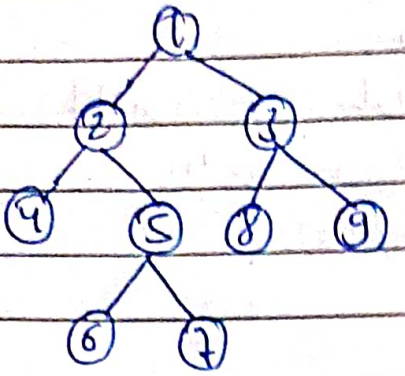
→ LCA (lowest Common Ancestor):



lca(4,7) is 2
ancestor of 4 are 2,1
7 are 5,2,1
lca = 2

lca(5,9) = 1
lca(2,6) = 2

a) Brute force
lca(4,7) node(4) path from root 1 → 2 → 4
node(7) ——— 1 → 2 → 5 → 7

store path in array

last matched node is our answer.

T(n) → O(N) + O(N) = O(N)
S(n) → O(N) + O(N) = O(2N)

b) We do d/s traversal
lca(n1, n2) We have following 3 cases for every node t

(i) If node is same as n1 or n2 we return node

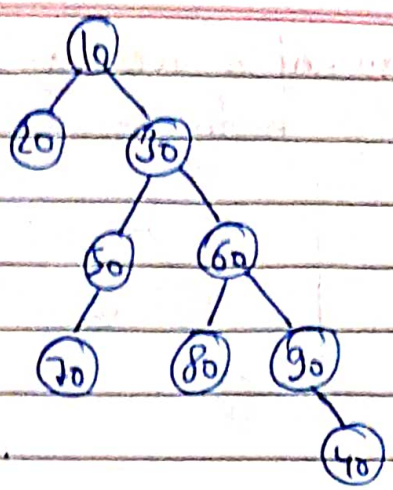
(ii) If node at one ~~side~~ subtree contain n1 and other contain n2 return node.

(iii) If none of node subtrees contain n1 or n2 return NULL.

TC → O(N)
SC → O(N) skewed tree

Case

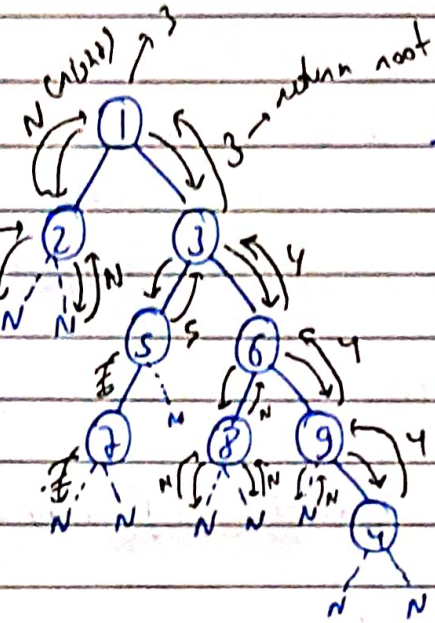
(I) n1 = 10, n2 = 40
return 10



(Case III) n1 = 80 n2 = 40
com_node = 60
return 60

(Case III) when we at node
n1 = 80 (2a) If left & right
n2 = 40 subtree don't contain
n1 or n2 return NULL.

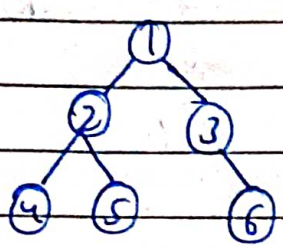
Example



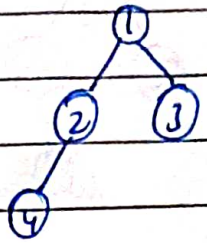

```
TreeNode *f(TreeNode *root, TreeNode *p, TreeNode *q)
{
    if(root == NULL) return root;   // case 3
    if(root == p || root == q) return root;   // case 1
    TreeNode *left = f(root->left, p, q);
    TreeNode *right = f(root->right, p, q);
    if(left == NULL)
        return right;
    else if(right == NULL)
        return left;
    else
        return root;   // case 2
}
```

→ Maximum Width of a Binary Tree :

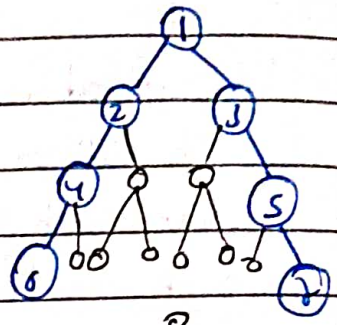
width = ~~ma~~ no. of nodes in a level between any 2 nodes



maximum width = 4

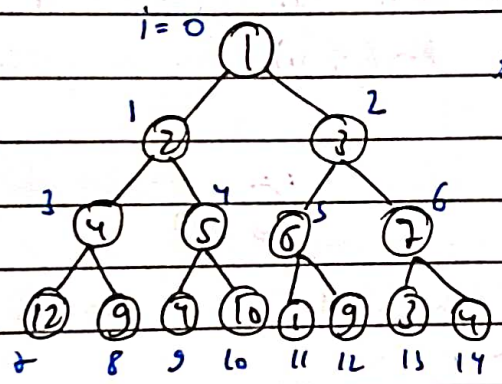


2



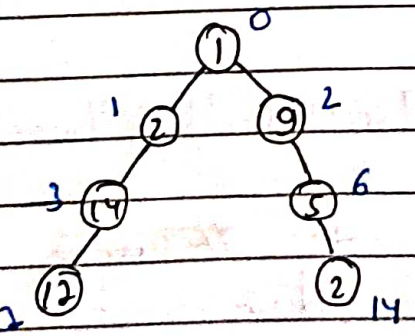
8

logic

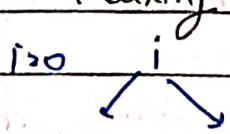


* We do indexing of every node
  left → (2i+1)
  right child → (2i+2)

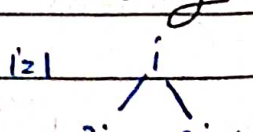
* at every level maximum width is (left index − right index + 1)



for 0-based indexing

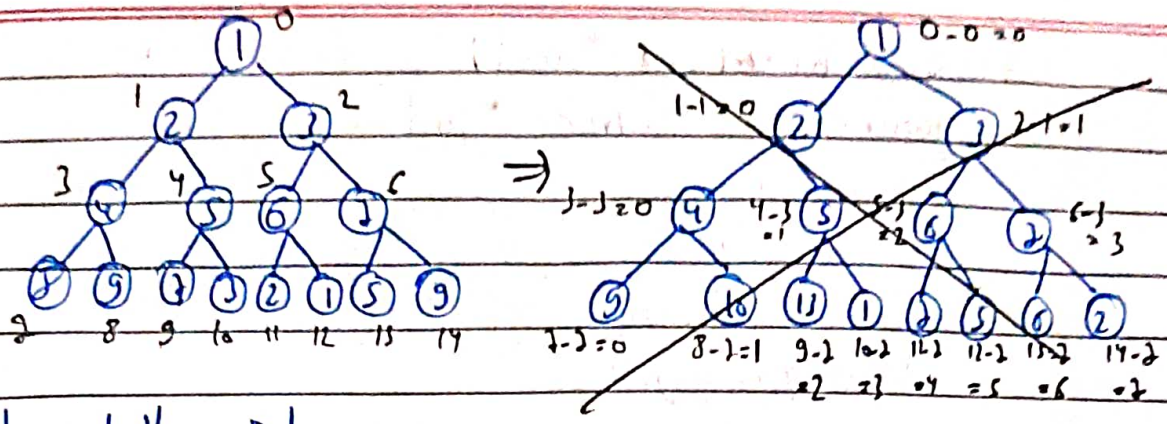


for 1-based indexing

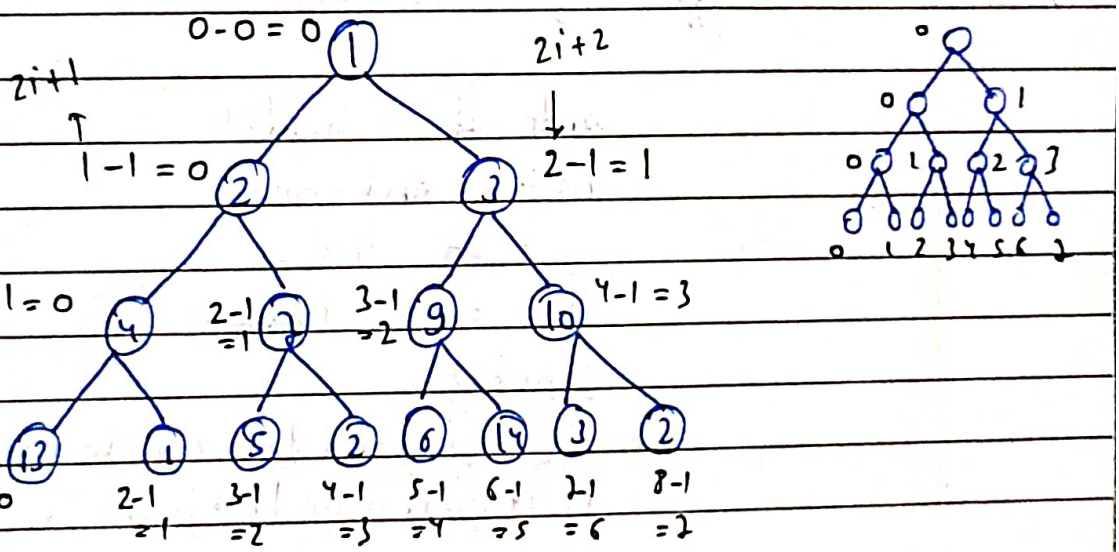


↳ Segment tree में यही logic आता है।

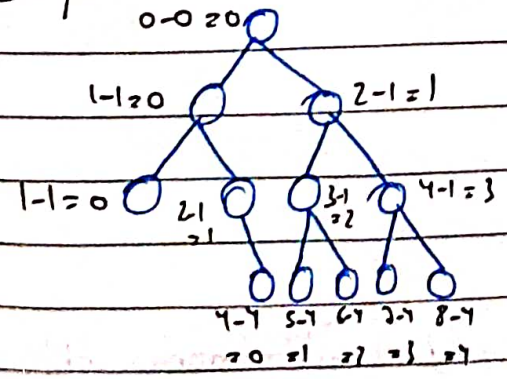
→ Problem in this approach is overflow
when doing (i+1) or (i+2) when the tree is skewed then these value will be very large causing Overflow



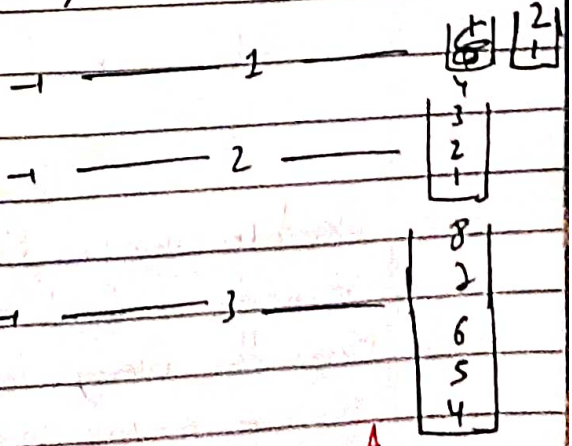
We changed the ~~index~~ logic in every level for every index, subtract the min ~~index~~ (min index is always queue.front()) → correct ✓



Example



→ for level 0 Queue, 0



Ans. 5

See level 4 properly

A

TC] O(N)
SC

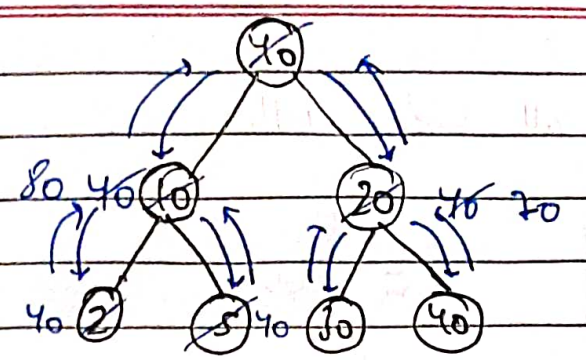
```
int width(TreeNode *root)
{   queue<pair<TreeNode *,int>> q;
    int ans=0;
    long long i=0;
    q.push({root,i});
    while(!q.empty())
    {
        long long int n= q.size();
        long long int min = q.front().second;
        int first,last;
        for(int j=0;j<n;j++)
        {
            auto it = q.front();
            i = it.second - min;
            if(j==0) first=0;
            if (j==n-1) last = i;
            q.pop();
            if(it.first->left) q.push({it.first->left,2*i+1});
            if(it.first->right) q.push({it.first->right,2*i+2});
        }
        ans = max(ans, last-first+1);
    }
    return ans;
}
```

→ Children Sum Property:

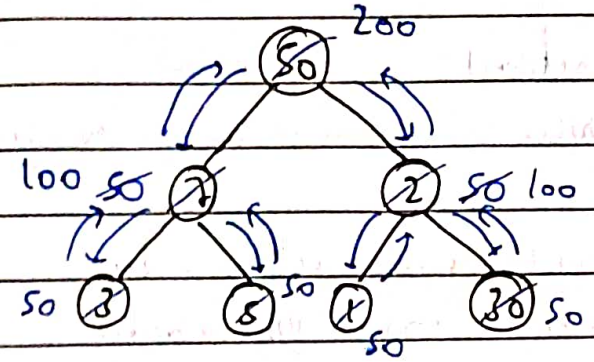
∀ node ⇒ (node->val) == (node->left->val) + (node->right->val)

Q Make the Binary Tree follow CSP

→ You can increase value of any node by +1 any no. of time

① 

- If sum of left & right child is < root, change both left & right child by root.
- while backtracking if (l+r) > root make root equal l+r

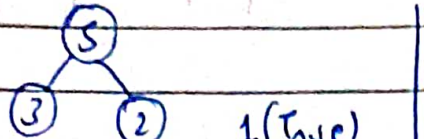
② 

O(N)

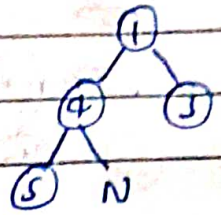
```
void f(TreNode *root)
{  if(root==NULL) return;
   int child=0;
   if(root->left)
       child += root->left->data;
   if(root->right)
       child += root->right->data;
   if(child >= root->data) root->data = child;
   else
   {  if(root->left) root->left->data = root->data;
      if(root->right) root->right->data = root->data;
   }
   f(root->left); f(root->right);
   int t=0;
   if(root->left) t += root->left->data;
   if(root->right) t += root->right->data;
   if(root->left && root->right) root->data = t; // leaf node
}
```

XIX Children Sum Parent

Given BT. Check if all nodes follow CSP.



1 (True)

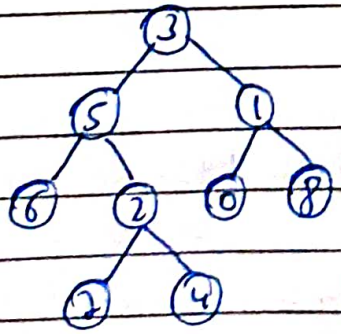


0

```
int f(Node *root)
{
    if(root == NULL) return 1;
*   if(root->left == NULL && root->right == NULL) return 1;
    int sum = 0;
    if(root->left)  sum += root->left->data;
    if(root->right) sum += root->right->data;
    return root->data == sum && f(root->left) && f(root->right);
}
```

- if a node is leaf we don't calculate further we return 1.

XX Print all Nodes at a distance k



k = 2 target = 5

o/p [7, 4, 1]

Step 1 : We ~~make~~ calculate parent of every node & store in map.

Step 2. we do level order traversal और हर node पर distance गिनते और k & target से

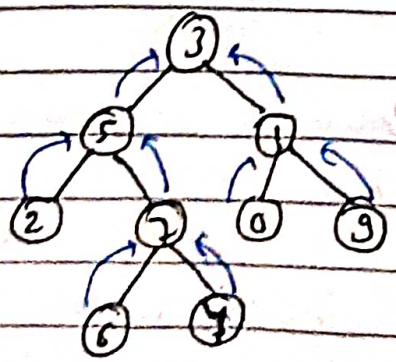
```
vector<int> f(TreeNode * root, TreeNode * target, int k)
{
    queue<TreeNode *> q;
    q.push(root);
    unordered_map<TreeNode *, TreeNode *> m;
    while(!q.empty())
    {
        TreeNode * node = q.front();
        q.pop();
        if(node->left)
        {
            m[node->left] = node;
            q.push(node->left);
        }
        if(node->right)
        {
            m[node->right] = node;
            q.push(node->right);
        }
    }
    q = queue<TreeNode *>();
    unordered_map<TreeNode *, bool> visited;
    visited[target] = 1;
    q.push(target);
    int dis = 0;
    while(!q.empty())
    {
        int n = q.size();
        if(dis == k) break;
        for(int i=0; i<n; i++)
        {
            TreeNode * node = q.front();
            q.pop();
```

calculating parent nodes

map में node नहीं है तो default value zero होता है

special case for root (root का कोई parent नहीं)

```
if(m[node] && visited[m[node]] == 0)
{
    visited[m[node]] = 1;
    q.push(m[node]);
}
if(node->left && visited[node->left] == 0)
{
    visited[node->left] = 1;
    q.push(node->left);
}
if(node->right && visited[node->right] == 0)
{
    visited[node->right] = 1;
    q.push(node->right);
}
}
dis++;
}
vector<int> ans;
while(!q.empty())
{
    ans.push_back(q.front()->val);
    q.pop();
}
return ans;
}
```

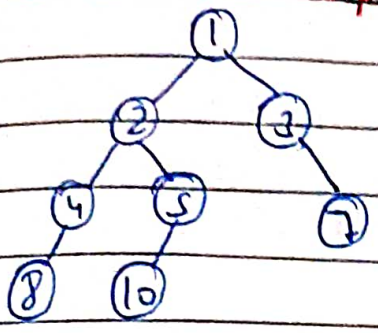


root का कोई parent नहीं।

T.C → O(N) + O(N)

S.C → O(N) + O(N) + O(N)

(XXI) Min Time Taken to Burn a BT from a node/leaf:



target = 10 (Given value of node not the node)
o/p 5

```
Node *t; // Global declaration
int minTime(Node *root, int target)
{
    f(root, target);
    queue<Node *> q;
    unordered_map<Node*, Node*> m;
    q.push(root);
    while(!q.empty())
    {
        Node *node = q.front();
        q.pop();
        if(node->left)
        {   m[node->left] = node;
            q.push(node->left);
        }
        if(node->right)
        {   m[node->right] = node;
            q.push(node->right);
        }
    }
    q = queue<Node *>();
    unordered_map<Node*, bool> visited;
    q.push(t);
    visited[t] = 1;
    int ans = 0;
```

Calculating Parent

```
while(!q.empty())                                   BFS traversal
{
    int n=q.size();
    int f=0;
    for(int i=0;i<n;i++)
    {
        Node *node=q.front();
        q.pop();
        if( m[node] && visited[m[node]]==0)
        {   visited[m[node]]=1;
            q.push(m[node]);
            f=1;
        }
        if(node->left && visited[node->left])
        {   visited[node->left]=1;
            q.push(node->left);
            f=1;
        }
        if(node->right && visited[node->right])
        {   visited[node->right]=1;
            q.push(node->right);
            f=1;
        }
    }
    if(f)
        ans++;
}
return ans;
}
```

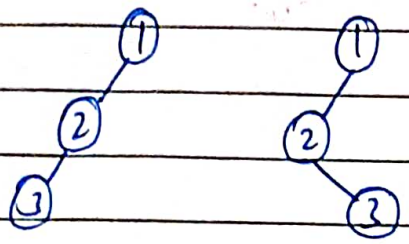
Search the target

```
void f(Node *root, int target)
{   if(root==NULL) return;
    if(root->data==target) { t=root; return;}
    f(root->left, target);
    f(root->right, target);
}
```

Point To Remember

You can create Unique BT only when
i) Inorder & Preorder Given
ii) Inorder & Postorder Given

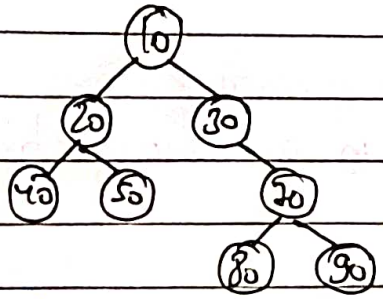
eg PreOrder 1 2 3
PostOrder 3 2 1



→ Two BT are possible or may be more, Therefore Inorder is necessary to make Unique BT.

(XXII) Construct BT from Inorder & Preorder:

in[] = {40, 20, 50, 10, 30, 80, 70, 90}
pre[] = {10, 20, 40, 50, 30, 70, 80, 90}



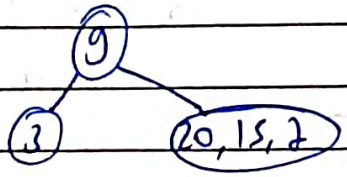
a)
```
int preindex=0; // Global                    instart ↓      ↓ inend
Node *f(int a[], int pre[], int is, int ie)                    O(n^2)
{
    if(is>ie) return NULL;
    Node *root = new Node(pre[preindex++]);
    int inindex;
    for(int i=is; i<=ie; i++)
        if(in[i]==root->key)
        {   inindex=i; break; }

    root->left = f(in, pre, is, inindex-1);
    root->right = f(in, pre, inindex+1, ie);
    return root;
}
```

Logic in[] = {3, 9, 20, 15, 7}

Root L R pre[] = {9, 3, 15, 20, 7}

* pre का root store होता है (pre का एक एक वर्ड है और हर एक Node के लिए in[] का उसका left and right subtree मिलेगा।

eg: for node (9) in the array in[]



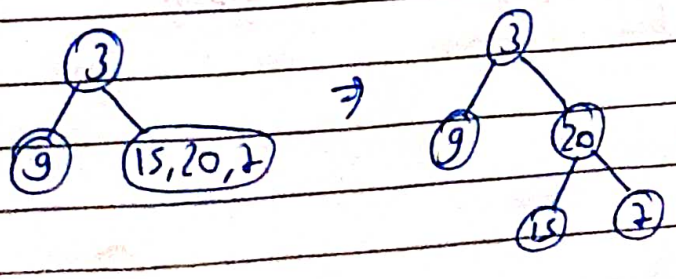
b)
```
TreeNode *buildTree(vector<int>& pre, vector<int>& in)
{
    unordered_map<int, int> m;
    for(int i=0; i<in.size(); i++)
        m[in[i]] = i;
    int preindex = 0;
    TreeNode *root = f(pre, in, 0, in.size()-1, preindex, m);
    return root;
}
TreeNode *f(vector<int>& pre, vector<int>& in, int instart,
            int inend, int& preindex, unordered_map<int,int>& m)
{
    if(instart > inend) return NULL;
    TreeNode *root = new TreeNode(pre[preindex++]);
    int x = m[root->val];
    root->left = f(pre, in, instart, x-1, preindex, m);
    root->right = f(pre, in, x+1, inend, preindex, m);
    return root;
}
```

(XIII) Construct BT from inorder & postorder: O(N)

```
TreeNode *buildTree(vector<int>&in, vector<int>&post)
{
    map<int,int> m;
    for(int i=0; i<in.size(); i++)
        m[in[i]]=i;
    int postindex = post.size()-1;
    TreeNode *root = f(in, 0, in.size()-1, post, postindex, m);
    return root;
}
TreeNode *f(vector<int>&in, int instart, int inend, vector<int>&post,
    int &postindex, map<int,int>&m)
{
    if(instart>inend) return NULL;
    TreeNode *root = new TreeNode(post[postindex--]);
    int x = m[root->val];
    root->right = f(in, x+1, inend, post, postindex, m);
    root->left = f(in, instart, x-1, post, postindex, m);
    return root;
}
```

root ka right पहले left बाद में ।
bcz postorder में L R Root

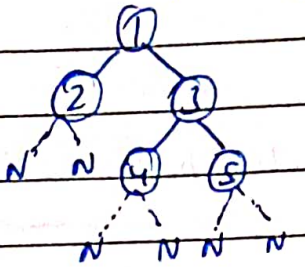
eg: in[] = [9,3,15,20,7]
post[] = [9,15,7,20,3] ★ postorder में root को right को left से पहले



(24) Serialize And De-serialize BT :

Given root of BT first serialize it then de-serialize it

(i) Serialize: convert BT into string (it your choice to choose b/s & d/s)

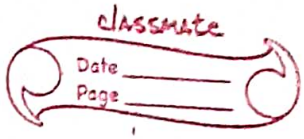


string s = "1,2,3,#,#,4,5,#,#,#,#,"

(ii) de-serialize : take this string & convert in BT

```
string serialize (TreeNode *root)          O(N)
{                                          O(N)
    string s;
    if(root == NULL) return s;
    queue<TreeNode *> q;
    q.push(root);
    while(!q.empty())
    {
        TreeNode *node = q.front();
        q.pop();
        if(node == NULL) s += "#,";
        else
            s += (to_string(node->val) + ",");
        if(node != NULL)
        {
            q.push(node->left);
            q.push(node->right);
        }
    }
    return s;
}
```

sth changes like → 1 → 2 → 3 → # → # ---.
';' जगह ही stop.




```
TreeNode * deserialize(string data)                    O(N)
{   if(data.size()==0) return NULL;                    O(N) → queue
    queue<TreeNode *> q;
    stringstream s(data);
    string str;          ——→ delimeter
    getline(s,str,',');
    TreeNode *root = new TreeNode(stoi(str));
    q.push(root);
    while(!q.empty())
    {
        TreeNode *node = q.front();
        q.pop();
        getline(s,str,',');
        if(str=="#")
            node->left = NULL;
        else
        {   TreeNode *new_node = new TreeNode(stoi(str));
            node->left = new_node;
            q.push(new_node);
        }
        getline(s,str,',');
        if(str=="#")
            node->right = NULL;
        else
        {   TreeNode *new = new TreeNode(stoi(str));
            node->right = new_node;
            q.push(new_node);
        }
    }
    return root;
}
```

sstream header file

① stringstream
syntax: stringstream s(str); // used for breaking words
. it helps you to read the string as if it was a stream (like cin)

② getline
. used to read a string or a line from a input stream
★ it extracts characters from the input stream and appends it to the string object until the delimiter character is encountered

eg: string s = "abc,1,2,ef,gh"
stringstream st(s); string str;
while(getline(st, str, ','))
cout<<str<<endl;

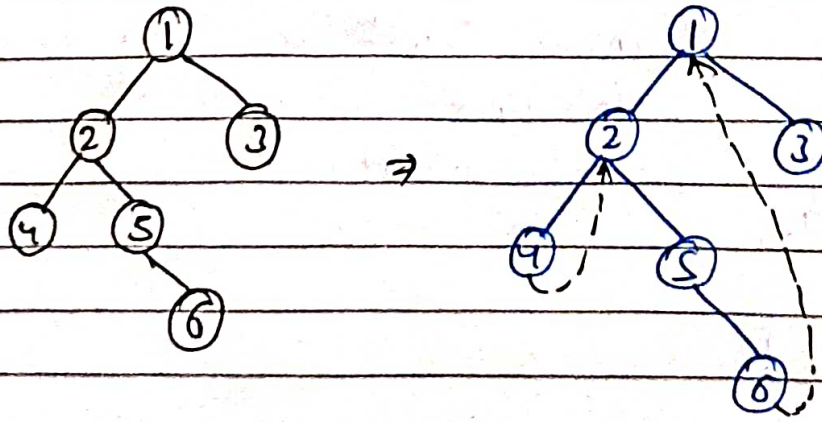
अब s एक stream बन गया है

o/p abc
1
2
ef
gh

automatically आगे बढ़ते जाएगा st में stringstream ही मदद से।

③ string s(5, 'k');
cout<<s; o/p kkkkk

→ Morris Traversal : TC-O(N) SC-O(1)



a) Inorder

```
vector<int> f(TreeNode *root)
{   vector<int> inorder;
    TreeNode *curr = root;
    while(curr != NULL)
    {
        if(curr->left == NULL)
        {   inorder.push_back(curr->val);
            curr = curr->right;
        }
        else
        {   TreeNode *prev = curr->left;
            while(prev->right && prev->right != curr)
                prev = prev->right;
            if(prev->right == NULL)
            {   prev->right = curr;
                curr = curr->left;
            }
            else
            {   prev->right = NULL;
                inorder.push_back(curr->val);
                curr = curr->right;
            }
        }
    }
    return inorder;
}
```

in total this while loop runs N times

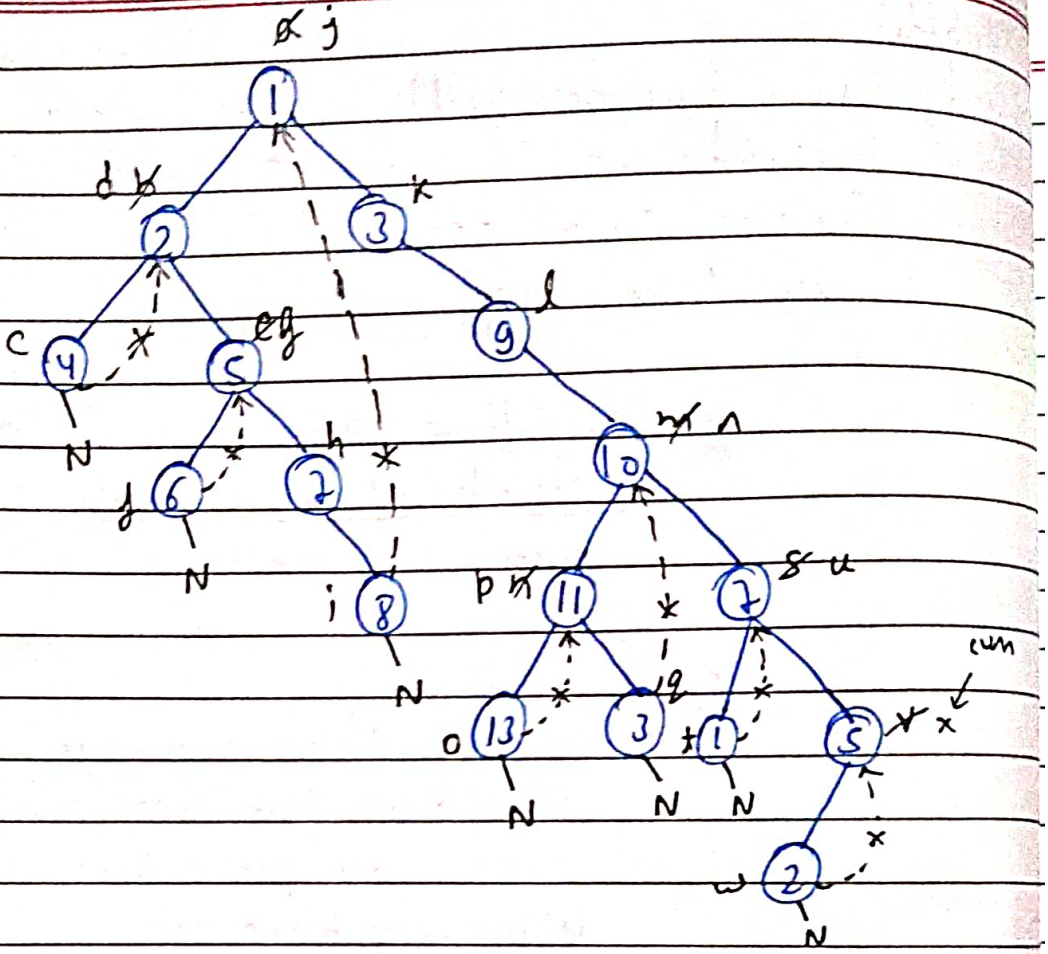
making the link

removing link

for preorder write this line here

Steps उठाओ curr का left NULL नहीं है तो (prev = curr->left) and we go right -> right->right so on until (prev->right != NULL)

(a, b, ... x) are just current variable



* A node is traversed at most 3 times (see node 8)

curr = curr->right
= NULL
while loop stop

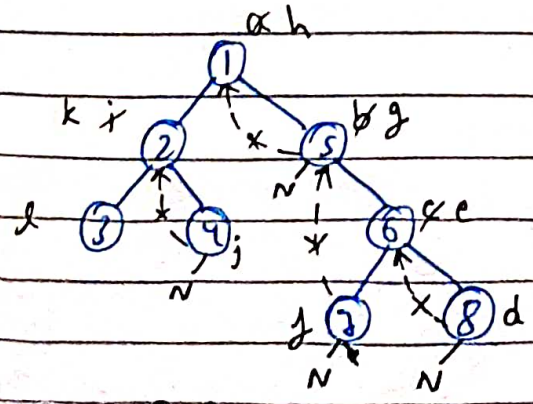
b) Postorder:

```
vector<int> postorder(TreeNode *root)
{
    TreeNode *curr = root;
    vector<int> pre;
    while(curr != NULL)
    {
        if(curr->right == NULL)
        {
            pre.push_back(curr->val);
            curr = curr->left;
        }
    }
}
```

variable galti se kr le taki don't be confused.

```
TreeNode *prev = cur->right;
while(prev->left && prev->left != cur)
        prev = prev->left;
if(prev->left == NULL)
{
        prev->left = cur;
        pre.push_back(cur->val);
        cur = cur->right;
}
else
{       prev->left = NULL;
        cur = cur->left;
}
}
}
reverse(pre.begin(), pre.end());
return pre;
}
```

* Here we modify preorder Morrical Traversal Root→Left→Right to (Root→Right→Left)
* Here we create thread from left most node of right subtree to cur node,
* We cover right subtree first then left subtree



Root R L

1 5 6 8 7 2 4 3

cur = cur->left
= NULL
stop

reverse

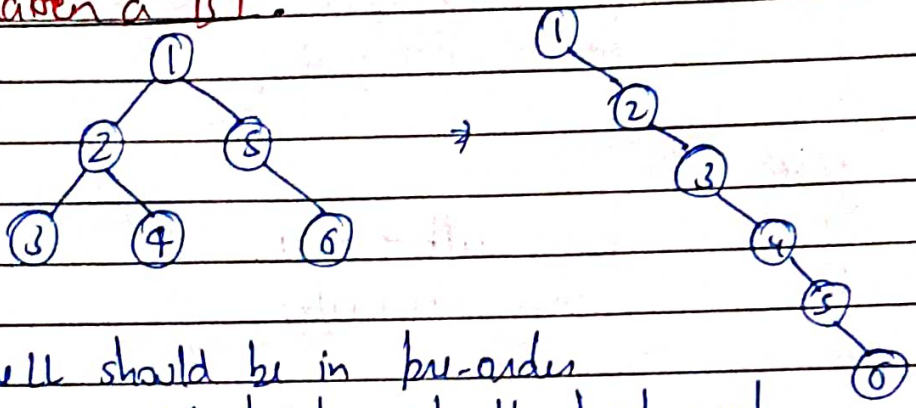
3 4 2 7 8 6 5 1

Very Imp Point

→ Preorder: Root L R

→ अगर हम Preorder को change कर दें Root R L & store the result in vector → then reverse the vector we get Postorder.

(25) Flatten a BT:

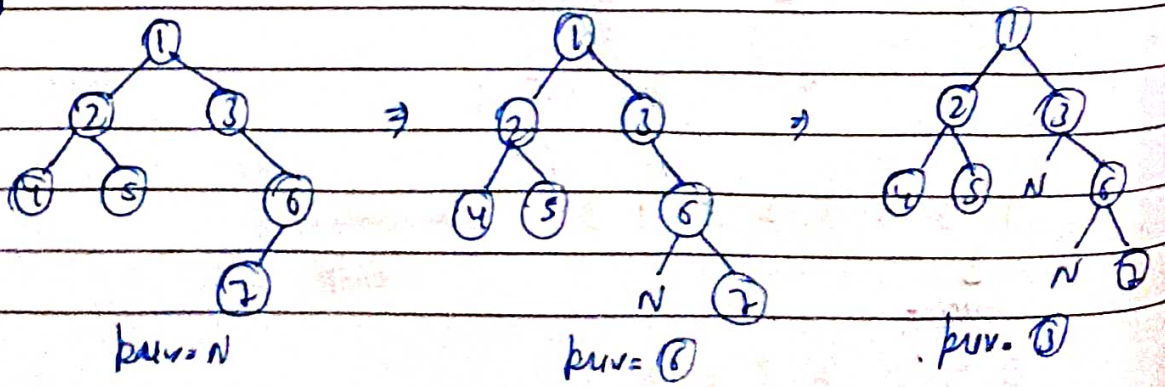


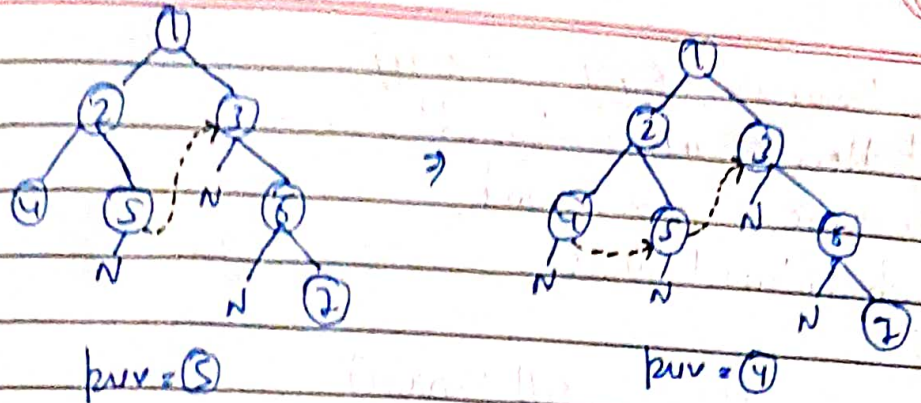
- The LL should be in pre-order
- The LL right pointer should point next node & left child ——— be NULL.

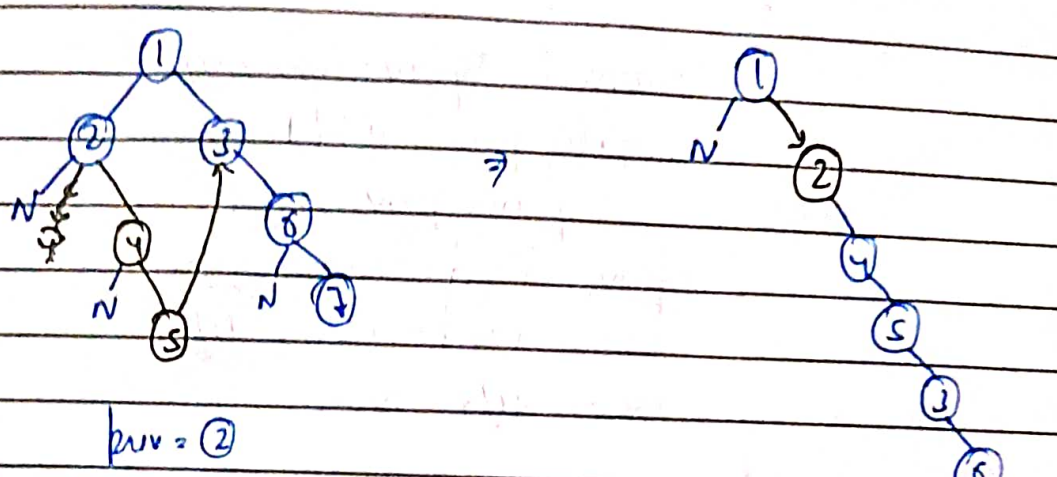
a) O(N) O(N) Recursion

```
TreeNode *prev = NULL;
void flatten(TreeNode *root)
{
    if(root == NULL) return;
    flatten(root->right);
    flatten(root->left);
    root->right = prev;
    root->left = NULL;
    prev = root;
}
```

See video if not get





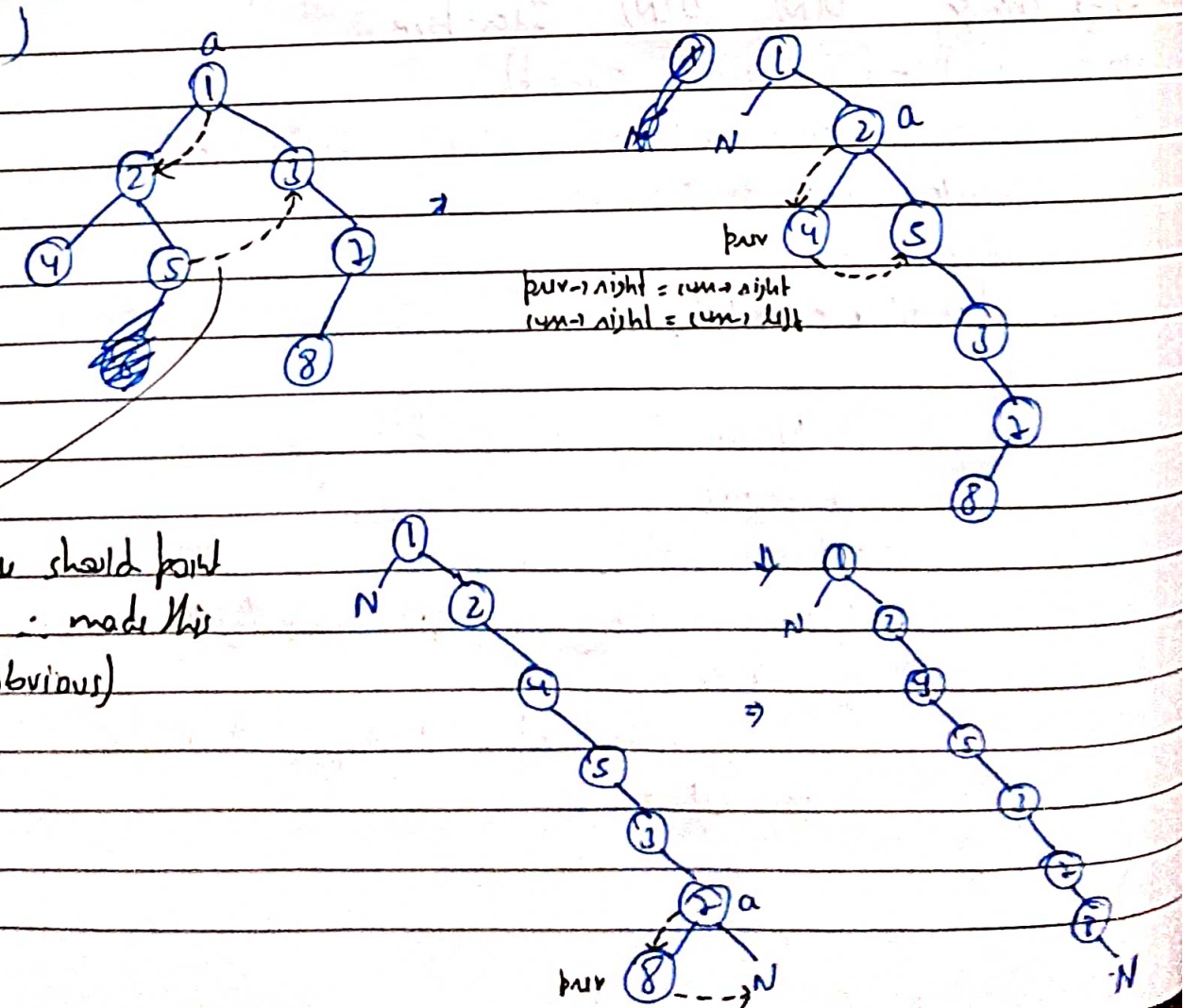


b) Using stack O(N) O(N) Iterative

```
void flatten(TreeNode *root)
{   if(root == NULL) return;
    stack<TreeNode *> st;
    st.push(root);
    while(!st.empty())
    {
        TreeNode *curr = st.top();
        st.pop();
        if(curr->right) st.push(curr->right);
        if(curr->left) st.push(curr->left);
        if(!st.empty())
            curr->right = st.top();
        curr->left = NULL;
    }
}
```

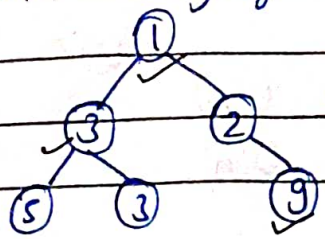
1) Morris traversal O(N) S(-O(1)

```
void flatten(TreeNode *root)
{   TreeNode *curr=root;
    while(curr!=NULL)
    {
        if(curr->left!=NULL)
        {
            TreeNode *prev=curr->left;
            while(prev->right!=NULL)
                prev=prev->right;
            prev->right=curr->right;
            curr->right=curr->left;
            curr->left=NULL;
        }
        curr=curr->right; // link change & : right
                             not left
    }
}
```



(26) Find Largest Value in Each Tree Row :

return array of largest value in each row of the tree



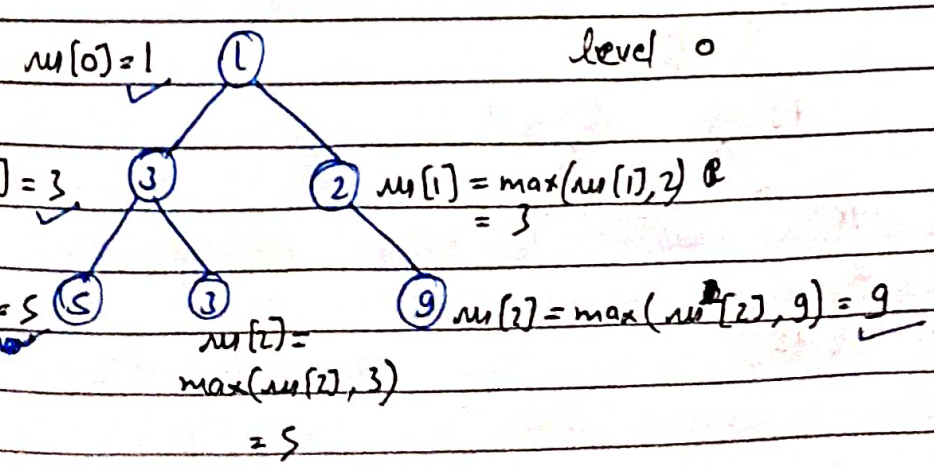
o/p [1, 3, 9]

DFS

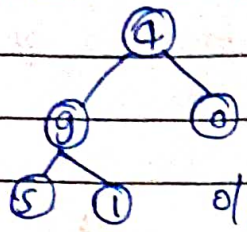
T.C - O(N)

S.C - O(h) → due to recursion

```
vector<int> largest(TreeNode *root)
{
    vector<int> ans;
    traverse(root, ans, 0);
    return ans;
}
void traverse(TreeNode *root, vector<int> &ans, int level)
{
    if(root == NULL) return;
    if(ans.size() == level)
        ans.push_back(root->val);
    else
        ans[level] = max(ans[level], root->val);

    traverse(root->left, ans, level+1);
    traverse(root->right, ans, level+1);
}
```



(27) Sum Root to leaf Numbers :



495 + 491 + 90

o/p → 1026

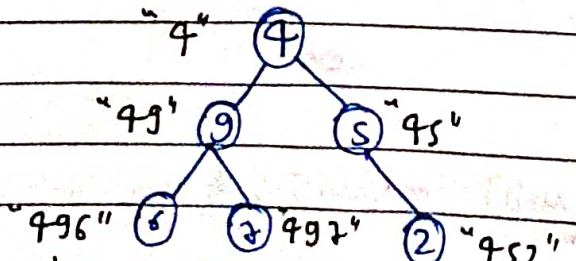
DFS

TC- O(N)

SC- O(h)

a)
```
int sum=0;
int sumNumbers(TreeNode *root)
{   f(root,0);
    return sum;
}
void f(TreeNode *root, int ans)
{
    if(root==NULL) return;
    if(root->left==NULL && root->right==NULL) // leaf
    {
        sum+= (ans*10)+root->val;
        return;
    }
    f(root->left, ans*10+root->val);
    f(root->right, ans*10+root->val);
}
```

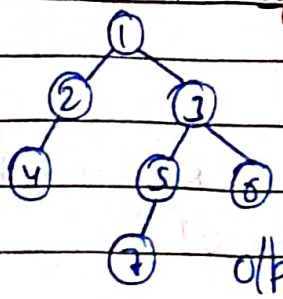
b) we can use string concept also to pass in f() in place of ans



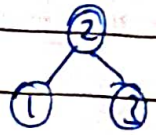
do it,

↳ ans+=stoi("496")

(28) Find Bottom left Tree Value



o/p → 7



o/p - 1

```
int findBottom(TreeNode *root)
{
    int ans=0, ma=0;
    f(root, 1, ma, ans);
    return ans;
}
void f(TreeNode *root, int level, int &ma, int &ans)
{
    if(root==NULL) return;
    if(level > ma)
    {
        ans = root->val;
        ma = level;
    }
    f(root->left, level+1, ma, ans);
    f(root->right, level+1, ma, ans);
}
```

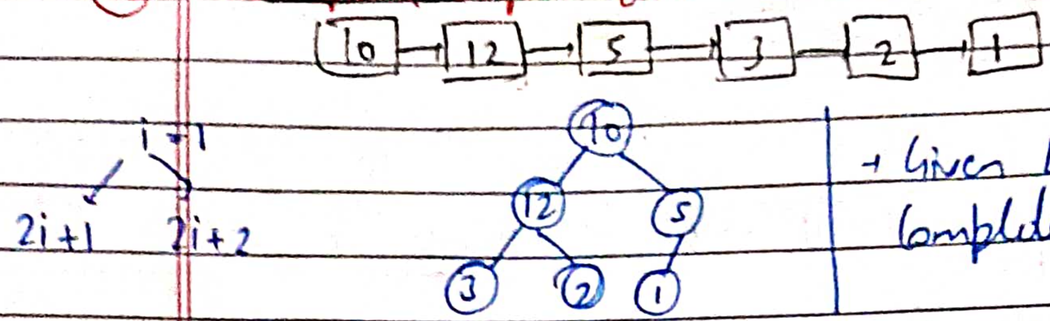
address wise therefore if ma changed can't be backtracked.

(29) Copying a Vector:

```
vector<int> v={3,9,2,5,1,9,6};
vector<int> ans;
copy(v.begin()+2, v.end()-2, back_inserter(ans));
for(auto x: ans)
    cout<<x<<" ";
```

o/p → 2 5 1

(29) BT from linked list:



→ Given LL Representation of Complete BT

BFS

O(N)

O(N)

```
void convert(Node *head, TreeNode *&root)
{
    if(head == NULL) return;
    queue<TreeNode *> q;
    root = new TreeNode(head->data);
    q.push(root);
    head = head->next;
    while(!q.empty() && head != NULL)
    {
        TreeNode *temp = q.front();
        q.pop();
        temp->left = new TreeNode(head->data);
        q.push(temp->left);
        head = head->next;
        if(head)
        {
            temp->right = new TreeNode(head->data);
            q.push(temp->right);
            head = head->next;
        }
    }
}
```

(`&root` → D)

(new TreeNode → don't count in SC *)

- while loop की एक iteration में particular node का left & right set कर रहे।

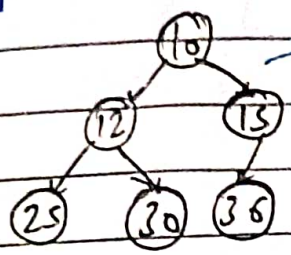
* b/z Node & TreeNode दो अलग-अलग structure हैं ∴ नया TreeNode तो बनाना ही पड़ेगा।

L Root R

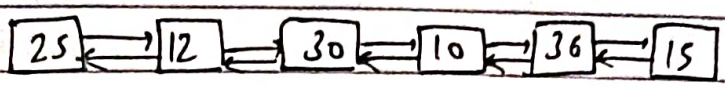
in-place

(30) BT to DLL :

~~prev to left~~ → left will be prev, right will be next.



→ inorder → 25 12 30 10 36 15



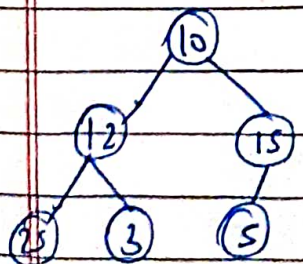
→ The order of nodes in DLL must be same as inorder of the given BT.

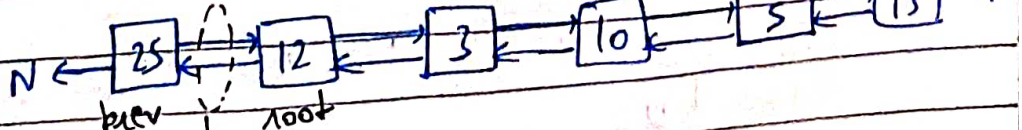
```
Node *prev = NULL;
Node *bToDLL(Node *root)
{
    if(root==NULL) return NULL;
    Node *head = bToDLL(root->left);
    if(prev==NULL) head=root;
    else
    {
        root->left = prev;
        prev->right = root;
    }
    prev = root;
    bToDLL(root->right);
    return head;
}
```

inorder ↓ left ← Root Right ←

// एक बार चलेगा (DLL का head assign होगा)

- left जाव, फिर करो, right जाव
  L Root R





else वाला में link लगा रहा

root → left = prev

prev → right = root

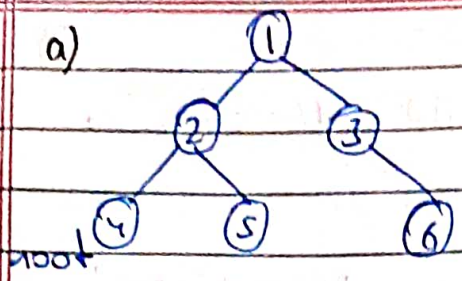
root->left = NULL (when)

head = root (4)
prev = root (4)

root->right = NULL when when head;

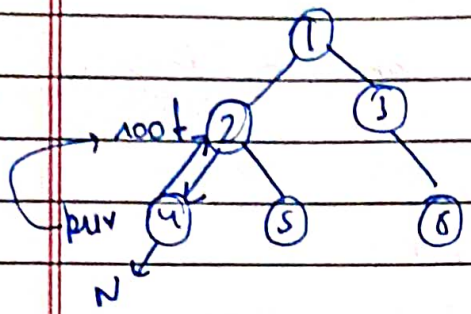
(3) prev = NULL

a)

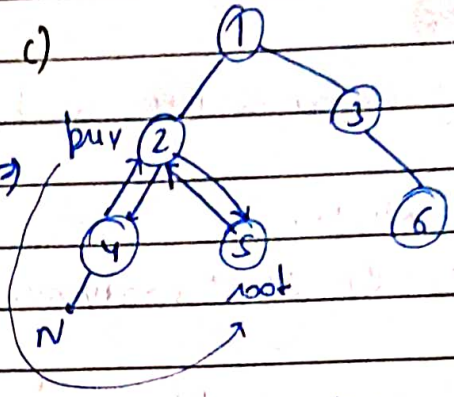


⇒

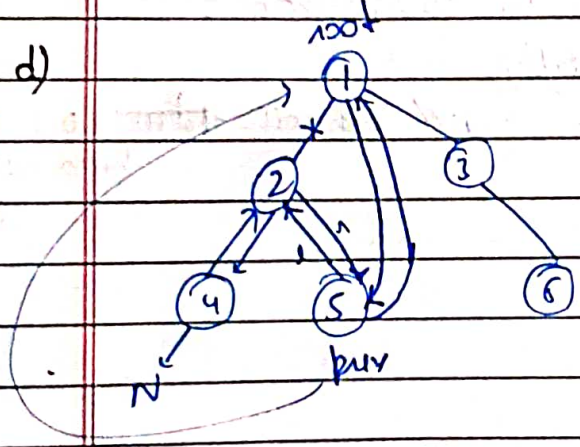
b)



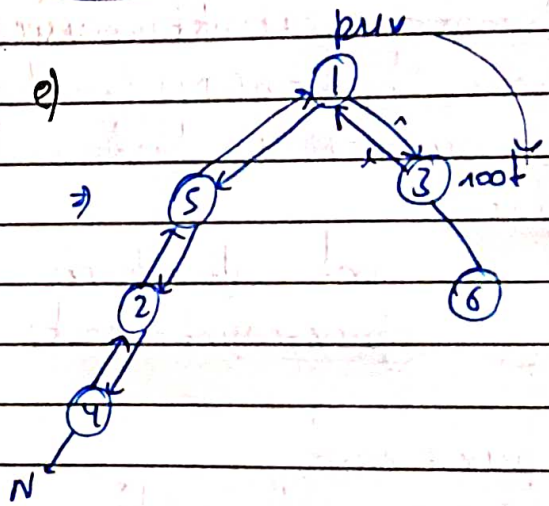
c)



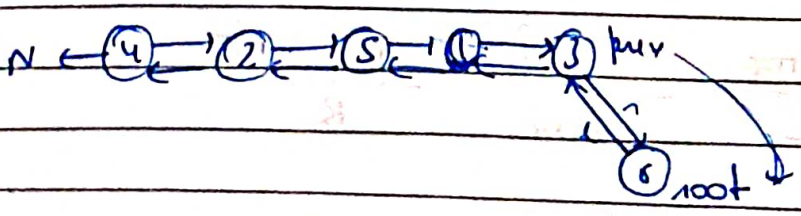
d)



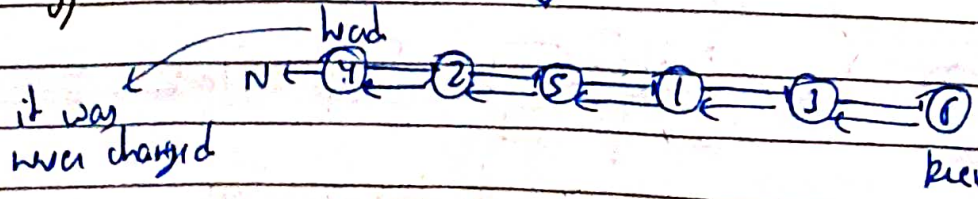
e)



f)



g)



1 (root->right) = NULL when when head;

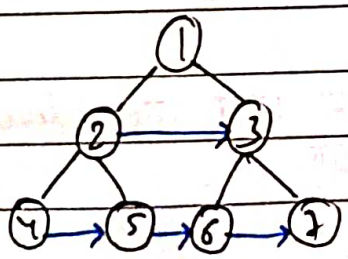
(31) BT to CDLL

```
Node *prev = NULL;
Node * CDLL (Node *root)
{
    Node *ans = f(root);
    ans->left = prev;
    prev->right = ans;
    return ans;
}
Node *f()  // same as previous Question
```

(32) Populating next right pointers in each Node:

```
struct Node{
  int val;
  Node *left, *right, *next;
};
```

- Given a perfect BT
- initially all next pointers are set NULL
-

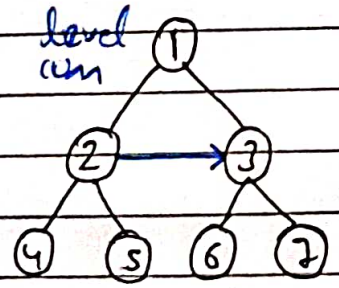


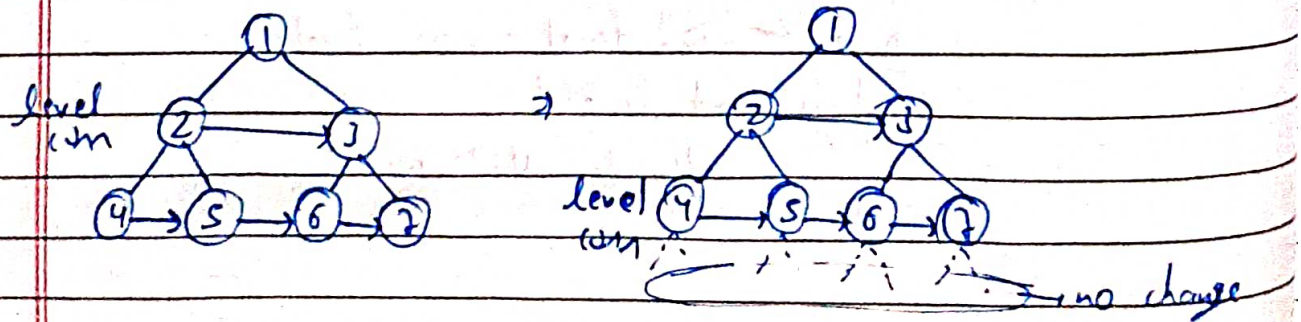

```
Node *connect(Node *root)
{   Node *curr_level = root;
    while(curr_level != NULL)
    {
        Node *next_level = curr_level;
        while(next_level)
        {   if(nl->left != NULL)
```

next_level = nl

```
Node *connect(Node *root)
{
    Node *level = root;
    while(level != NULL)
    {
        Node *run = level;
        while(run != NULL)
        {
            if(run->left != NULL)
                run->left->next = run->right;
            if(run->next != NULL && run->right != NULL)
                run->right->next = run->next->left;
            run = run->next;
        }
        level = level->left;
    }
    return root;
}
```

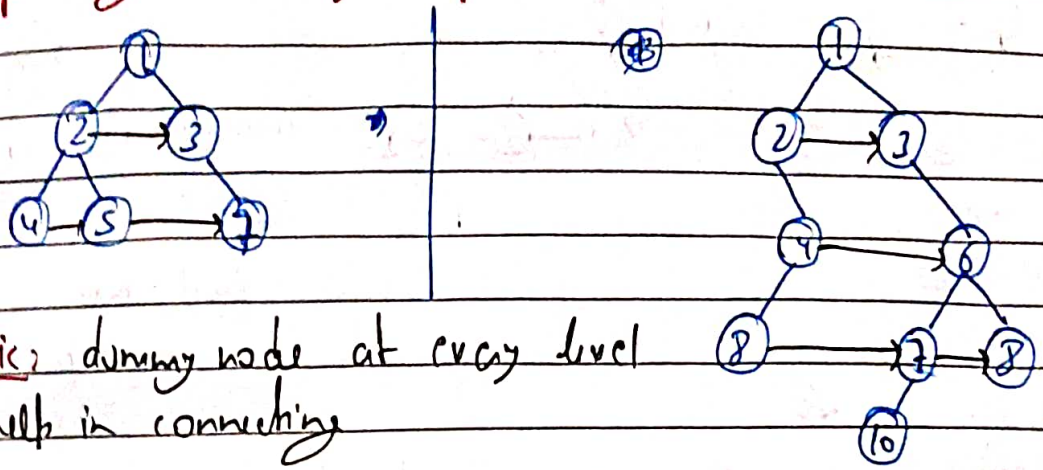
* level को fix रखो, run को बढ़ाते जाव
* run अपने नीचे वाले level को join करेगा





• run = run->next help in level में आगे जाने में।

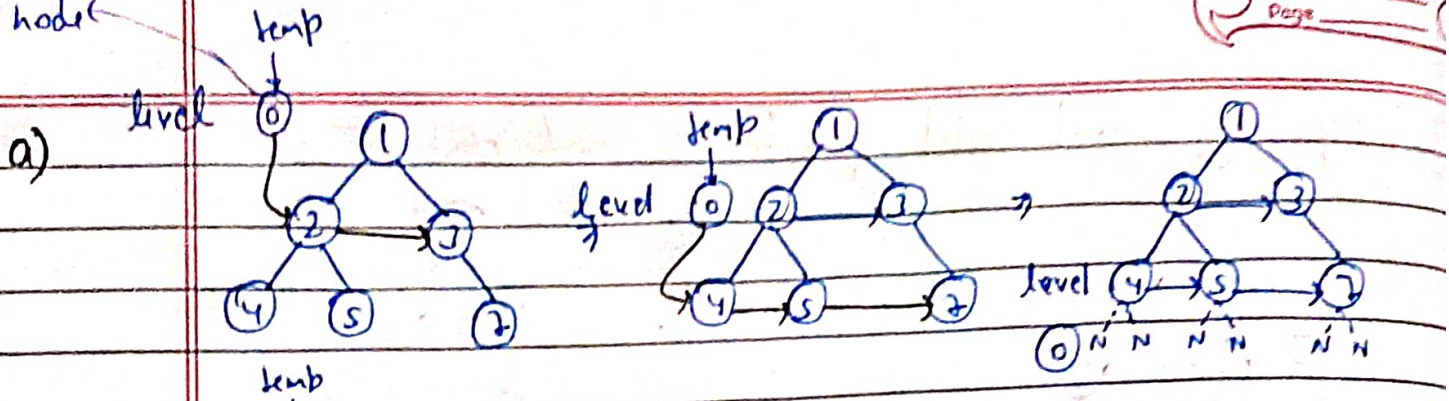
(33) Populating next right pointer in each Node II :



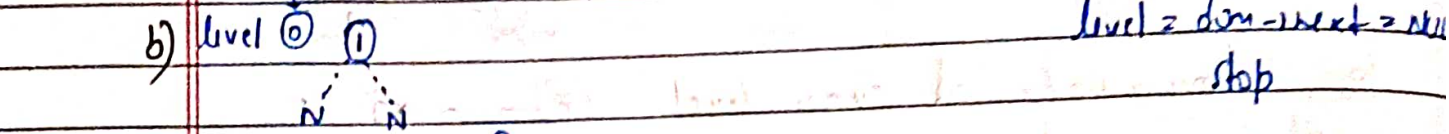
logic: dummy node at every level help in connecting

```
Node *connect(Node *root)
{   if(root == NULL) return root;
    Node *level = root;
    while(level != NULL)
    {
        Node *curr = level;
        Node *dum = new Node(0);
        Node *temp = dum;
        while(curr != NULL)
        {
            if(curr->left)
            {   temp->next = curr->left;
                temp = temp->next;
            }
            if(curr->right)
            {   temp->next = curr->right;
                temp = temp->next;
            }
            curr = curr->next;
        }
        level = dum->next;
```

a)



b)



while loop चलाएंगे, जब तक curr = curr->next then stop

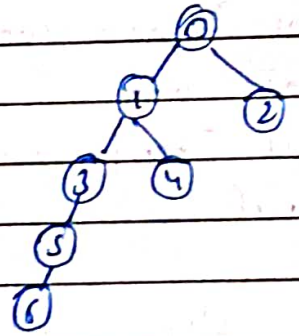
level = dum->next = NULL stop

(34) Construct BT from Parent Array :

O(N) · If two element have same parent the one that appear first in the array will be left child.

(i) a[] = {-1, 0, 0, 1, 1, 3, 5}
root

o/p = 0 1 2 3 4 5 6



(ii) a[] = {2, 0, -1}

o/p = 2 0 1

```
Node *createTree(int parent[], int N)
{
    unordered_map<int, Node*>> m; // to store nodes
    for(int i=0; i<N; i++)
        m[i] = new Node(i);      } simply index का node बना देंगे।
    Node *root = NULL;
    for(int i=0; i<N; i++)
    {
        if(parent[i] == -1)
        {
            root = m[i];          } root of tree calculation
            continue;
        }
```

यदि left child full है तो right में जाएँ

```
if(m[parent[i]]->left == NULL)
    m[parent[i]]->left = m[i];
else
    m[parent[i]]->right = m[i];
}
return root;
}
```

- parent[] = {-1, 0, 0, 1, 1, 3, 5}
  (0 1 2 3 4 5 6)

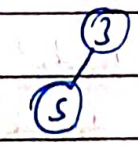
1 का parent 0 है

* array indexes are the value in tree nodes and array values gives the parent node of that particular index.

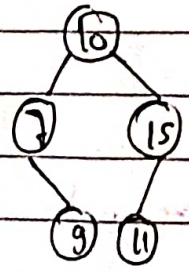
map में सारे nodes पड़े हैं इसी का उसको connect कर दें

say i=5
इसीलिए पता है 3 का left child है 5
parent[i] == 3
m[parent[i]] = (3)
m[parent[i]]->left

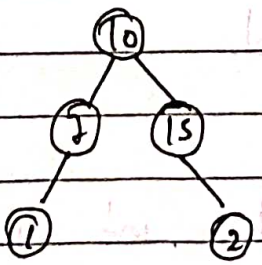
m[parent[i]]->left = m[i]



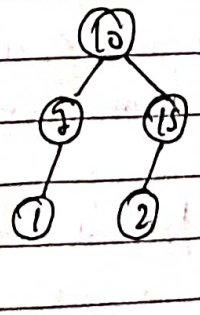
(35) Foldable BT



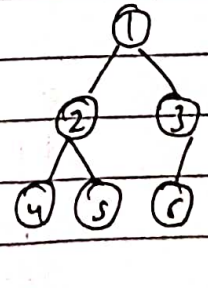
o/p 1



1



0



0

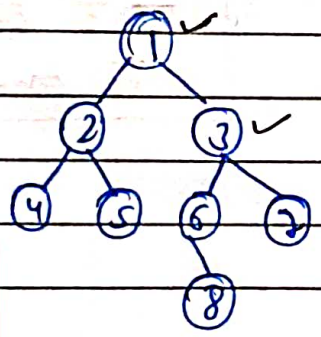
left & right subtree should be structure wise foldable.

```
bool isFoldable(Node *root)
{   if(root == NULL) return 1;
    bool a= return f(root->left, root->right)
}

bool f(Node *p, Node *q)
{   if(p==NULL || q==NULL) return p==q;
    return f(p->left, q->right) && f(p->right, q->left);
}
```

(See the implementation of map)

(36) **Node at distance :**

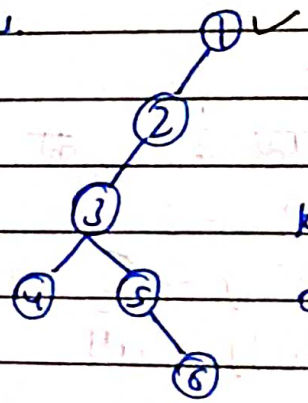
→ task is to count all distinct nodes that are distance k from a leaf node.



k=2

o/p 2

level



k=4

o/p = 1

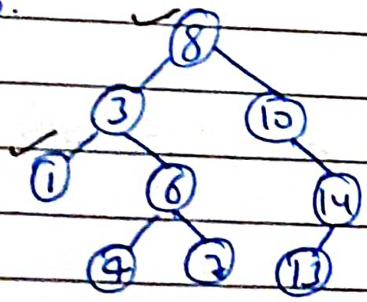
```
unordered_map<int, pair<Node*, bool>> m;
int nodeatdis(Node *root, int k)
{
    int count=0;
    f(root, k, 0, count);
    return count;
}
void f(Node *root, int k, int level, int &count)
{
    if(root == NULL) return;
    m[level].first = root;
    m[level].second = false;
```

```
    if(root->left==NULL && root->right==NULL) // leaf node
    {
        if((level-K)>=0 && m[level-K].second==false)
        {
            m[level-K].second = true;
            count++;
        }
    }

    f(root->left, k, level+1, count);
    f(root->right, k, level+1, count);
}
```

एक बार तो node के लिए तो बताएं न तो ∴ maintaining map (visited)

(32) Maximum Difference B/w Node And Ancestor:

→ find maximum v = | a.val - b.val | & a is the ancestor of b.



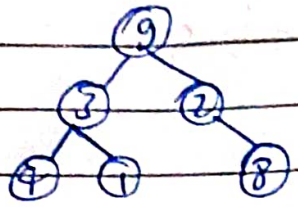
|8-1| = 7 Ans

```
int ans= INT_MIN;
int ancestor(TreeNode *root)
{
    int maxi=-1, mini=
    return f(root, root->val, root->val);  return ans;
}

void f(TreeNode *root, int maxi, int mini)
{
    if(root==NULL) return;
    maxi = max(maxi, root->val);
    mini = min(mini, root->val);
    f(root->left, maxi, mini);
    f(root->right, maxi, mini);
    ans = max(ans, maxi-mini);
}
```

(38) Verify Preorder Serialization of a BT :

Given string s = "9,3,4,#,#,1,#,#,2,#,6,#,#"

is Null Node



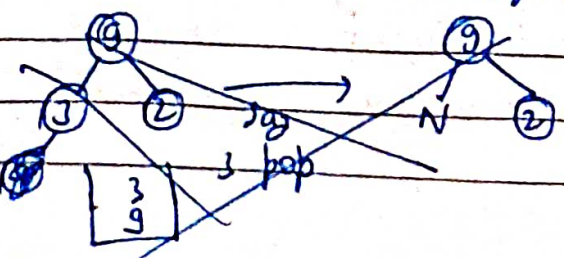
→ return true if it is correct preorder traversal.

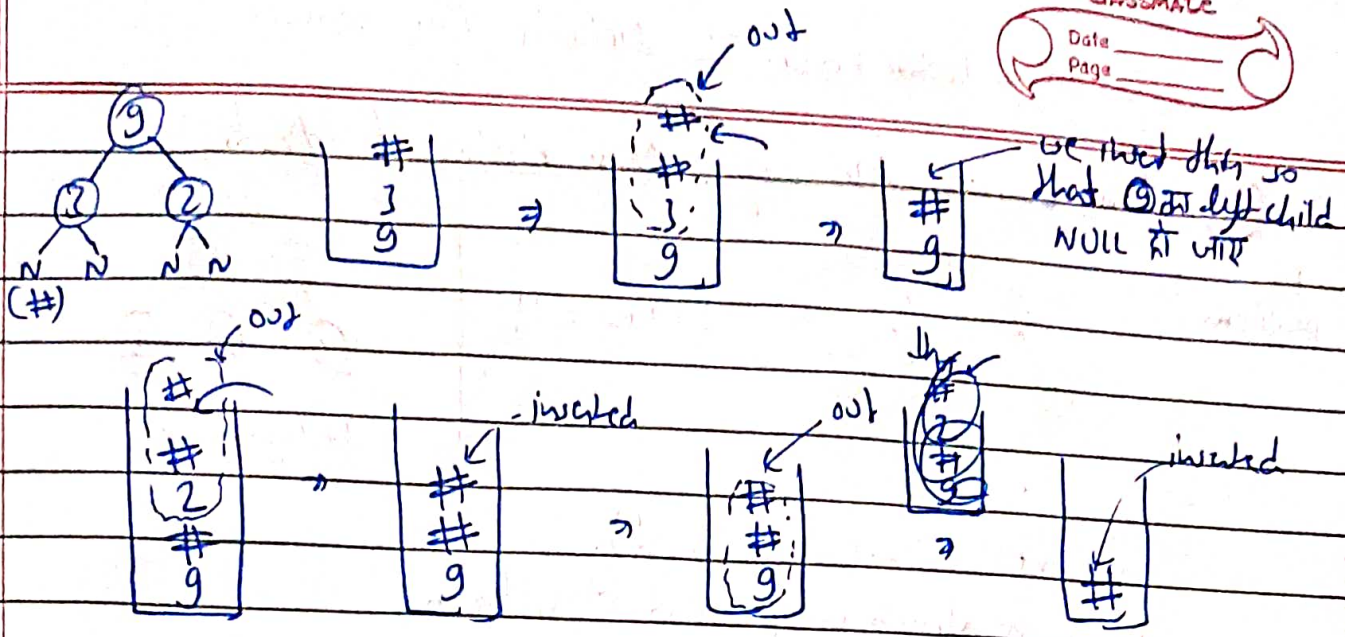
→ Not allowed to construct BT.

a) Using Stack

```
bool isValid(string preorder)
{   stringstream s(preorder);
    string str;
    stack<string> st;
    while(getline(s, str, ','))
    {
        string curr = str;
        while(!st.empty() && st.top()=="#")
        {
            st.pop();
            if(st.empty())
                return 0;
            st.pop();
        }
        st.push(curr); // curr हमेशा push हो रहा
    }
    return st.size()==1 && st.top()=="#";
}
```

. सोचो जब कोई character (node) जब pop होगा तो उसका जगह खाली हो जाएगा, ∴ we push "#" also so that उसके ऊपर के node का left/right child null हो जाए।





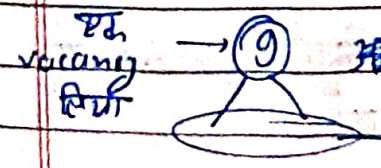
∴ last # # condition is
st.size == 1 && st.top() == "#

b) Simple logic (vacancy logic)

```
bool isValid(string preorder)
{
    int vacancy = 1;
    stringstream s(preorder);
    string str;
    while(getline(s, str, ','))
    {
        vacancy--;
        if(vacancy < 0) return 0;
        if(str != "#")
            vacancy += 2;
    }
    return vacancy == 0;
}
```

- जब भी node आयेगा तो vacancy create होगी और एक vacancy होगी (string में number)
- जब भी "#" आयेगा एक vacancy होगी।

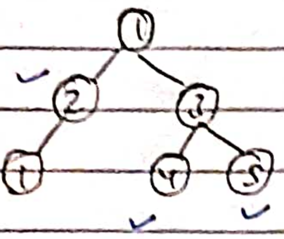
∴ vacancy--; always.



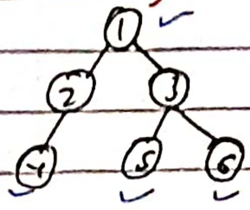
337. House Robber III Leetcode (See discuss)

(39) Maximum Sum of Non-Adjacent Nodes:

DP problem



2+4+5 = 11



1+4+5+6 = 16

```
int getMaxSum(Node *root)
{    unordered_map<Node*, int> m;
     return f(root, m);
}

int f(Node *root, unordered_map<Node*, int> &m)
{       if(root==NULL) return 0;
        if(m[root]) return m[root];
        int withnode = root->data;          // take root
        if(root->left)
        {
              withnode += f(root->left->left, m);
              withnode += f(root->left->right, m);
        }
        if(root->right)
        {
              withnode += f(root->right->left, m);
              withnode += f(root->right->right, m);
        }
        int withoutnode = f(root->left, m) + f(root->right, m);
        return m[root] = max(withnode, withoutnode);
}                              // don't take root
```

map का use कर रहे ताकि दुबारा किसी node पे जाए तो उसकी value map से ले सके।

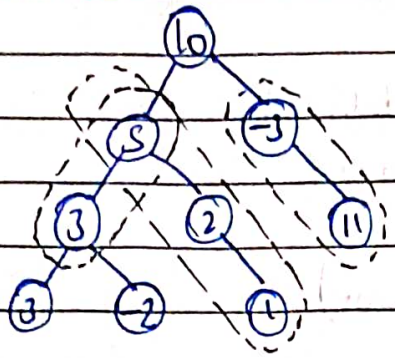
Roobu III sol

DP sol

```
int rob(TreeNode *root)
{
    vector<int> sum = robSub(root);
    return max(sum[0], sum[1]);
}
vector<int> robSub(TreeNode *root)
{
    if(root==NULL)
        return vector<int>(2,0);
                    → Dp array of size 2
    vector<int> left = robSub(root->left);
    vector<int> right = robSub(root->right);

    vector<int> sum(2,0);
    sum[0] = max(left[0], left[1]) + max(right[0], right[1]);
    sum[1] = root->val + left[0]+ right[0];

    return sum;
}
```

. The left & right vector's 0th index value if root is not robbed, second value is if root is robbed.

(10) Path Sum III



target = 8

```
int pathSum(TreeNode *root, int t)
{   unordered_map<long long int, int> m;
    int c=0;
    long long cum_sum=0;
    f(root, t, m, c, cum_sum);
    return c;
}

void f(TreeNode *root, int t, unordered_map<long long int,
    int> m, int &c, long long int cum_sum)
{
    if(root==NULL) return;
    cum_sum += root->val;

    if(cum_sum==t) c++;
    if(m[cum_sum-t]) c += m[cum_sum-t];

    m[cum_sum]++;

    f(root->left, t, m, c, cum_sum);
    f(root->right, t, m, c, cum_sum);
}
```

(41) SubArray with given sum k

nums = [2, 3, 3, -1, 0, 7, -2, 5] k = 5

prefix sum → 2 5 8 7 7 14 12 17

12 - 5 = 7 (7 वां तक आया है)

∴ ct += m[sum-k];

```
int sum(vector<int> &nums, int k)
{   unordered_map<int,int> m;
    m[0]=1;
    int sum=0, ct=0;
    for(int i=0; i<nums.size(); i++)
    {
        sum+=nums[i];
        if(m[sum-k]) ct+=m[sum-k];
        m[sum]++;
    }
    return ct;
}
```