Lecture 35 Binary Trees

1. <https://leetcode.com/problems/lowest-common-ancestor-of-a-binary-tree/>

class Solution {

public:

TreeNode\* lowestCommonAncestor(TreeNode\* root, TreeNode\* p, TreeNode\* q) {

if(root==NULL)

{

return NULL;

}

if(root==p || root==q)

{

return root;

}

TreeNode\* leftLca = lowestCommonAncestor(root->left, p, q);

TreeNode\* rightLca = lowestCommonAncestor(root->right, p, q);

if(leftLca!=NULL && rightLca!=NULL)

{

return root;

}

return leftLca?leftLca:rightLca;

}

};

1. <https://leetcode.com/problems/merge-two-binary-trees/>

class Solution {

public:

TreeNode\* mergeTrees(TreeNode\* t1, TreeNode\* t2) {

if(t1==NULL)

{

return t2;

}

if(t2==NULL)

{

return t1;

}

t1->val += t2->val;

t1->left = mergeTrees(t1->left, t2->left);

t1->right = mergeTrees(t1->right, t2->right);

return t1;

}

};

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

class Solution {

class HBPair

{

public:

int height;

bool balance;

};

HBPair isHeightBalanced(TreeNode\* root)

{

HBPair p;

if(root==NULL)

{

p.height = 0;

p.balance = true;

return p;

}

HBPair left = isHeightBalanced(root->left);

HBPair right = isHeightBalanced(root->right);

p.height = max(left.height, right.height) + 1;

if(abs(left.height-right.height)<=1 && left.balance && right.balance)

{

p.balance = true;

}

else

{

p.balance = false;

}

return p;

}

public:

bool isBalanced(TreeNode\* root) {

if(isHeightBalanced(root).balance)

{

return true;

}

else

{

return false;

}

}

};

1. <https://leetcode.com/problems/construct-binary-tree-from-preorder-and-inorder-traversal/>

class Solution {

int i = 0;

TreeNode\* Traversal(vector<int>& preorder, vector<int>& inorder, int s, int e)

{

if(s>e)

{

return NULL;

}

TreeNode\* root = new TreeNode(preorder[i]);

int index = -1;

for(int j=s; j<=e; j++)

{

if(preorder[i]==inorder[j])

{

index = j;

break;

}

}

i++;

root->left = Traversal(preorder, inorder, s, index-1);

root->right = Traversal(preorder, inorder, index+1, e);

return root;

}

public:

TreeNode\* buildTree(vector<int>& preorder, vector<int>& inorder) {

TreeNode\* root = NULL;

int n = preorder.size();

root = Traversal(preorder, inorder, 0, n-1);

return root;

}

};