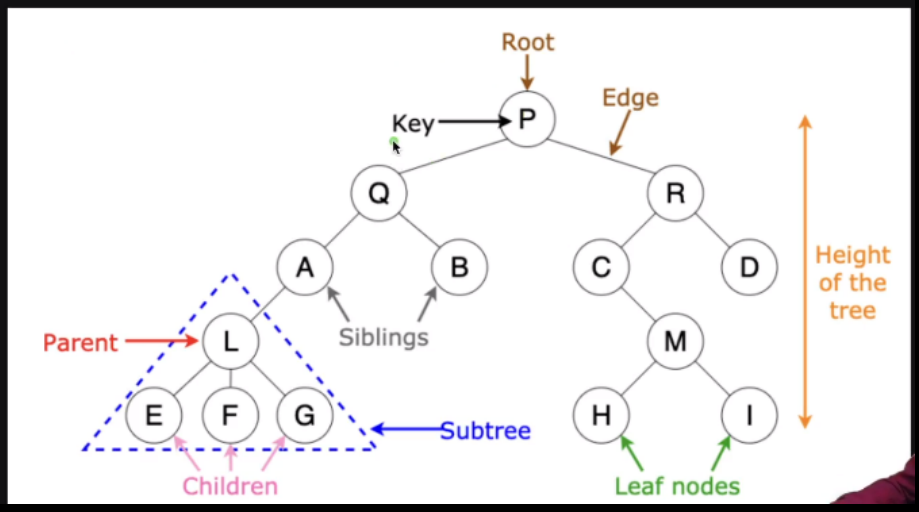
"Binary Trees"

Binary Tree Recap



If we build the Root then left then Right it is called Preorder Traversal.

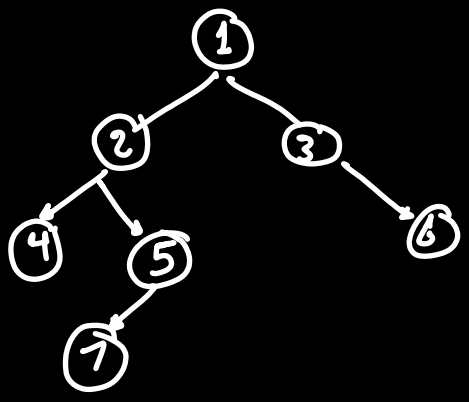
* How to build a tree?

```cpp
class node {

    public:
    int data;
    node *left;
    node *right;

    node(int d) {
        data = d;
        left = NULL;
        right = nullptr;
    }
};

node* buildTree() {
    int data;
    cin >> data;
    if(data == -1) return NULL;
    node* newNode = new node(data);
    newNode->left = buildTree();
    newNode->right = buildTree();
    return newNode;
}
```

→ This is called the Constructor which initialize a tree once I pass a value to it.

* Implement level order Traversal: →

↳ Suppose you have a tree like that: →



⇒ The Result after level order Traversal should be like that

1
2 3
4 5 6
7

How that can be done?

* Implementation: →

1	2	3	4	5	6

① First you should Push the Root and the size of the que will be one, that donates that there is only one element in this level.

② Add its two child and Pop out the elemnt [1]
[2,3]

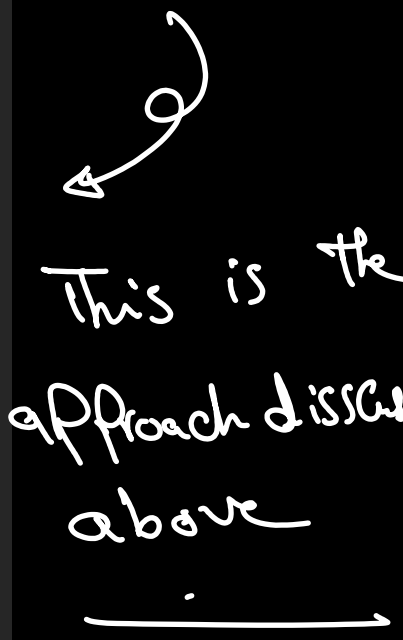
③ the new size will be 2 which indicates the new line Contains two elemnts only

④ After adding the new children the new que size became three which indicates the no of Elemnts in the next line.

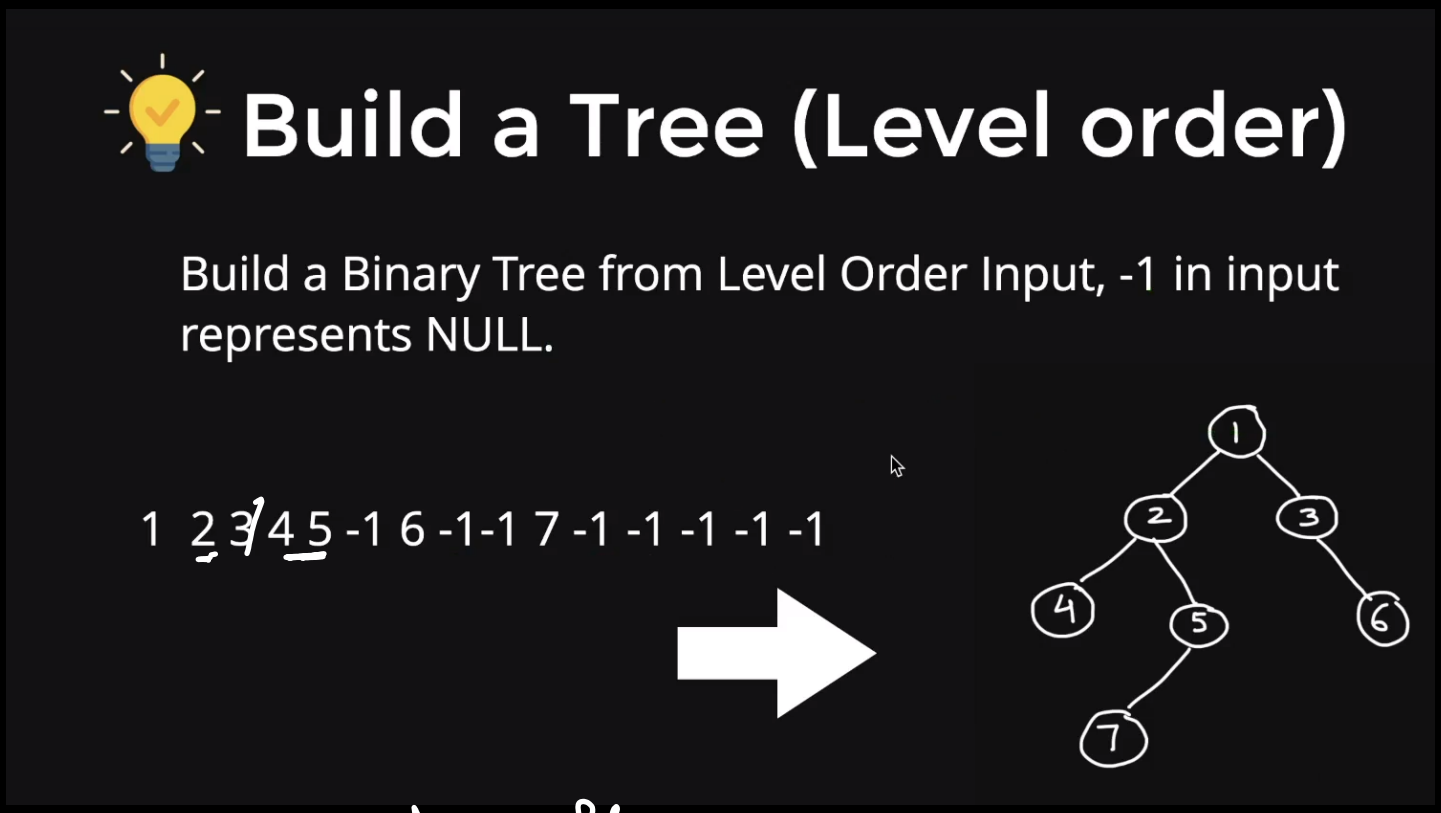
↳ and so on.....

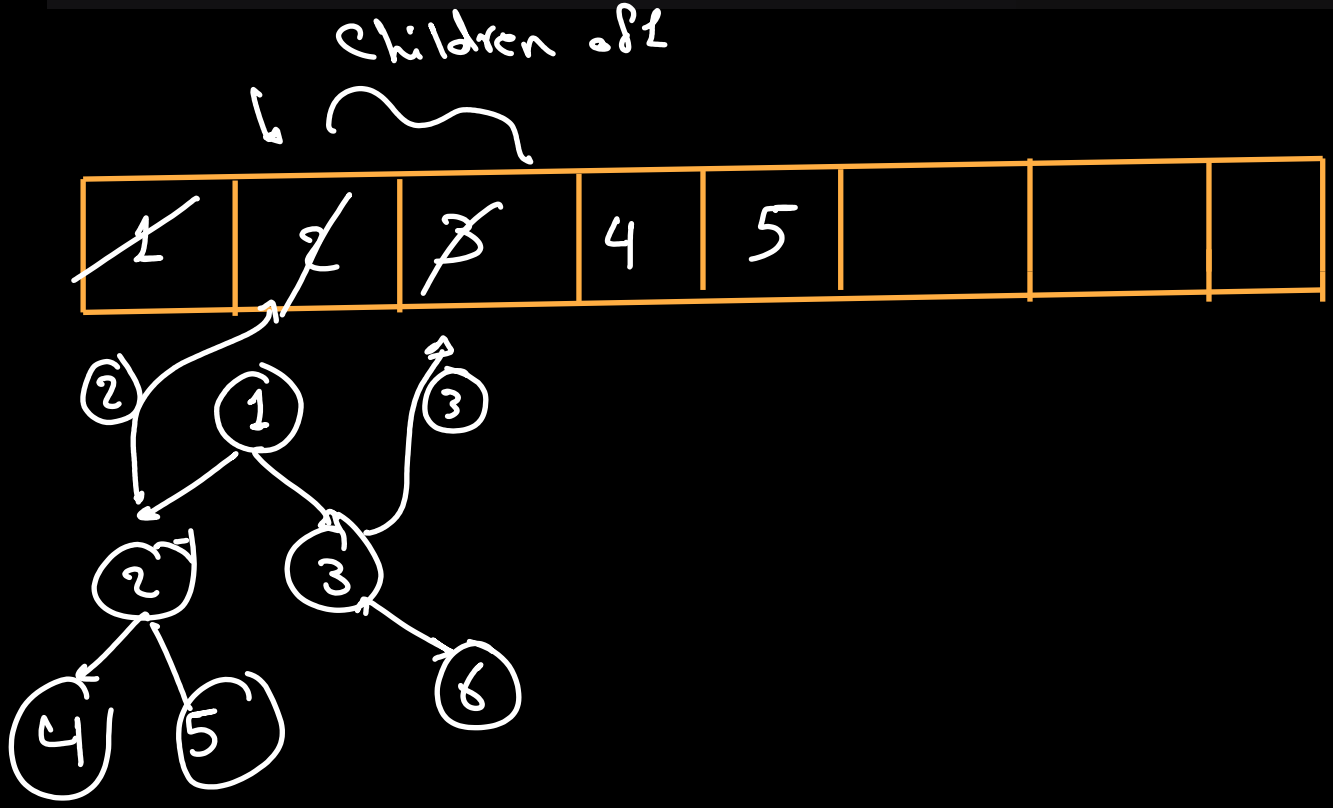
```cpp
void levelOrderTraversal(node *root) {
    queue<node*> qu;
    qu.push(root);
    qu.push(NULL);
    node *tempNode = nullptr;
    while(!qu.empty()) {
        tempNode = qu.front();
        if(!qu.front()) {
            cout << endl;
            qu.pop();
            if(!qu.empty())
              qu.push(NULL);
        } else {
            qu.pop();
            cout << tempNode->data << " ";
            if(tempNode->left)
              qu.push(tempNode->left);
            if(tempNode->right)
              qu.push(tempNode->right);
        }
    }
    return;
}
```

```cpp
vector<vector<int>> levelOrder(TreeNode* root) {
    vector<vector<int>> res;
    if(!root) return res;
    vector<int> tempRes;
    queue<TreeNode *> qu;
    qu.push(root);
    int newQSize = 0;
    while(!qu.empty()) {
        newQSize = qu.size();
        for(int i = 0 ; i < newQSize; i++) {
          tempRes.push_back(qu.front()->val);
          if(qu.front()->left)
            qu.push(qu.front()->left);
          if(qu.front()->right)
            qu.push(qu.front()->right);
          qu.pop();
        }
        res.push_back(tempRes);
        tempRes.clear();
    }
    return res;
}
```



* How to perform the Level order build:→





* First step is to Push the root to que.
* check if the left and Right != -1 this mean there is a child otherwise it is null.

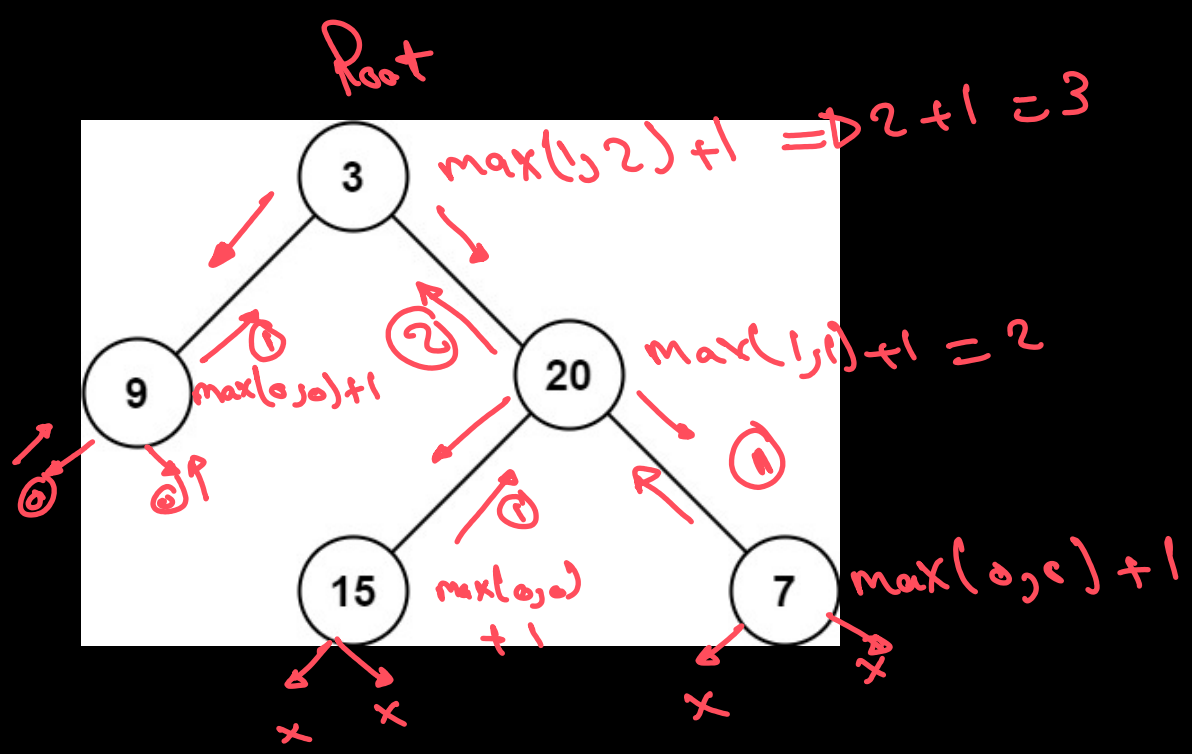
```
node *levelOrderBuild() {
    int data;
    queue<node *> qu;
    cin >> data;
    node *current = new node(data);
    node *root = current;
    qu.push(current);
    while(!qu.empty()) {
        int leftChild, rightChild;
        cin >> leftChild >> rightChild;
        current = qu.front();
        if(leftChild != -1) {
          current->left = new node(leftChild);
          qu.push(current->left);
        }

        if(rightChild != -1) {
            current->right = new node(rightChild);
            qu.push(current->right);
        }
        qu.pop();
    }
    return root;
}
```

* Binary Tree height *

* Calculate height of ordinary Binary Trees?

```
int maxDepth(TreeNode* root) {
    if(!root)
        return 0;
    int leftHeight = maxDepth(root->left);
    int rightHeight = maxDepth(root->right);
    return 1 + max(leftHeight, rightHeight);
}
```

*It can be done by simply Traversing the left Part of the Tree and the Right Part then get the max of Both.



Height Balanced Tree Concept

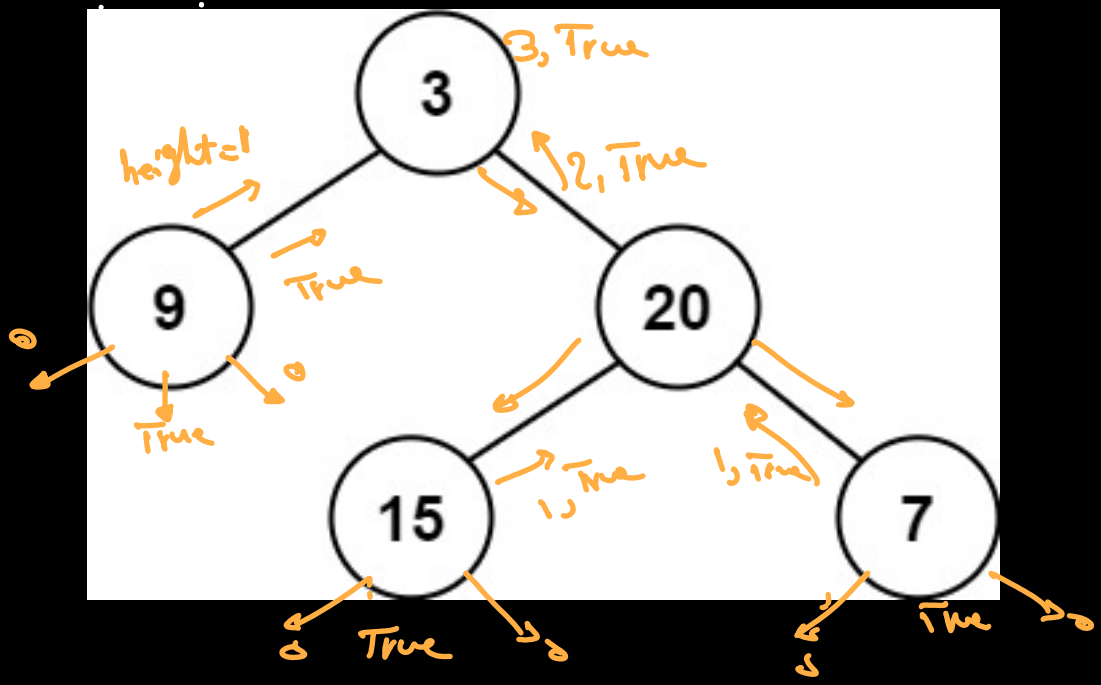
Height Balanced

Given a binary tree, check if it is height balanced.

A non-empty binary tree T is balanced if:

1) Left subtree of T is balanced
2) Right subtree of T is balanced
3) The difference between heights of left subtree and right subtree is not more than 1.

Is this Tree height Balance or not?



two Conditions to be balanced :→

① height of left Tree - height of Right ≤ 1

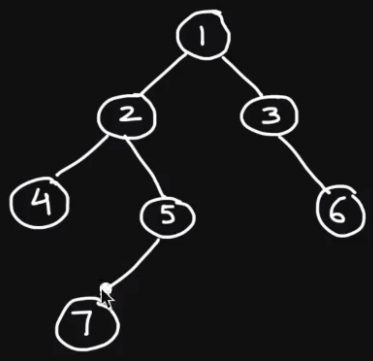
↳ This must be happen For every Sub Tree.

```cpp
int heightCalc(TreeNode *root) {
        if(!root)
           return 0;
        int leftHeight = heightCalc(root->left);
        int rightHeight = heightCalc(root->right);
        return max(leftHeight, rightHeight) + 1;
    }
    bool isBalanced(TreeNode* root) {
        if(!root) return true;
        int leftHeight = heightCalc(root->left);
        int rightHeight = heightCalc(root->right);
        if(abs(leftHeight - rightHeight) > 1)
          return false;
        return isBalanced(root->left) && isBalanced(root->right);
    }
```

* Binary Tree Diameter :→

Diameter

Find the diameter of the binary tree, diameter is defined as the largest distance between any two nodes of the tree



→ Diameter : 5

* The First Case if the Diameter Passes throught the root, then the Diameter Can be Calculated by:→

↳ (left Hight - 1) + (Right Hight - 1) + 2

= Left Light + Right height. → D_1

* The Second Case if it doesn't Pass throught the root and occurs in the left Part ⇒ D_2

* The third Case if it doesn't Pass throught the root and occurs in the Right Part. ⇒ D_3

finalRes = max(D_1, D_2, D_3)

```cpp
int height(TreeNode *root) {
    if(!root) return 0;
    int h1 = height(root->left);
    int h2 = height(root->right);
    return 1 + max(h1, h2);
}

int diameterCalc(TreeNode *root) {
    if(!root) return 0;
    int d1 = height(root->left) + height(root->right);
    int d2 = diameterCalc(root->left);
    int d3 = diameterCalc(root->right);
    return max(d1, max(d2, d3));
}
int diameterOfBinaryTree(TreeNode* root) {
    if(!root) return 0;
    return diameterCalc(root);
}
```

1st approach

$O(n^2)$

optimized approach :→

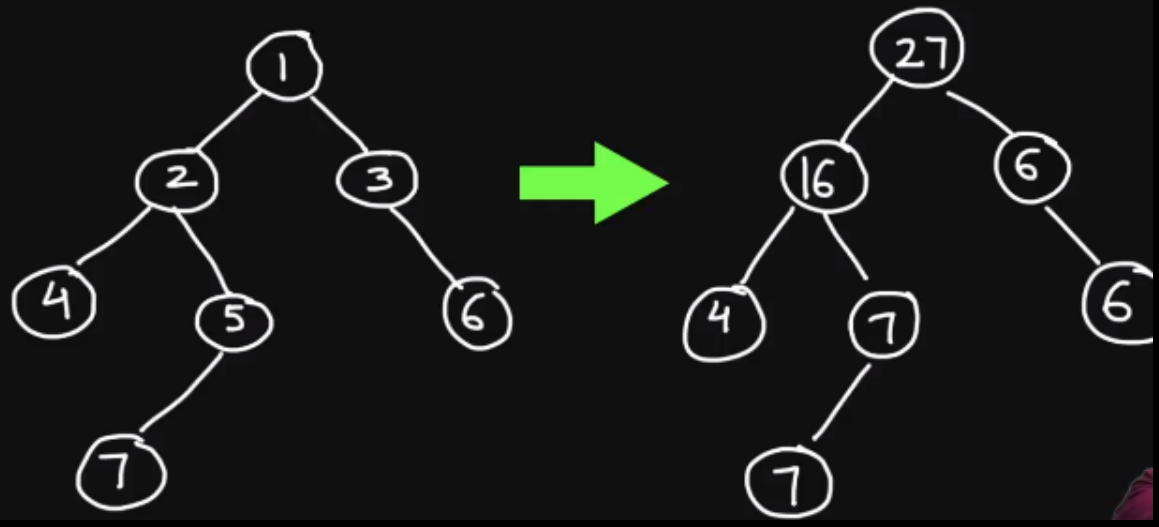
```cpp
pair<int, int> treeDiameterCalc(TreeNode *root) {
    pair<int, int> p, leftTree, rightTree;
    if(!root) {
        p.first = 0; // First -->> height
        p.second = 0; // Second -->> Diameter
        return p;
    }

    leftTree = treeDiameterCalc(root->left);
    rightTree = treeDiameterCalc(root->right);
    p.first = max(leftTree.first, rightTree.first) + 1;
    int D1 =  leftTree.first + rightTree.first;
// Diameter Passing Throught The Root = (LeftHeight + 1) + (right Height + 1) - 2 For Edges
    int D2 =  leftTree.second; // LeftTree Diameter
    int D3 = rightTree.second; // RightTree Diameter
    p.second = max(D1, max(D2, D3));
    return p;
}
int diameterOfBinaryTree(TreeNode* root) {
    return treeDiameterCalc(root).second;
}
```

* By this approach it will be solved By $O(N)$ Time.

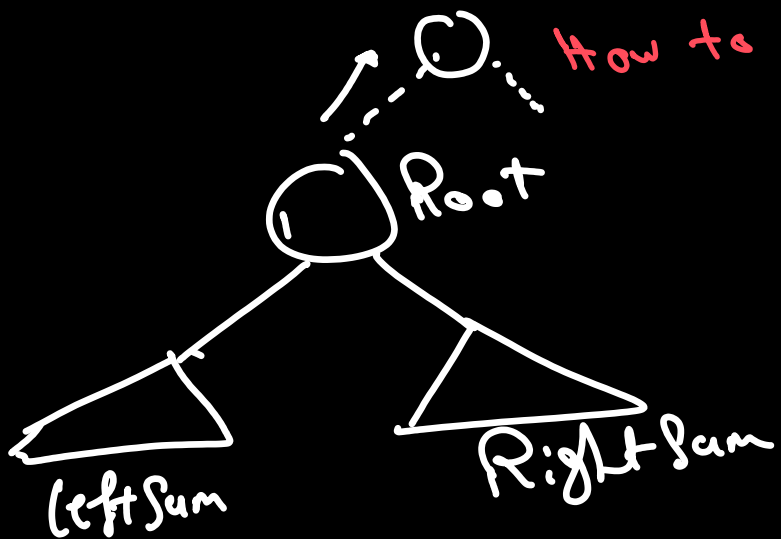
* It simply make the P(root node) to store two values the first for height and Diameter.

Replace with Sum

Given a binary tree, replace every node by sum of all it descendants, leave leaf nodes intact.



How to approach :→



⟹ temp = root → data;

root → Data = leftSum + RightSum

return temp + root → data

```
int replaceWithSum(node *root) {
    // Base Case
    if(!root)
       return 0;
    if(!root->left && !root->right)
      return root->data;
    // Recursive Case
    int leftSum = replaceWithSum(root->left);
    int rightSum = replaceWithSum(root->right);
    int temp = root->data;
    root->data =  leftSum + rightSum;
    return root->data + temp;
}
```

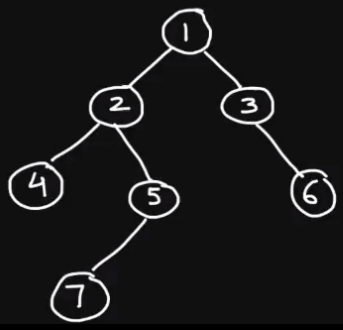
→ We modify root → data = LS + RS

→ But Before that we Store the original Data of root.

→ When we going to return data, we will return root → data + original Data of the Root.

Max Subset Sum

Find the largest sum of a subset of nodes in a binary tree, such that if a node is included in the sum then its parent and children must not be included in the subset sum.



Max Sum 18

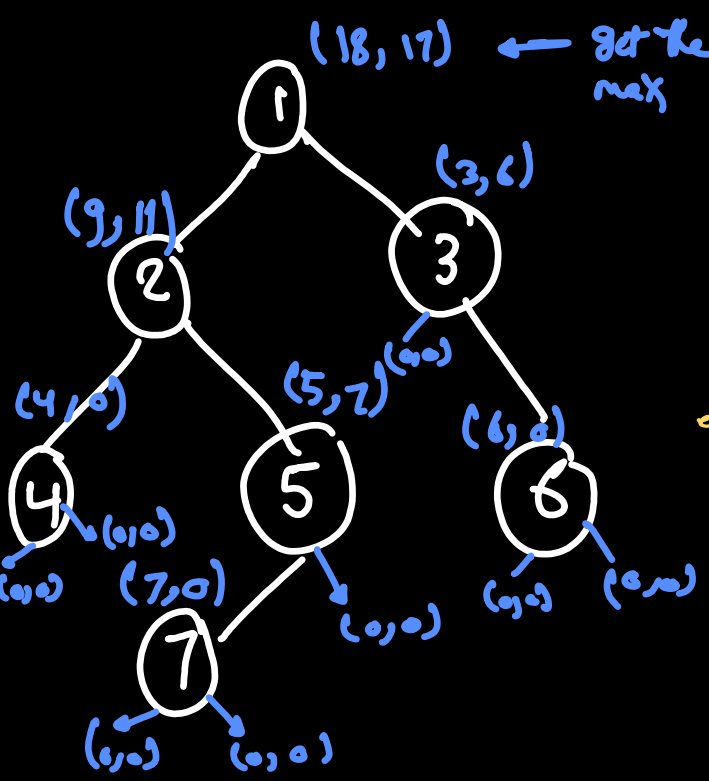
How to approach:→

inc = root→data + L.exc + R.exc

This tells us if we include the current Root node we must exclude the left and Right one.

exc = max(L.inc, L.exc) + max(R.inc, R.exc)

This tells us if we exclude the current Root node we may or may not include the left so we have to determine if we include the left node this will generate maximum sum or maximum sum will be generated if we exclude the Current left and the Right is similar.



* Simple Dry Run of the above Recurrence

nCr root→data + L.exc + R.exc

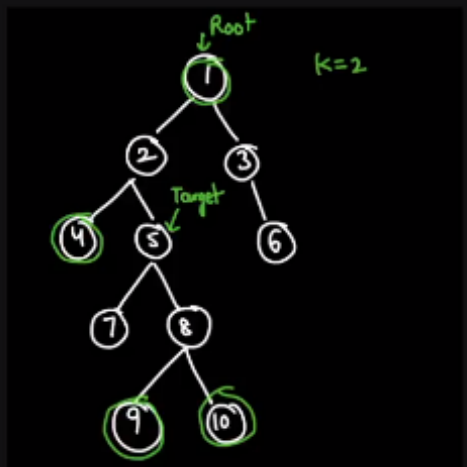
exc = max(L.inc, L.exc) + max(R.inc, R.exc)

(include, exclude)

```cpp
pair<int, int> maxSubsetTree(node *root) {
    pair<int, int> p, leftSubTree, rightSubTree;
    if(!root) {
        p.first = 0; // first ->> inc
        p.second = 0; // second -->> exec
        return p;
    }
    leftSubTree = maxSubsetTree(root->left);
    rightSubTree = maxSubsetTree(root->right);
    p.first = root->data + leftSubTree.second + rightSubTree.second;
    p.second = max(leftSubTree.first, leftSubTree.second ) + max(rightSubTree.first, leftSubTree.first);
    return p;
}

```

Nodes at Distance K

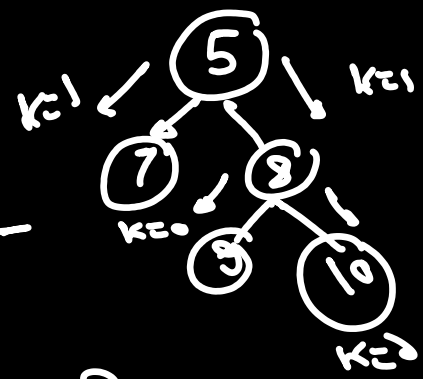
Given a Binary Tree, and a target Node T. Find all nodes distance K from given node, where K is also an integer input.



How to approach :→

*First Part :→

We can apply simple traversal to target→left and for target→Right untill the k=0 this means we reached the targetted nodes.

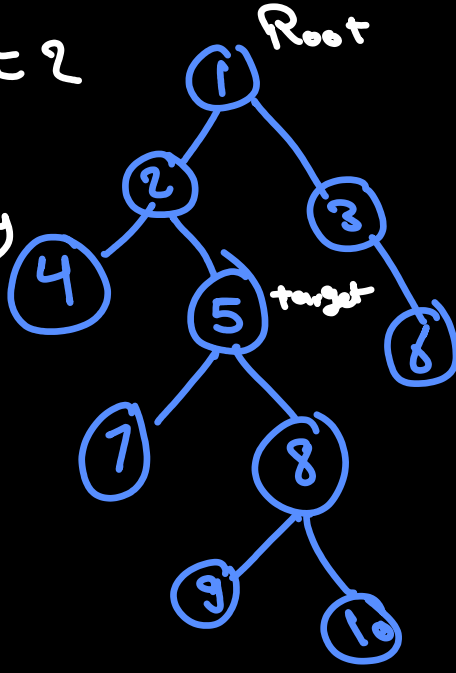


* So when (root == null) return.

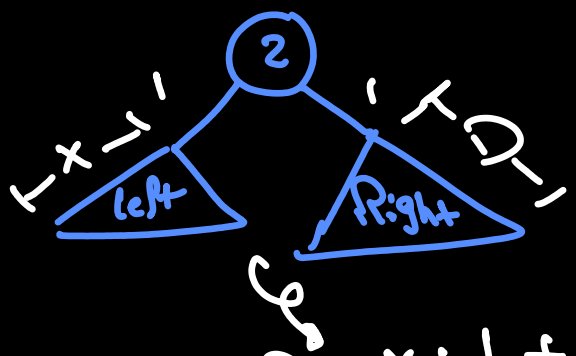
* k==0 → push these values to Res vector.

* Second Part (For nodes that are not a child of the target node) :→

2-I



* From root to target we will pass [1, 2, 5]
  ↑ ↳ Calculated Previously



$D + X + 1 + 1 = K$

↳ we want to calculate X which is the level after one path of ②

$\Rightarrow X = K - D - 2$

$D = 0$ in this case

$\Rightarrow X = K - 2$

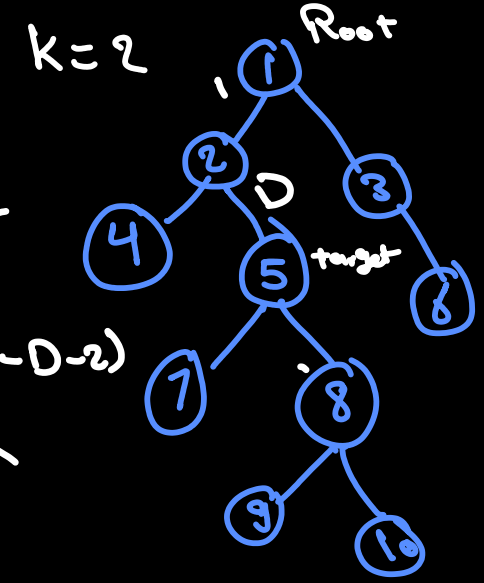
-2

$X = 0$ in this case

↳ means that you should print this

* Part two II :→

[1, 2, 5]



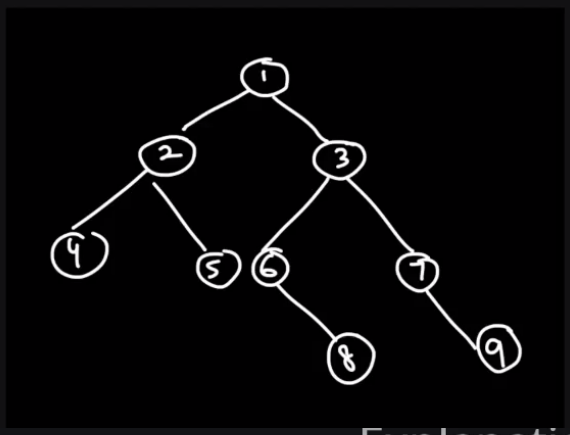
if (D + 1 == K) ← we reach the Root
Push the root to res vector

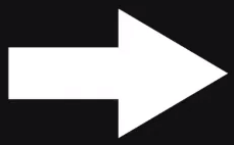
else {
if (target on left) → call on Right (K-D-2)
if (target on Right) → call on the left (K-D-2)
}

```cpp
void printAtLevelK(TreeNode *root, int k, vector<int> &finalRes) {
    if(!root) return;
    if(k == 0)
      finalRes.push_back(root->val);
    printAtLevelK(root->right, k - 1, finalRes);
    printAtLevelK(root->left, k - 1, finalRes);
    return;
}

int mainDistanceCalc(TreeNode *root, TreeNode* target, int k, vector<int> &finalRes) {
    // Base Case
    if(!root) {
        return -1;
    }

    // Recursive Case ->> First We Reach the target node ->> PrintAtLevek K Deep
    if(root == target) {
        printAtLevelK(target, k , finalRes);
        return 0;
    }


    // The Second Part - First ->> if the target on the left
    int DL = mainDistanceCalc(root->left, target, k , finalRes);
    if(DL != -1) {
        // Check If we Reach the Root or make Rec Call On othe Direction
        if(DL + 1 == k)
          finalRes.push_back(root->val);
        else
          printAtLevelK(root->right, k - DL - 2, finalRes);
        return 1 + DL;
    }


    // The Second Part - Second ->> if the target on the right
    int DR = mainDistanceCalc(root->right, target, k , finalRes);
    if(DR != -1) {
        // We Reach the root or make recursive call on the right subtree
        if(DR + 1 == k)
          finalRes.push_back(root->val);
        else
          printAtLevelK(root->left, k - DR - 2, finalRes);
        return 1 + DR;
    }
    return -1;
}
```

Vertical Order Print

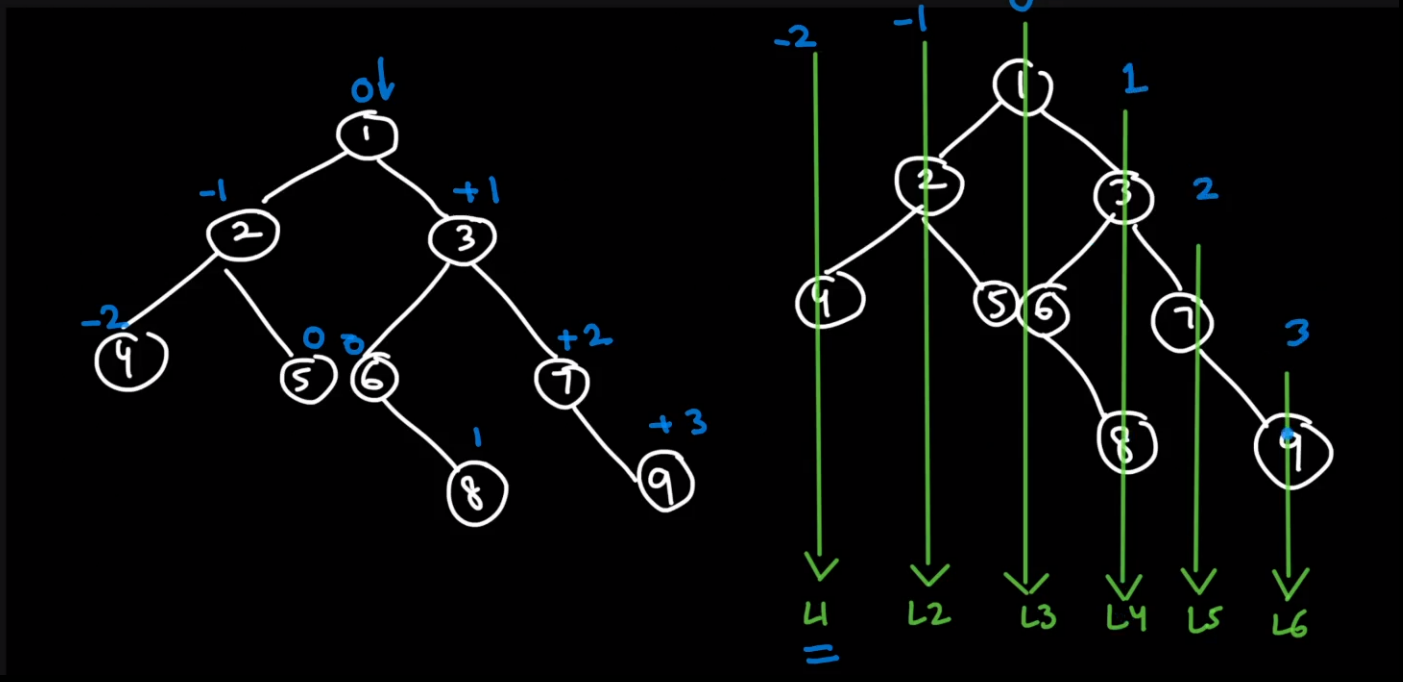
Given a Binary Tree, we need to print it vertically, refer example for explanation.





```
4
2
1  5  6
3  8
7
9
```

Explanation

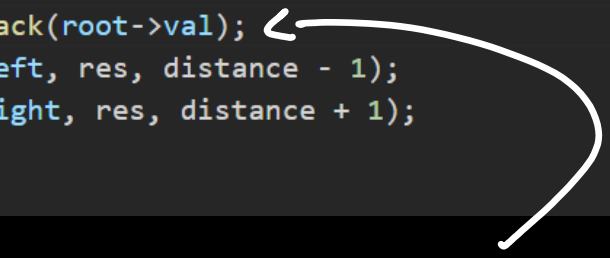


we will make a map (int, vector)

distance

values

```cpp
void traverseTree(TreeNode *root, map<int, vector<int>> &res, int distance) {
    if(!root)
      return;
    res[distance].push_back(root->val);
    traverseTree(root->left, res, distance - 1);
    traverseTree(root->right, res, distance + 1);
    return;
}
```



associate every distance with the corresponding node values.