the `Iterator` Interface

Method	Return Type	Description
`hasNext()`	`boolean`	Checks if the iteration has more elements. Returns `true` if calling `next()` will successfully return an element, and `false` if the end of the collection has been reached.
`next()`	`E` (Element type)	Returns the **next element** in the iteration and **advances the iterator's position** past that element. It typically throws a `NoSuchElementException` if called when `hasNext()` is false.