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

C++

```
class Solution {
public:
    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        // 1. Current sum
        // 2. the current path
        vector<vector<int>> res;
        vector<int> currPath;

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

    void pathSumUtil(TreeNode *root, int targetSum, int currSum, vect
or<int> &currPath, vector<vector<int>> &res) {
        if (root == NULL)
            return;

        currSum += root->val;
        currPath.push_back(root->val);

        if (root->left == NULL && root->right == NULL) {
            if (currSum == targetSum) {
                res.push_back(currPath);
            }

            currPath.pop_back();
            return;
        }

        pathSumUtil(root->left, targetSum, currSum, currPath, res);
        pathSumUtil(root->right, targetSum, currSum, currPath, res);

        currPath.pop_back();
    }
};
```

THis is for a variation of the problem

```
class Solution {
public:
    vector<vector<int>> pathSum(TreeNode* root, int targetSum) {
        // 1. Current sum
        // 2. the current path
        vector<vector<int>> res;
        vector<int> currPath;

        pathSumUtil(root, targetSum, currPath, res);
        return res;
    }

    void pathSumUtil(TreeNode *root, int targetSum, vector<int> &curr
Path, vector<vector<int>> &res) {
        if (root == NULL)
            return;

        targetSum -= root->val;

        if (targeSum < 0) {
            return;
        }
        currPath.push_back(root->val);

        if (root->left == NULL && root->right == NULL && targetSum ==
0) {
            res.push_back(currPath);
            currPath.pop_back();
            return;
        }

        pathSumUtil(root->left, targetSum, currPath, res);
        pathSumUtil(root->right, targetSum, currPath, res);

        currPath.pop_back();
```

In [ ]:
```
1
```

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

Level Order Traversal: Python

```
    res = [ ]
            q = collections.deque()
            q.append(root)
            while q:
                rightside = None
                l = len(q)
                for i in range(l):
                    node = q.popleft()
                    if node:
                        rightside = node
                        q.append(node.left)
                        q.append(node.right)
                if rightside:
                    res.append(rightside.val)

            return res
```

Level Order: java

```
class Solution {
```

Depth Order: C++

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), lef
t(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<int> rightSideView(TreeNode* root) {
        vector<int> res;
        rightSideViewUtil(root, 1, res);
        return res;
    }

    void rightSideViewUtil(TreeNode* root, int level, vector<int> &re
s) {
        if (root == NULL) return;

        if (res.size() < level ) {
            res.push_back(root->val);
        } else {
            res[level-1] = root->val;
        }

        rightSideViewUtil(root->left, level+1, res);
        rightSideViewUtil(root->right, level+1, res);
    }

};
```

Depth Order: C++

```
    vector<int> rightSideView(TreeNode* root) {
        vector<int> res;
        rightSideViewUtil(root, 1, res);
        return res;
    }

    void rightSideViewUtil(TreeNode* root, int level, vector<int> &re
s) {
        if (root == NULL) return;

        if (res.size() < level ) {
            res.push_back(root->val);
        }
```

In [ ]:
```
1
```

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

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 [ ]:
```
1
```

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

Solution-1: C++

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        // pos=col, level=row
        //  <pos , array <level, value>   >
        map<int, vector< pair<int, int>  >> mp;
        vector<vector<int>> res;

        // populate the map
        verticalTraversalUtil(root, 0, 0, mp);

        // iterate values of map ordered by key (pos) in ascending or
der
        for(auto it=mp.begin(); it != mp.end(); it++) {

            auto temp = it->second ; //vector=array with values <leve
l, val>
            std::sort(temp.begin(), temp.end()); // sort by level; br
eak ties by value

            // example-1
            // -1 : [ (1,9)]
            //  0 : [ (0,3), (2,15)]
            //  1 : [ (1,20)]
            //  2:  [ (2,7)]

            // example-2
            // 0: [ (0,1), (2,5), (2,6)]

            // example-3
            // 0: [ (0,1), (2,6), (2,5)] -> [ (0,1), (2,5), (2,6)]

            // add only values to the result
            vector<int> col;
            for(auto it1=temp.begin(); it1!=temp.end(); it1++) {
                col.push_back(it1->second);
            }
            res.push_back(col);

        }
```

In [ ]:

```
// Solution-2
// (pos, level, value)
// [ (0,0,3), (-1,1,9), (1,1,20),(0,2,15), (2,2,7)]
// [ (-1,1,9), (0,0,3), (0,2,15), (1,1,20), (2,2,7)]
// -1 : 9
// 0  : [3,15]
// 1  : [20]
// 2  : [7]
```

Solution-2

```
class Solution {
public:
    vector<vector<int>> verticalTraversal(TreeNode* root) {
        // pos=col, level=row
        //         (pos, level, value)
        vector< tuple<int, int, int>  > data;
        vector<vector<int>> res;

        // populate the map
        traverse(root, 0, 0, data);
        // [ (0,0,3), (-1,1,9), (1,1,20),(0,2,15), (2,2,7)]
```

Solution-3: Level Order: Python

```
class Solution:
    def verticalTraversal(self, root: Optional[TreeNode]) -> List[Lis
t[int]]:

        if root is None:
            return []

        import queue

        q = queue.Queue() # each element = <pos, value>
        mp = {} # map <pos, array < (level,value) > >

        q.put((root,0))

        min_pos = 0
```