Question
https://leetcode.com/problems/path-sum-ii/ (https://leetcode.com/problems/path-sum-ii/)

TC: O(n) n = no. of nodes in the tree
SC: O(h) h = max height of the tree

In [ ]:

```
class Solution {
public:
    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        vector<int> currPath;
        vector<vector<int>> results;

        pathSumUtil(root, results, currPath, 0, targetSum);
        return results;
    }

    void pathSumUtil(TreeNode* root, vector<vector<int>> &results, vector<int>
        if (root == NULL) {
            return;
        }

        currSum += root->val;
        currPath.push_back(root->val);
        if (root->right == NULL && root->left  == NULL) {
            if (currSum == targetSum) {
                results.push_back(currPath);
            }
        } else {
            pathSumUtil(root->left, results, currPath, currSum, targetSum);
            pathSumUtil(root->right, results, currPath, currSum, targetSum);
        }
        currPath.pop_back();
    }
};
```

In [ ]:

Right view:
https://leetcode.com/problems/binary-tree-right-side-view/
(https://leetcode.com/problems/binary-tree-right-side-view/)

Solution-1 BFS:

1. In BFS solution keep additional Node pointer
2. Update the pointer each time we pop from queue (update when not null)
3. When we get the level change marker i.e. NULL, update the result with the value in pointer.

Solution-2 DFS: 1. Preorder 2. Inorder 3. PostOrder

rRL

In [ ]:

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> ans = new ArrayList<Integer>();
        find(root,ans,0);
        return ans;
    }
    public void find(TreeNode root, List<Integer> ans, int level){
        if(root == null)return;

        if(level == ans.size()) ans.add(root.val);

        find(root.right,ans,level+1);
        find(root.left,ans,level+1);
    }
}
```

In [ ]:

```
class Solution {
    List<Integer> res = new ArrayList<>();
    public List<Integer> rightSideView(TreeNode root) {
        level(root, 0);
        return res;
    }
    public void level(TreeNode node, int level){
        if(node == null)
            return;
        if(res.size() <= level){
            res.add(node.val);
        }
        level(node.right, level+1);
        level(node.left, level+1);
    }
}
```

In [ ]:

```
class Solution {
    int maxlevel = 1;
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> list  = new ArrayList<>();
        right(root,2,list);
        return list ;
    }
    void right(TreeNode root,int level,List<Integer> list){
        if(root==null){
            return ;
        }
        if(maxlevel<level){
            list.add(root.val);
            maxlevel=level;
        }
        right(root.right,level+1,list);
        right(root.left,level+1,list);

    }
}
```

In [ ]:

In [ ]:

Top View
https://www.hackerrank.com/challenges/tree-top-view/problem (https://www.hackerrank.com/challenges/tree-top-view/problem)

In [ ]:

```
class Node:
    def __init__(self, info):
        self.info = info
        self.left = None
        self.right = None
        self.level = None

    def __str__(self):
        return str(self.info)

class BinarySearchTree:
    def __init__(self):
        self.root = None

    def create(self, val):
        if self.root == None:
            self.root = Node(val)
        else:
            current = self.root

            while True:
                if val < current.info:
                    if current.left:
                        current = current.left
                    else:
                        current.left = Node(val)
                        break
                elif val > current.info:
                    if current.right:
                        current = current.right
                    else:
                        current.right = Node(val)
                        break
                else:
                    break

"""
Node is defined as
self.left (the left child of the node)
self.right (the right child of the node)
self.info (the value of the node)
"""
def topView(root):
    #Write your code here

    mp = {}

    topViewUtil(root, mp, 0, 0)

    res = []
    for i in range(min(mp.keys()), max(mp.keys) + 1):
        res.append(mp[i][0])

    return res


def topViewUtil(root, mp, level, pos):
    if root is None:
        return

    if pos not in mp:
        mp[pos] = (root.info, level)
    else:
        if mp[pos][1] > level:
            mp[pos] = (root.info, level)

    topViewUtil(root.left, mp, level + 1, pos-1)
    topViewUtil(root.right, mp, level + 1, pos+1)


tree = BinarySearchTree()
t = int(input())

arr = list(map(int, input().split()))

for i in range(t):
    tree.create(arr[i])

topView(tree.root)
```

In [ ]:

In [ ]:

Bottom view

https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1 (https://practice.geeksforgeeks.org/problems/bottom-view-of-binary-tree/1)

In [ ]:

Vertical Traversal

https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/ (https://leetcode.com/problems/vertical-order-traversal-of-a-binary-tree/)

In [ ]:

In [ ]:

DIY

LCA: Lowest common ancestor

https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/ (https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/)

In [ ]: