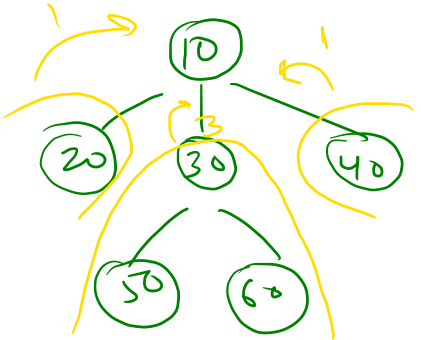
Generic TREE

Q1 Size of generic tree

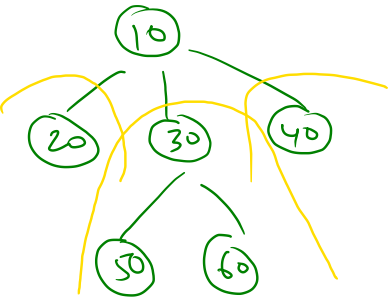



```java
public static int size(Node node){
  // write your code here
  if(node==null){
    return 0;
  }
  int size=0;
  for(Node child:node.children){
    size+=size(child);
  }
  return size+1;
}
```

1 + 1 + 3 = size = 5

return size+1 = 6

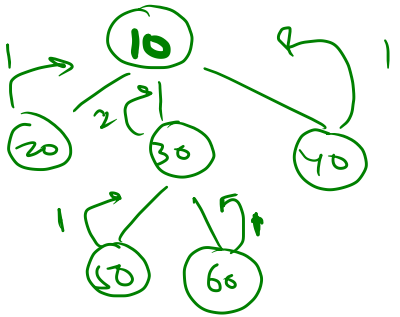
MAXIMUM IN A GENERIC TREE




```java
public static int max(Node node) {
  // write your code here
  if(node==null){
      return 0;
  }
  int max = Integer.MIN_VALUE;
  for(Node child:node.children){
      max = Math.max(max,max(child));
  }
  return Math.max(max,node.data);
}
```

max = 20
max = 60
max = 40
→ max = 60 → max = Math.max(10, 60)
= 60

Height of Generic Tree

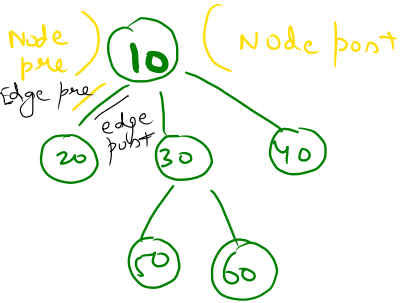



```java
public static int height(Node node) {
  // write your code here
  if(node==null){
    return 0;
  }
  int ht = -1 ;
  for(Node child:node.children){
    ht = Math.max(ht,height(child));
  }
  return ht+1;
}
```

ht = 1
ht = 2
ht = 1

→ ht = 2

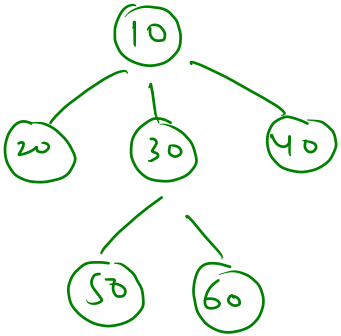
TRAVERSALS (pre and post order)

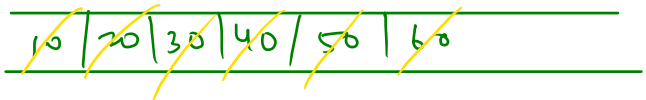



```
Node Pre 10
Edge Pre 10--20
Node Pre 20
Node Post 20
Edge Post 10--20
Edge Pre 10--30
Node Pre 30
Edge Pre 30--50
Node Pre 50
Node Post 50
Edge Post 30--50
Edge Pre 30--60
Node Pre 60
Node Post 60
Edge Post 30--60
Node Post 30
Edge Post 10--30
Edge Pre 10--40
Node Pre 40
Node Post 40
Edge Post 10--40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

```java
public static void traversals(Node node){
  // write your code here
  if(node==null){
    return;
  }
  System.out.println("Node Pre "+node.data);
  for(Node child:node.children){
    System.out.println("Edge Pre "+node.data+"--"+child.data);
    traversals(child);
    System.out.println("Edge Post "+node.data+"--"+child.data);
  }
  System.out.println("Node Post "+node.data);
}
```

Level ORDER TRAVERSAL IN A GENERIC TREE

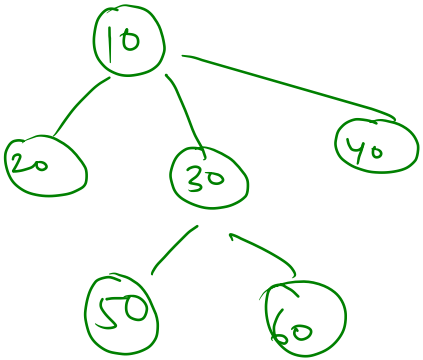




o/p → 10 20 30 40 50 60

```java
public static void levelOrder(Node node){
  // write your code here
  if(node==null){
    return;
  }
  Queue<Node> q = new ArrayDeque<>();
  q.add(node);
  while(q.size()>0){
    Node rnode = q.remove();
    System.out.print(rnode.data+" ");
    for(Node child:rnode.children){
      q.add(child);
    }
  }
  System.out.println(".");
}
```

LEVEL ORDER LINEWISE



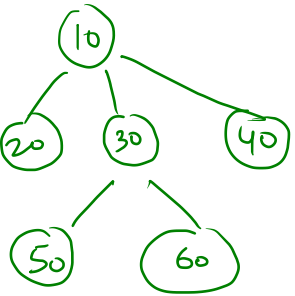
10	-1	20	30	40	-1	50	60	-1

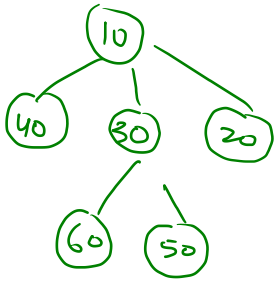
```java
public static void levelOrderLinewise(Node node){
  // write your code here
  if(node==null){
    return;
  }
  Node one = new Node();
  one.data = -1;
  Queue<Node> q = new ArrayDeque<>();
  q.add(node);
  q.add(one);
  while(q.size()>0){
    Node rnode = q.remove();
    if(rnode.data == -1 && q.size()!=1){
      System.out.println();
      q.add(one);
    }else{
      System.out.print(rnode.data+" ");
    }
    for(Node child:rnode.children){
      q.add(child);
    }
    if(q.size()==1){
      return;
    }
  }
```

o/p

10
20 30 40
50 60

Mirror In Generic Tree

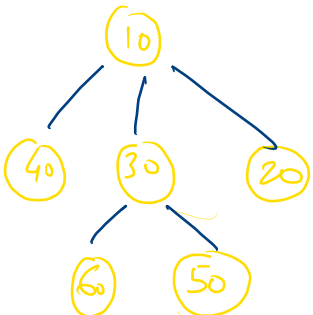




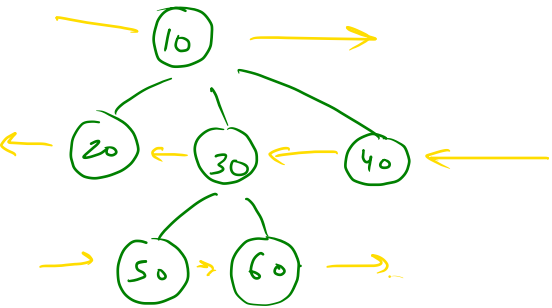

```java
public static void mirror(Node node){
  // write your code here
  // Collections.reverse(node.children);
  int l=0,r=node.children.size()-1;
  while(l<r){
    Node temp = node.children.get(l);
    node.children.set(l,node.children.get(r));
    node.children.set(r,temp);
    l++;r--;
  }
  for(Node child:node.children){
    mirror(child);
  }
}
```

Reverse

Reverse



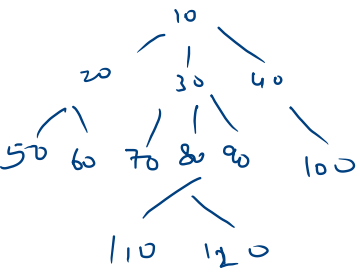
LEVEL ORDER ZIGZAG



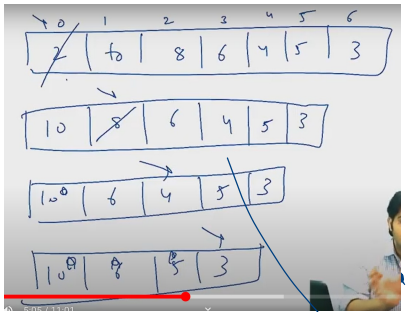

```java
public static void levelOrderLinewiseZZ(Node node){
  // write your code here
  if(node==null){
    return;
  }
  Stack<Node> ms = new Stack<>();
  Stack<Node> cs = new Stack<>();
  ms.add(node);
  int level = 1;
  while(ms.size()>0){
    Node rnode = ms.pop();
    System.out.print(rnode.data+" ");
    if(level%2==1){
      for(int i=0;i<rnode.children.size();i++){
        cs.push(rnode.children.get(i));
      }

    }else{
      for(int i=rnode.children.size()-1;i>=0;i--){
        cs.push(rnode.children.get(i));
      }
    }
    if(ms.size()==0){
      System.out.println();
      ms=cs;
      cs = new Stack<>();
      level++;
    }
```

REMOVE LEAF NODE




```
public static void removeLeaves(Node node) {
  // write your code here
  if(node==null){
    return;
  }
  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);
    }
  }
  for(Node child:node.children){
    removeLeaves(child);
  }
}
```

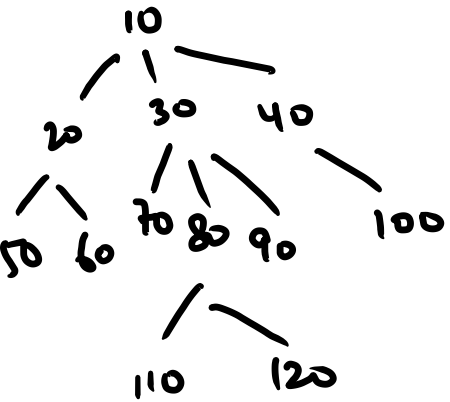


Step1. Remove leaf nodes
Step2. Traverse through child nodes

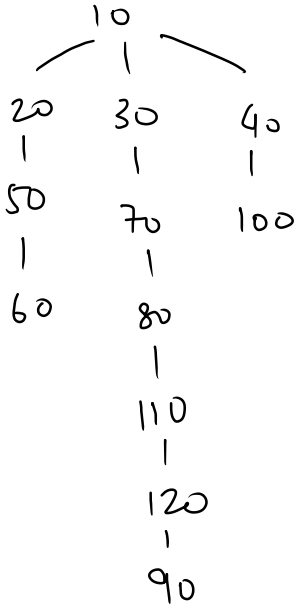
Loop piche se chalega jisse vo har value ko traverse kr sake and koi value miss na ho jaye.

Post order mein agr remove krenge to jo leaf nhi hai, vo bhi leaf jaise nazar aayenge
Isliye remove hmesha preorder mein hoga.

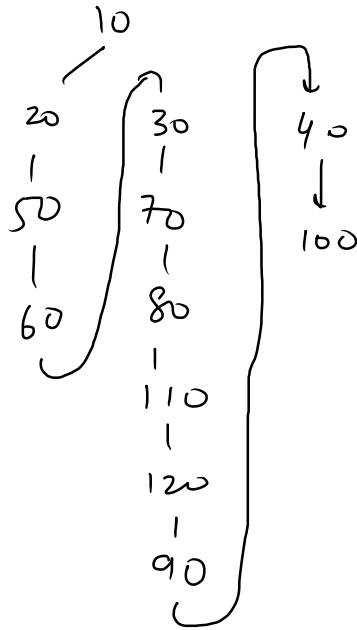
LINEARIZE A GENERIC TREE



=>



=>



```java
public static void linearize(Node node){
  // write your code here
  for(Node child:node.children){
    linearize(child);
  }
  while(node.children.size()>1){
    Node lc = node.children.remove(node.children.size()-1);
    Node slc = node.children.get(node.children.size()-1);
    Node slt = getTail(slc);
    slt.children.add(lc);
  }
}
public static Node getTail(Node node){
  while(node.children.size()==1){
    node = node.children.get(0);
  }
  return node;
}
```

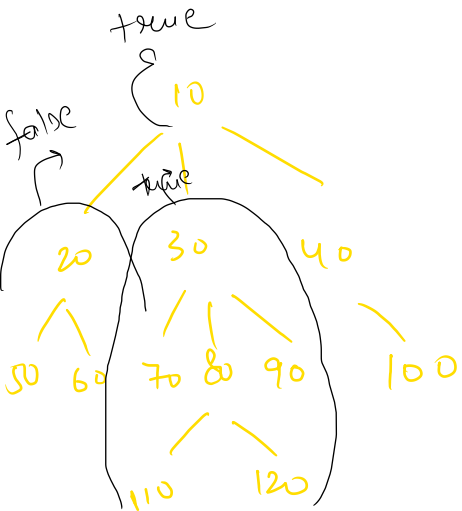
→ remove last child

→ get second last child

→ getting tail of second last child

→ joining second last tail to last child

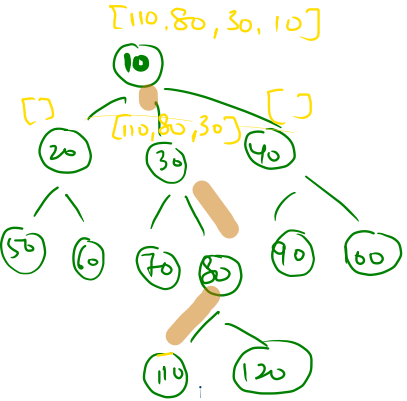
FIND IN GENERIC TREE



data = 110

```java
public static boolean find(Node node, int data) {
  // write your code here
  if(node==null){
    return false;
  }
  if(node.data==data){
    return true;
  }
  for(Node child:node.children){
    boolean res = find(child,data);
    if(res==true){
      return true;
    }
  }
  return false;
}
```

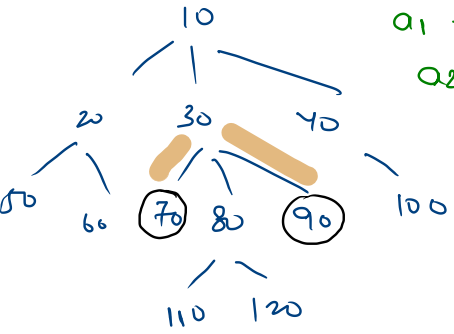
NODE TO ROOT PATH




```java
public static ArrayList<Integer> nodeToRootPath(Node node, int data){
    // write your code here
    if(node.data==data){
      ArrayList<Integer> arr = new ArrayList<Integer>();
      arr.add(node.data);
      return arr;
    }
    for(Node child:node.children){
      ArrayList<Integer> ptc = nodeToRootPath(child,data);
      if(ptc.size()>0){
        ptc.add(node.data);
        return ptc;
      }
    }
    return new ArrayList<>();
}
```

similar logic as of find in generic tree
when node matches the value of data, we will store that in arraylist and if the size is greater than 0, we will add its parent node to complete the path

DISTANCE BETWEEN TWO NODES



$a_1 = [70, 30, 10]$

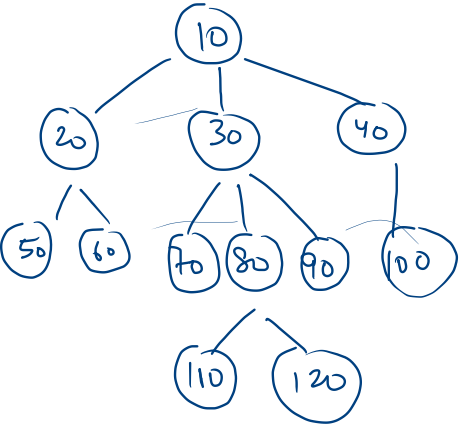
$a_2 = [90, 30, 10]$

lca = 30

```java
public static int distanceBetweenNodes(Node node, int d1, int d2){
  // write your code here
  ArrayList<Integer> a1 = nodeToRootPath(node,d1);
  ArrayList<Integer> a2 = nodeToRootPath(node,d2);
  int i = a1.size()-1;
  int j = a2.size()-1;
  while(i>=0 && j>=0){
    if(a1.get(i)!=a2.get(j)){
      break;
    }
    i--;j--;
  }
  return i+j+2;
}
```

firstly, we will calculate node to root path, then we will calculate lca, from that we will get the values of i and j and using that we can calculate the path,

LOWEST COMMON ANCESTOR



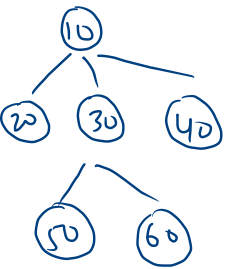
a1 = [110, 80, 30, 10]

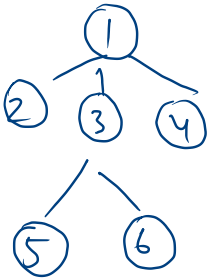
a2 = [80, 30, 10]

```java
public static int lca(Node node, int d1, int d2) {
  // write your code here
  ArrayList<Integer> a1 = nodeToRootPath(node,d1);
  ArrayList<Integer> a2 = nodeToRootPath(node,d2);
  int i=a1.size()-1;
  int j = a2.size()-1;
  while(i>=0 && j>=0){
    if(a1.get(i)!=a2.get(j)){
      break;
    }
    i--;j--;
  }
  i++;
  return a1.get(i);
}
```

we will traverse from end of arraylist and will decrease i and j simultaneously and whenever we found different value of a1[i] and a2[j], we will break the loop, increament value of i, to return the first common value in both the loops

ARE TREES SIMILAR IN SHAPE



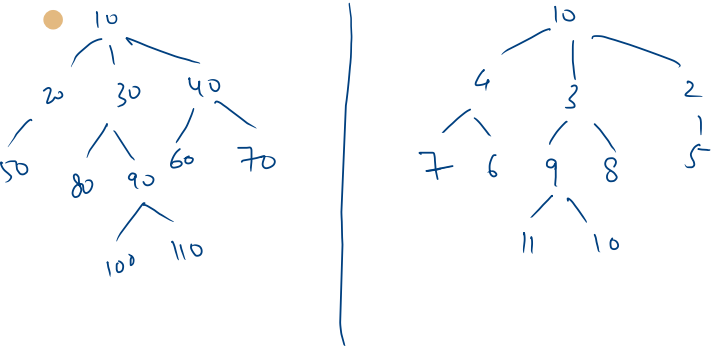



```java
public static boolean areSimilar(Node n1, Node n2) {
  // write your code here
  if(n1.children.size()!=n2.children.size()){
    return false;
  }
  for(int i=0;i<n1.children.size();i++){
    Node c1 = n1.children.get(i);
    Node c2 = n2.children.get(i);
    if(areSimilar(c1,c2)==false){
      return false;
    }
  }
  return true;
}
```

if we have to check if trees are similar or not, then we have to focus on it's children and not on value

So first we will check if children size of node 10 is equal to children size of node 1, and if they are equal, then we will check further on their children else return false

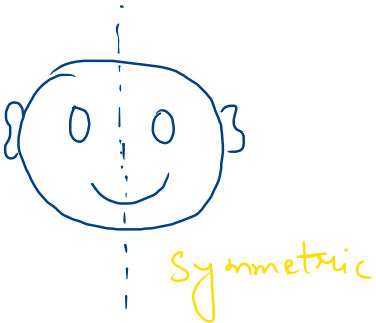
ARE TREES MIRROR IN SHAPE




```java
public static boolean areMirror(Node n1, Node n2) {
  // write your code here
  if(n1.children.size()!=n2.children.size()){
    return false;
  }
  int i = 0,j=n2.children.size()-1;
  while(i<=j){
    Node c1 = n1.children.get(i);
    Node c2 = n2.children.get(j);
    if(areMirror(c1,c2)==false){
      return false;
    }
    i++;j--;
  }
  return true;
}
```

we will check children size and compare it to calculate if the trees are mirror or not. In this we will traverse from i=0 and j = n2.children.size()-1 to check for mirror

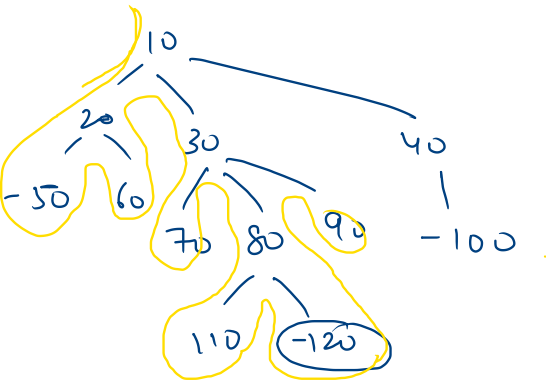
Is Generic Tree Symmetric




```java
public static boolean IsSymmetric(Node node) {
  // write your code here
  return areMirror(node,node);
}
```

So, for this question we will check for the same node if the same node is mirror of itself or not.

PREDECESSOR AND SUCCESSOR OF AN ELEMENT

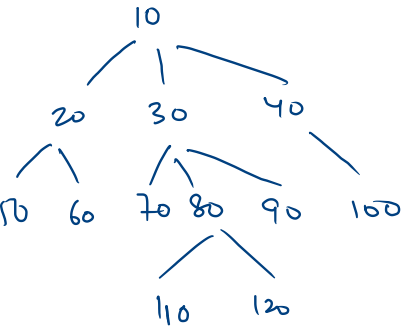


Pre → 110
Suc → 90

```
static Node predecessor;
static Node successor;
static int state = 0;
public static void predecessorAndSuccessor(Node node, int data) {
    // write your code here
    if(node.data==data){
        state = 1;
    }else if(state==0){
        predecessor = node;
    }else if(state == 1){
        successor = node;
        state = 2;
    }
    for(Node child:node.children){
        predecessorAndSuccessor(child,data);
    }
}
```

First if cond^n, we will check if we got the value, and will update the state, if state = 0, i.e. we haven't found the data == node value and we are traversing through the loop, and will update the value of pre, if state = 1, this means we have we have found node and update succ with the next node.

CEIL AND FLOOR IN GENERIC TREE



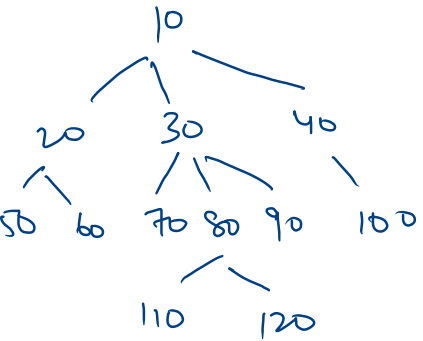
data = 65
ceil = ~~∞~~ 70
floor = ~~-∞~~ 60

```java
static int ceil;
static int floor;
public static void ceilAndFloor(Node node, int data) {
  // Write your code here
  if(node.data>data){
    if(node.data<ceil){
      ceil = node.data;
    }
  }
  if(node.data<data){
    if(node.data>floor){
      floor = node.data;
    }
  }
  for(Node child:node.children){
    ceilAndFloor(child,data);
  }
}
```

floor → largest among smaller values
ceil → smallest among larger values.

we will update floor and ceil and then call child node

K^{th} Largest Element in Generic Tree



K = 3

ele = 100

floor → largest among smaller

∞ → 120

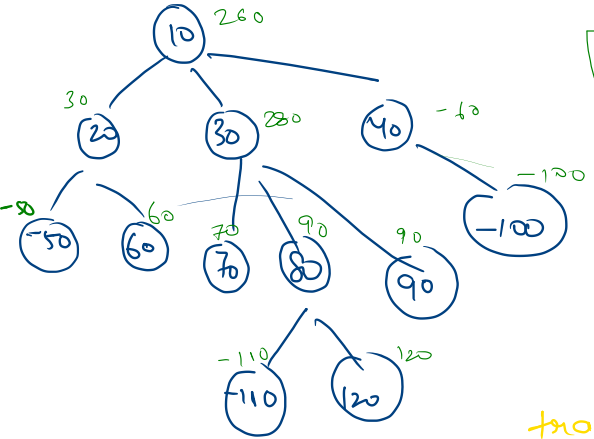
120 → 110

110 → 100

[calculate floor K times]

```java
public static int kthLargest(Node node, int k){
  // write your code here
  floor = Integer.MIN_VALUE;
  int factor = Integer.MAX_VALUE;
  while(k-- > 0){
    ceilAndFloor(node,factor);
    factor = floor;
    floor = Integer.MIN_VALUE;
  }
  return factor;
}
```

NODE WITH MAXIMUM SUBTREE SUM



30 @ 280

```java
public static int MSum = Integer.MIN_VALUE;
public static Node MNode;
public static int maxSubtreeSum(Node root){
  int sum =0 ;
  for(Node child:root.children){
    int cs = maxSubtreeSum(child);
    sum+=cs;
  }
  sum+=root.data;
  if(sum>MSum){
    MSum = sum;
    MNode = root;
  }

  return sum;
}
```

calculating sum of child node

add root value to sum

if sum > MSum update MSum

NOTE

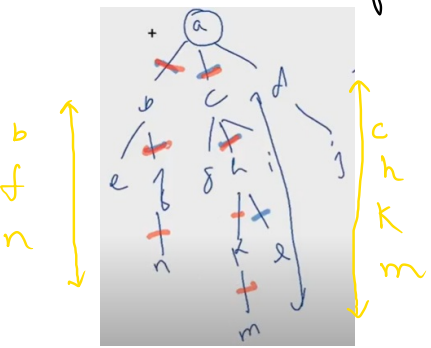
travel and change rule

≠

returning sum, but printing MSum and MNode

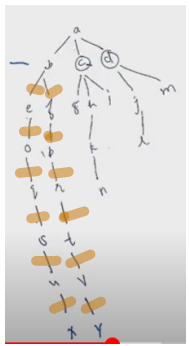
DIAMETER OF GENERIC TREE

↳ max no. of edges between any two node in a tree



largest child height
+ second largest child height
+ 2

[This formula will not work for below case]



diameter doesn't need to specifically pass from root node, diameter is max no. of edges b/w any 2 nodes as shown in this ex.

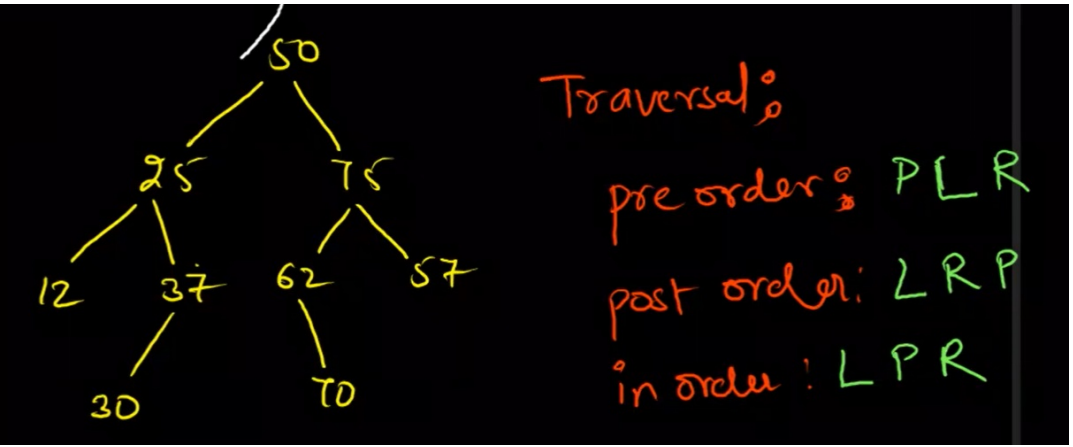
```
  calcuateDiaReturnHeight(root);
  System.out.println(dia);
}
static int dia=0;
public static int calcuateDiaReturnHeight(Node root){
  int dch = -1,sdch=-1; // deepest child height  // second deepest child height
  for(Node child:root.children){
    int ch = calcuateDiaReturnHeight(child);
    if(ch>dch){
      sdch = dch;
      dch = ch;
    }else if(ch>sdch){
      sdch = ch;
    }
  }
  if(dch+sdch+2>dia){
    dia=dch+sdch+2;
  }
  dch+=1;
  return dch;
}
```

→ func call

→ col child height

update dch & sdch

Diameter humesha root se pass nhi hoga.



L R P

L P R

Pre :- 50 25 12 37 30 75 62 70 57 ✓

Post : 12 30 37 25 70 62 57 75 50 ✓

In : 12 25 30 37 50 62 70 75 57 ✓