**Queue**

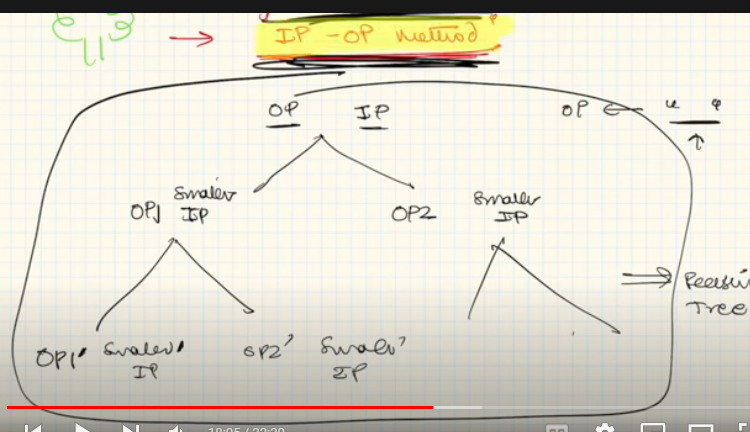
Priority Queueis an abstract data type, which is similar to a queue, however, in the priority queue, every element has some priority. The priority of the elements in a priority queue determines the order in which elements are removed from the priority queue. Therefore all the elements are either arranged in an ascending or descending order.

For more info - https://www.geeksforgeeks.org/priority-queue-set-1-introduction/

**Recursion**

In recursion, input doesn’t get smaller, we take decision due to which the problem gets smaller. Based on our decision our problem gets smaller. Recursion is applied when there is a combination of choice and decision.

In recursion we create recursive tree, combination of input –output is called node and number of branches is the number of choices ,last node in a tree which does not have any children is called leaf node.

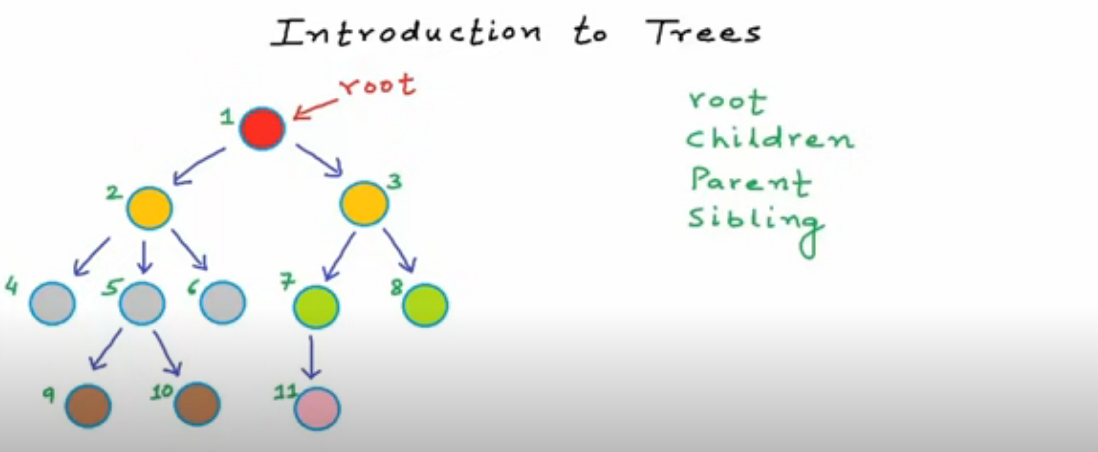


There is a method to solve recursive problems, it consists of three things:

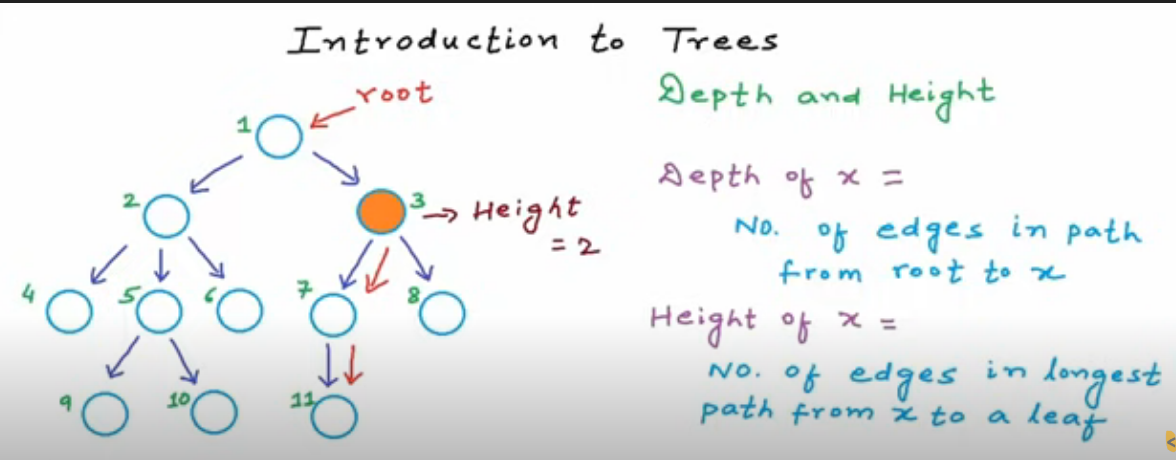
1. Hypothesis 🡪 the design / declaration/ behavior of your function
2. Induction 🡪 the main logic of the function
3. Base Condition 🡪find the smallest valid input

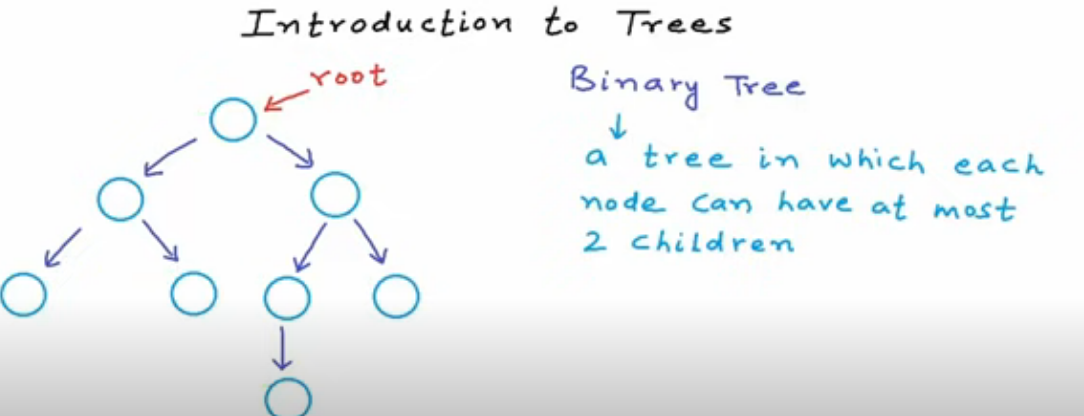
**TREE**

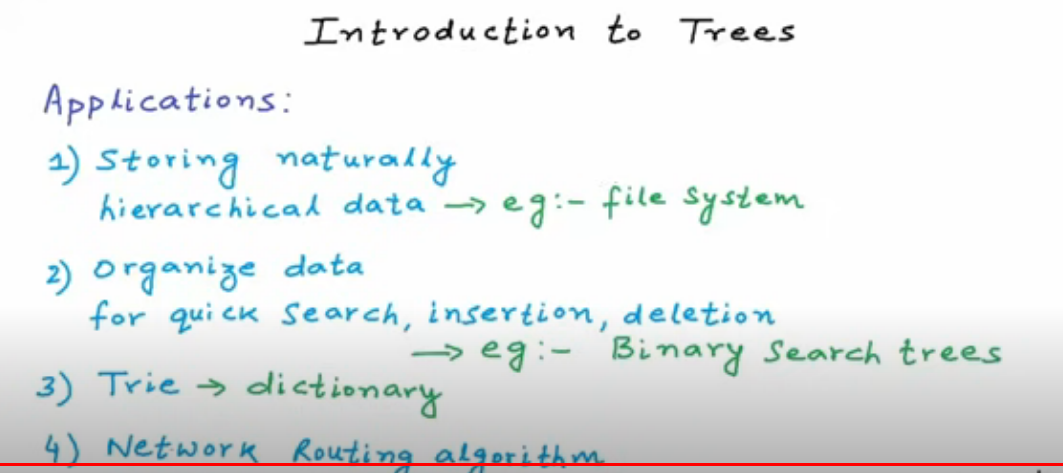
Tree is used to store hierarchical data. It is a non linear/recursive data structure. It used combination of nodes to represent the hierarchy. Nodes which does not have a child is called leaf node. Parent Node is called root node. In a tree with N nodes , there will be n-1 edges.











<https://www.geeksforgeeks.org/binary-tree-set-1-introduction/>

Types of binary tree

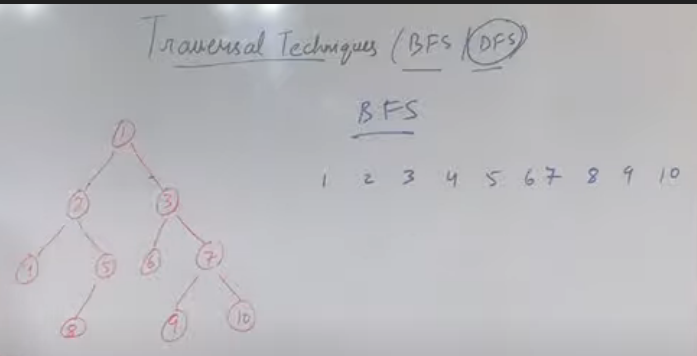
1. Full binary tree – Each node has two or 0 children
2. Complete Binary Tree – all levels are completely filled except the last level

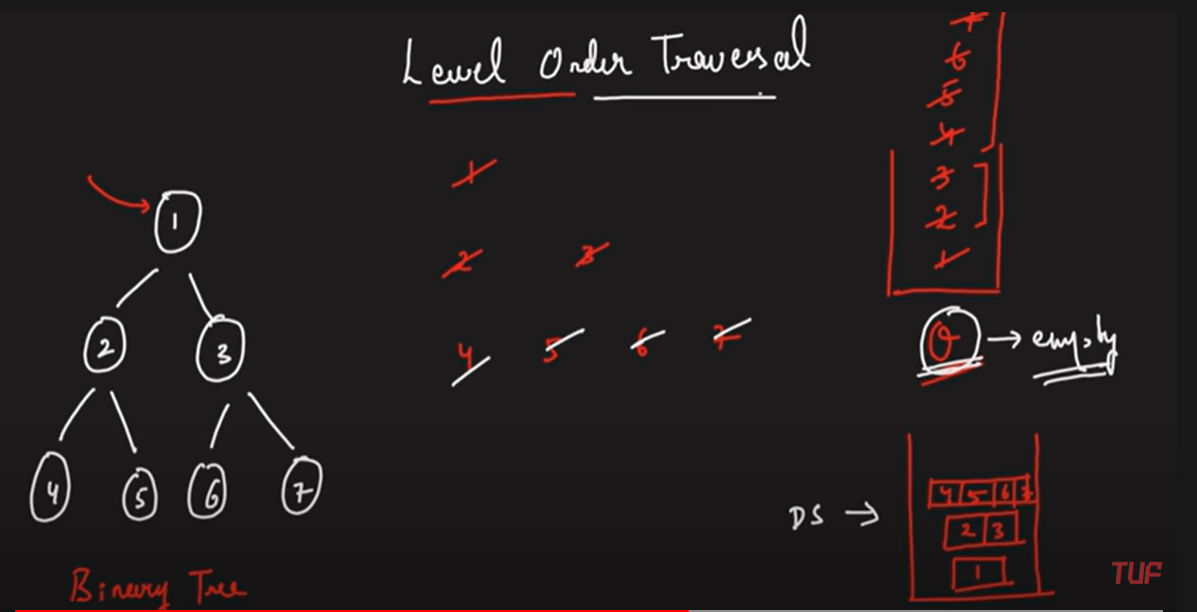
* The last level has all the nodes as left as possible.

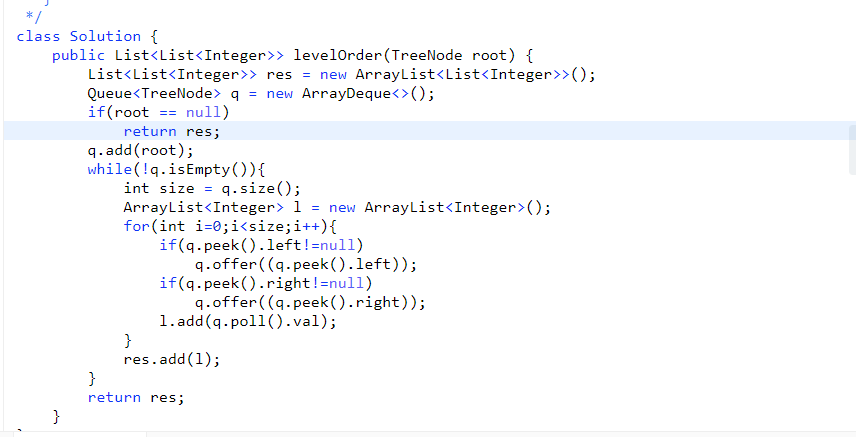
1. Perfect Binary Tree - all leaf nodes are at the same level.
2. Balanced Tree - Height of tree at max would be log (N) where N = number of nodes
3. Degenerate Tree -- This is a skewed tree (similar to linked List)

There are two techniques for Tree Traversal:

1. BFS(Breadth first search) – this technique traverse the tree in level wise

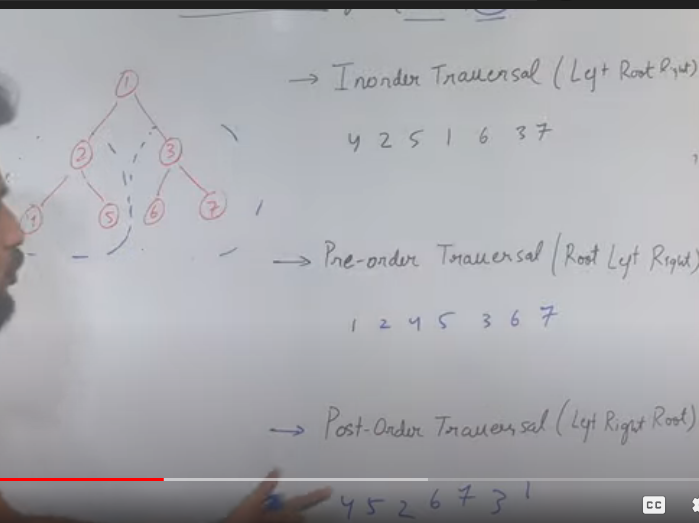




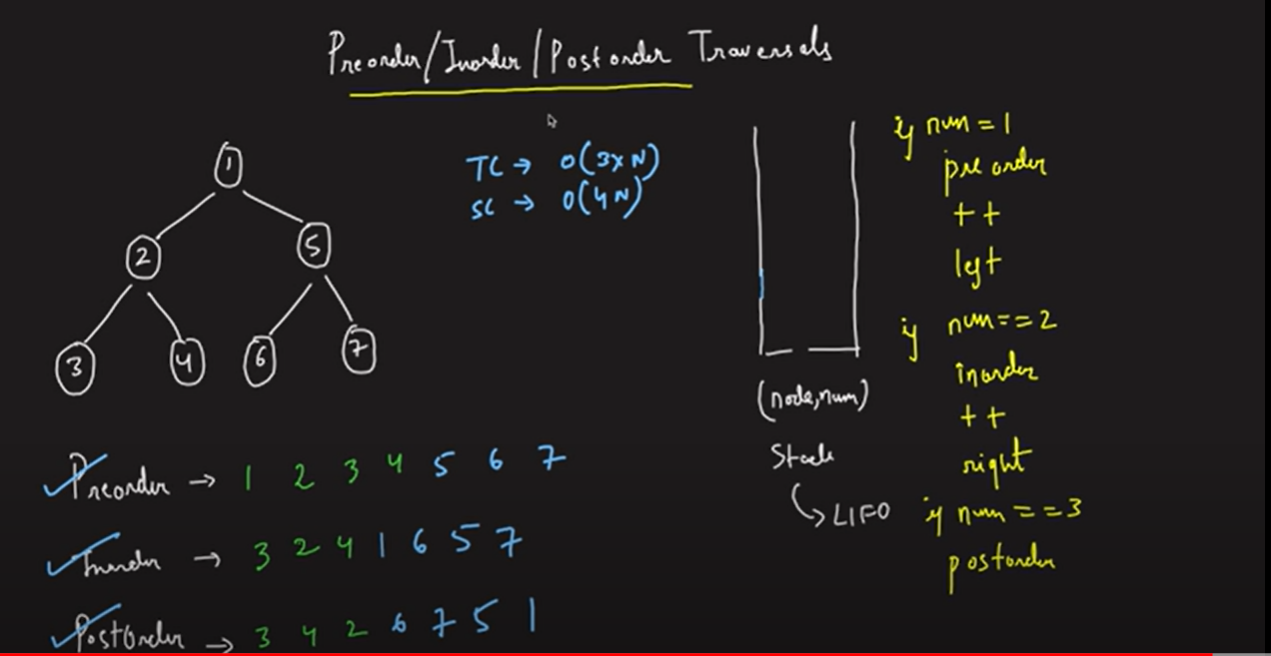


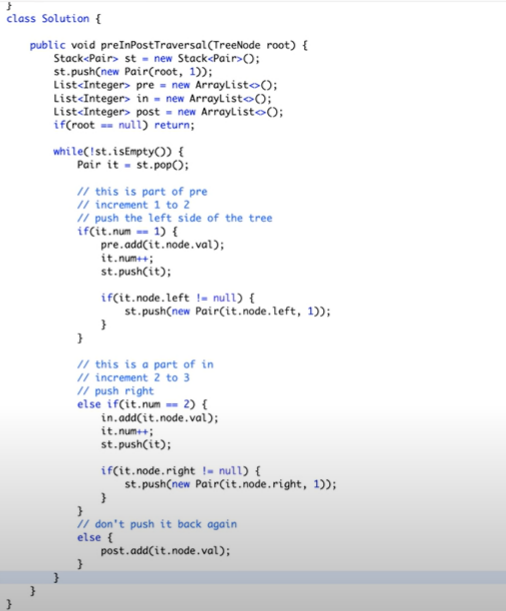
1. DFS(Depth first search) –this technique is of three types
   1. Inorder traversal (left root right)
   2. Pre order traversal (root left right) – time complexity is O(N) where N is number of nodes of a tree.
   3. Postorder traversal(left right root)

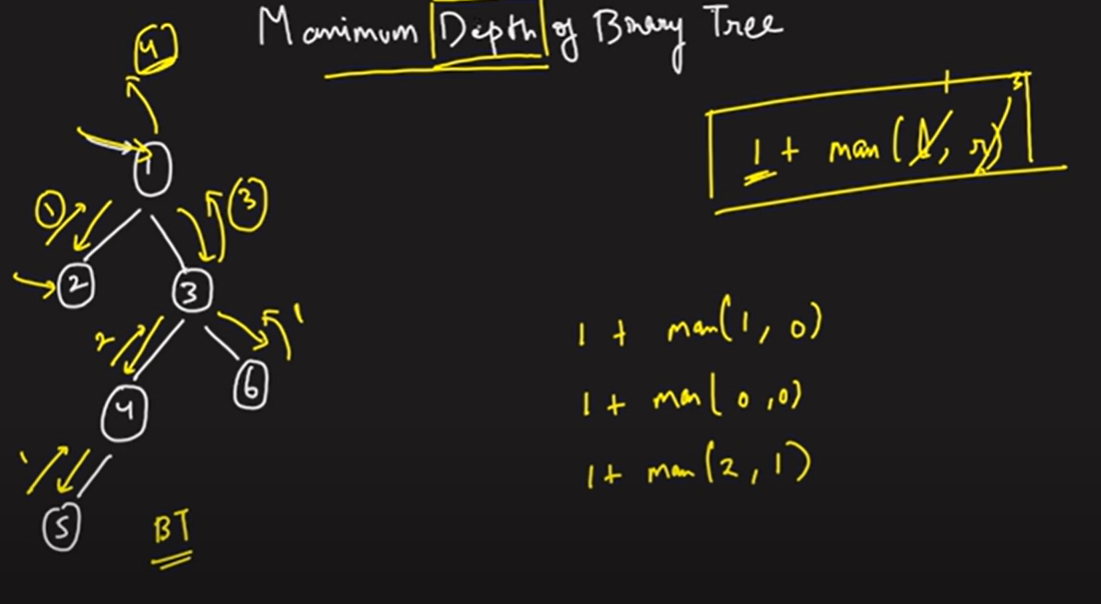
Naming convention is as follows : in inorder (root is between),in preorder traversal (root is first), in postorder traversal (root is at last).

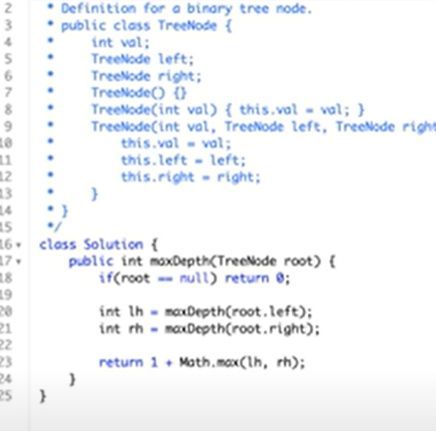


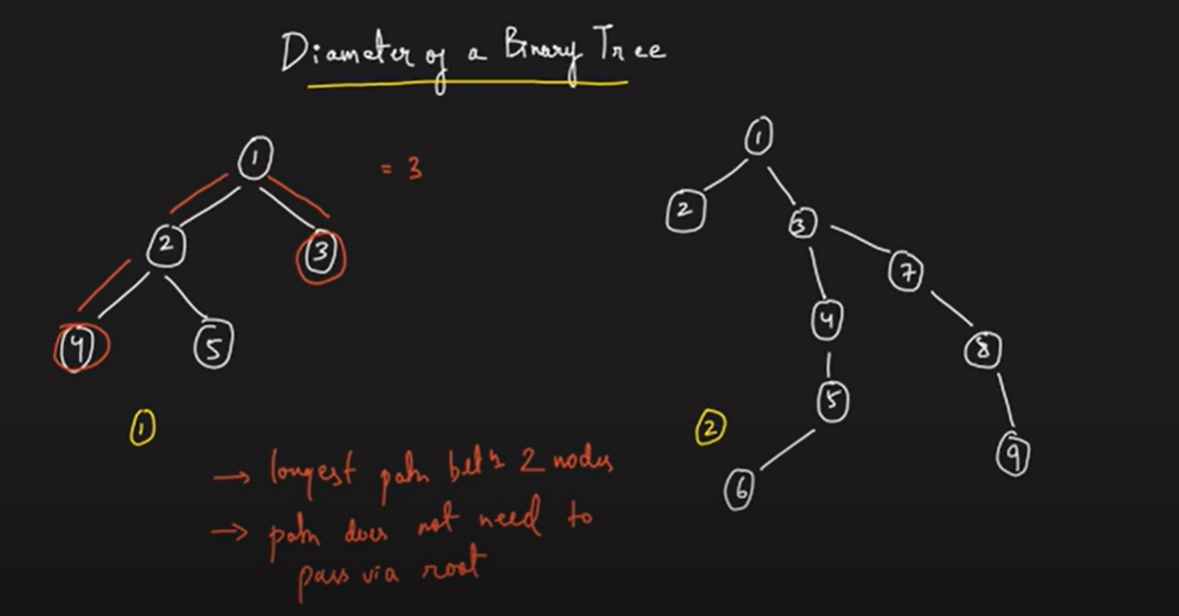
**Inorder ,preorder and post order in a single loop**

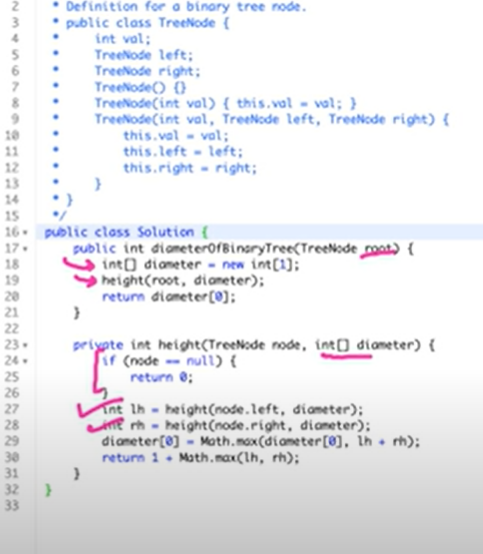




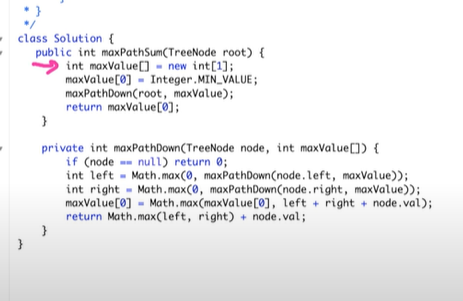








**Maximum path sum in a binary tree**

****

**Vertical order traversal(Important)**

**Algo**

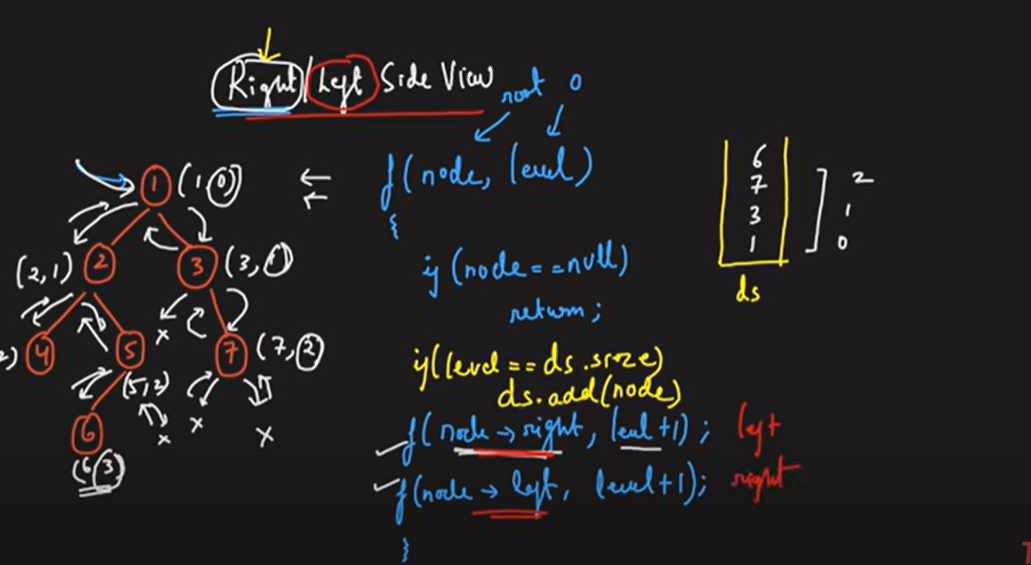
1. **Using level order traversal, store(create new class with node and row,column as attribute) node values along with the the row and column.**
2. **Sort this using Collections.sort(list,com)**

**Code is submitted at the below path:**

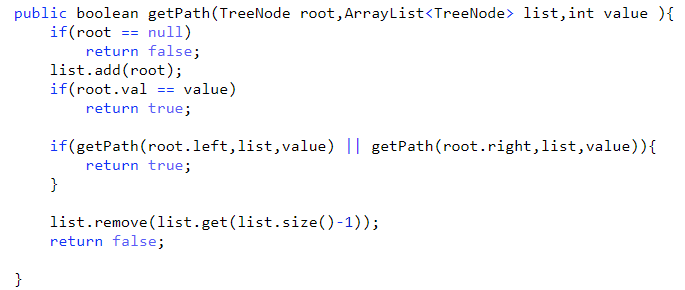
<https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/>

important for top and bottom view of binary tree

Right/left view of binary tree



Find path from root to any node in a binary tree



**GRAPH**

It is a non-linear data structure, which consists of nodes(vertex) and edges. Edges connects two nodes.

There are two types of graphs:

1. Directed Graph
2. Undirected Graph

**Dynamic Programming**

Dynamic Programming is a combination of Recursion + Storage. DP ‘s parent is recursion. For every problem first write recursive solution , then memorize it and then create top-down approach.

How to identify dynamic programming questions

1. When a choice is given i.e whether the thing is to be considered in the solution.
2. When optimal solution is required, i.e minimum, maximum, largest, greatest For example – find maximum profit etc.

**Knapsack Problem**

In this problem , weight and values of some items are given and capacity of a bag is given, we have to add the items in a bag, so that it has maximum value, constraint is the weight of that bag.

There are two types of knapsack

1. 0-1 knapsack – in which items can be only be added as individual
2. Fractional knapsack – in which items can be broken and added in that bag as per the weight accordingly.
3. Unbounded knapsack – in this we can add multiple occurrences of that single item unlike 0-1 knapsack where one item can be added only once. There is unlimited supply of a single item.

**In order to write base condition for recursion, we have to find the smallest valid input.**

For example , we have if there is no item or the bag capacity is zero or the bag is full, then base condition is equal to

if(n==0 || W==0)

return 0**;**

Here n is no. of items in the bag and W is capacity of the bag.

Memoization is the top-down approach to solving a problem with dynamic programming. It's called memoization because we will create a memo, or a “note to self”, for the values returned from solving each problem.

Sudo Code for knapsack

//create an double dimensional array of size greater than the capacity

// this array will store the maximum result for that particular array size and capacity

int dp[][] = new int[array\_size][ capacity]

//initalise this array with each value as -1

int knapsack(int weight[], int value[], int capacity, int array\_size){

//base condition would be decided as mentioned above

if(array\_size ==0 || capacity ==0)

return 0;

//if value in dp is not equal to -1 , then we will return

If(dp[array\_size][ capacity] != -1)

return dp[array\_size][ capacity];

//here we will check whether that item has weight less than or greater than the capacity

// if weight is lesser than capacity ,we have two choices whether to consider that item or not

if(weight[n-1] < = capacity)

return dp[array\_size][ capacity] = max(value[n-1] + knapsack(weight[],value[],capacity – weight[n-1], array\_size-1), knapsack (weight[],value[],capacity ,array\_size-1));

//if weight is greater than the knapsack capacity, then we will not consider it.

else if(weight[n-1] > capacity)

return dp[array\_size][ capacity] = knapsack (weight[],value[],capacity ,array\_size-1);

}

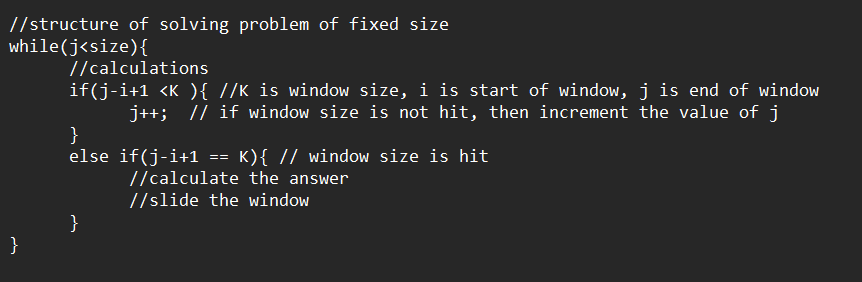
**Sliding Window Algorithm**

It is mostly used in array or string where subarray or substring is mentioned and window size is given or largest or smallest value/sum is given. It is basically an optimization method.

Example - <https://practice.geeksforgeeks.org/problems/max-sum-subarray-of-size-k5313/1>

They are of 2 types:

1. Fixed Size window



Example - <https://practice.geeksforgeeks.org/problems/count-occurences-of-anagrams5839/1> very good approach to solve problem

Size of window can be calculated by j-i+1

1. Variable Size window

