Assignment 2

[Maximum Depth of Binary Tree - LeetCode](https://leetcode.com/problems/maximum-depth-of-binary-tree/description/)

/\*\*

 \* 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), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

    int maxDepth(TreeNode\* root) {

        if(root == NULL)

        return 0;

        int lh = maxDepth(root->left);

        int rh = maxDepth(root->right);

        return 1+max(rh,lh);

    }

};

[Maximum Level Sum of a Binary Tree - LeetCode](https://leetcode.com/problems/maximum-level-sum-of-a-binary-tree/submissions/1141816794/)

/\*\*

 \* 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), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

    int maxLevelSum(TreeNode\* root) {

        if(root==NULL)

        return 0;

        int sum = 0;

        queue<TreeNode\*> qsol;

        qsol.push(root);

        TreeNode\* temp = root;

        int max = root->val-1 ;int level = 0; int maxlevel = 0;

        while(!qsol.empty())

        {

            int size = qsol.size();

            sum = 0;

            for(int i =0;i<size;i++){

                temp = qsol.front();

                qsol.pop();

                if(temp->left!=NULL)qsol.push(temp->left);

                if(temp->right!=NULL)qsol.push(temp->right);

                sum+=temp->val;

            }

            level++;

            if(sum>max)

            {

                max = sum;

                maxlevel = level;

            }

        }

        return maxlevel;

    }

};

[Sum Root to Leaf Numbers - LeetCode](https://leetcode.com/problems/sum-root-to-leaf-numbers/submissions/1132112463/)

/\*\*

 \* 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), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

    int sum(TreeNode\* root, int num)

    {

        if(root==NULL)

        return 0;

        num = num\*10 + root->val;

        if(root->left==NULL&& root->right==NULL)

        return num;

        return sum(root->left,num) + sum(root->right,num);

    }

    int sumNumbers(TreeNode\* root) {

        return sum(root,0);

    }

};

[Find Bottom Left Tree Value - LeetCode](https://leetcode.com/problems/find-bottom-left-tree-value/submissions/1141302664/)

/\*\*

 \* 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), left(left), right(right) {}

 \* };

 \*/

class Solution {

public:

    int findBottomLeftValue(TreeNode\* root) {

        if(root==NULL)

        return 0;

        vector<vector<TreeNode\*>> ans;

        queue<TreeNode\*> qsol;

        qsol.push(root);

        TreeNode\* temp = root;

        while(!qsol.empty())

        {

            int size = qsol.size();

            vector<TreeNode\*> level;

            for(int i =0;i<size;i++){

                temp = qsol.front();

                qsol.pop();

                if(temp->left!=NULL)qsol.push(temp->left);

                if(temp->right!=NULL)qsol.push(temp->right);

                level.push\_back(temp);

            }

            ans.push\_back(level);

        }

        int sizevec = ans.size();

        for(int i =0;i<ans[sizevec-1].size();i++)

        {

            if(ans[sizevec-1][i]!=NULL)

            return ans[sizevec-1][i]->val;

        }

        return 0;

    }

};