Lecture 36 Binary Trees

1. <https://leetcode.com/problems/minimum-depth-of-binary-tree/>

class Solution {

public:

int minDepth(TreeNode\* root) {

if(root==NULL)

{

return 0;

}

queue<TreeNode\*> q;

q.push(root);

int minLevel = 0;

while(!q.empty())

{

minLevel++;

int n = q.size();

for(int i=1; i<=n; i++)

{

TreeNode\* f = q.front();

q.pop();

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

{

return minLevel;

}

if(f->left)

{

q.push(f->left);

}

if(f->right)

{

q.push(f->right);

}

}

}

return -1;

}

};

1. <https://leetcode.com/problems/diameter-of-binary-tree/>

class Solution {

pair<int, int> diameter(TreeNode\* root)

{

if(root==NULL)

{

return {0, 0};

}

pair<int, int> leftD = diameter(root->left);

pair<int, int> rightD = diameter(root->right);

int lh = leftD.first;

int rh = rightD.first;

int ld = leftD.second;

int rd = rightD.second;

pair<int, int> p;

p.first = max(lh, rh) + 1;

p.second = max(lh+rh, max(ld, rd));

return p;

}

public:

int diameterOfBinaryTree(TreeNode\* root) {

pair<int, int> p = diameter(root);

return p.second;

}

};

1. <https://leetcode.com/problems/sum-root-to-leaf-numbers/>

class Solution {

int res = 0;

void sumRootLeaf(TreeNode\* root, int curr)

{

if(root==NULL)

{

return;

}

curr \*= 10;

curr += root->val;

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

{

res += curr;

return;

}

sumRootLeaf(root->left, curr);

sumRootLeaf(root->right, curr);

}

public:

int sumNumbers(TreeNode\* root) {

if(root==NULL)

{

return 0;

}

sumRootLeaf(root, 0);

return res;

}

};

1. <https://leetcode.com/problems/binary-tree-preorder-traversal/>

class Solution {

public:

vector<int> preorderTraversal(TreeNode\* root) {

if(root==NULL)

{

return {};

}

stack<TreeNode\*> s;

vector<int> res;

s.push(root);

while(!s.empty())

{

TreeNode \* temp = s.top();

s.pop();

res.push\_back(temp->val);

if(temp->right)

{

s.push(temp->right);

}

if(temp->left)

{

s.push(temp->left);

}

}

return res;

}

};