1. **Balanced tree maximum path sum**

#include <iostream>

#include <algorithm>

#include <climits>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

int maxPathSum(TreeNode\* root) {

max\_sum = INT\_MIN; // Initialize max\_sum to the smallest integer

maxPathFromNode(root);

return max\_sum;

}

private:

int max\_sum; // This will store the maximum path sum

int maxPathFromNode(TreeNode\* node) {

if (!node) {

return 0; // Base case: if the node is null, return 0

}

int left\_max = std::max(maxPathFromNode(node->left), 0); // Only consider positive sums

int right\_max = std::max(maxPathFromNode(node->right), 0); // Only consider positive sums

int current\_sum = node->val + left\_max + right\_max;

max\_sum = std::max(max\_sum, current\_sum);

return node->val + std::max(left\_max, right\_max);

}

1. **Binary tree pre order traversal**

#include <iostream>

#include <vector>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

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

std::vector<int> result;

traverse(root, result);

return result;

}

private:

void traverse(TreeNode\* node, std::vector<int>& result) {

if (node) {

result.push\_back(node->val); // Visit the root

traverse(node->left, result); // Traverse left subtree

traverse(node->right, result); // Traverse right subtree

}

}

};

Binary tree post order traversal

#include <iostream>

#include <vector>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

std::vector<int> postorderTraversal(TreeNode\* root) {

std::vector<int> result;

traverse(root, result);

return result;

}

private:

void traverse(TreeNode\* node, std::vector<int>& result) {

if (node) {

traverse(node->left, result); // Traverse left subtree

traverse(node->right, result); // Traverse right subtree

result.push\_back(node->val); // Visit the root

}

}

};

1. **Binary tree maximum path sum**

#include <iostream>

#include <algorithm>

#include <climits>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

int maxPathSum(TreeNode\* root) {

max\_sum = INT\_MIN; // Initialize max\_sum to the smallest integer

maxPathFromNode(root);

return max\_sum;

}

private:

int max\_sum; // This will store the maximum path sum

int maxPathFromNode(TreeNode\* node) {

if (!node) {

return 0; // Base case: if the node is null, return 0

}

// Recursively get the maximum path sum from left and right subtrees

int left\_max = std::max(maxPathFromNode(node->left), 0); // Only consider positive sums

int right\_max = std::max(maxPathFromNode(node->right), 0); // Only consider positive sums

// Calculate the maximum path sum passing through the current node

int current\_sum = node->val + left\_max + right\_max;

// Update the global maximum path sum

max\_sum = std::max(max\_sum, current\_sum);

// Return the maximum path sum extending from this node

return node->val + std::max(left\_max, right\_max);

}

};

1. **Binary tree in order traversal**

#include <iostream>

#include <vector>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

std::vector<int> inorderTraversal(TreeNode\* root) {

std::vector<int> result;

traverse(root, result);

return result;

}

private:

void traverse(TreeNode\* node, std::vector<int>& result) {

if (node) {

traverse(node->left, result); // Traverse left subtree

result.push\_back(node->val); // Visit the root

traverse(node->right, result); // Traverse right subtree

}

}

};

1. **Pre order**

#include <iostream>

#include <vector>

struct TreeNode {

int val;

TreeNode\* left;

TreeNode\* right;

TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}

};

class Solution {

public:

std::vector<int> postorderTraversal(TreeNode\* root) {

std::vector<int> result;

traverse(root, result);

return result;

}

private:

void traverse(TreeNode\* node, std::vector<int>& result) {

if (node) {

traverse(node->left, result); // Traverse left subtree

traverse(node->right, result); // Traverse right subtree

result.push\_back(node->val); // Visit the root

}

}

};