



資料結構

Data Structure

Lab 07

姓名： 曾致嘉

學號： 113AB0014

Lab07-Q1

Modify the inorder traversal function to a postorder traversal function in Part1.

Code

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

// 樹的節點
class TreeNode {
public:
    int value;           // 節點的值
    TreeNode* left;      // 左子節點
    TreeNode* right;     // 右子節點

    TreeNode(int val) : value(val), left(nullptr), right(nullptr) {} // 初始化節點
};

// 樹結構
class BinaryTree {
public:
    TreeNode* root;

    BinaryTree() : root(nullptr) {} // 初始化樹

    // 用陣列構建二元樹
    TreeNode* buildTree(vector<int>& arr) {
        if (arr.empty()) return nullptr;

        queue<TreeNode*> q; // 建立 queue 儲存待處理的節點
        root = new TreeNode(arr[0]); // 建立根節點 (陣列第一個元素)
        q.push(root); // 將根節點加入 queue

        size_t i = 1; // 陣列索引
        while (!q.empty()) { // i < arr.size()
            TreeNode* current = q.front(); // 取出 queue 中的節點
```

```

        q.pop();

        // 添加左子節點
        if (i < arr.size()) {
            current->left = new TreeNode(arr[i]);
            q.push(current->left); // 將左子節點加入 queue
            i++;
        }

        // 添加右子節點
        if (i < arr.size()) {
            current->right = new TreeNode(arr[i]);
            q.push(current->right); // 將右子節點加入 queue
            i++;
        }

    }

    return root;
}

// 中序遍歷
void inorderTraversal(TreeNode* node) {
    if (node == nullptr) return; // 如果節點為空，忽略

    inorderTraversal(node->left); // 遍歷左子樹
    cout << node->value << " "; // 訪問當前節點
    inorderTraversal(node->right); // 遍歷右子樹
}

void postorderTraversal(TreeNode* node){
    if (node == nullptr) return; // 已經到底拉~

    postorderTraversal(node->left); //先左邊
    postorderTraversal(node->right); //再右邊
    cout << node->value << " "; // 最後自己
}

```

```
};
```

```
int main() {
```

```
    BinaryTree tree; // 宣告二元樹
```

```
    // 輸入陣列用於構建樹，NULL 表示空子節點
```

```
    vector<int> arr = { 1, 2, 3, 4, 5, 6, 7 };
```

```
    tree.buildTree(arr); // 建立樹
```

```
    // 中序遍歷輸出
```

```
    cout << "Inorder Traversal: ";
```

```
    tree.inorderTraversal(tree.root);
```

```
    cout << endl;
```

```
    // 後序遍歷輸出
```

```
    cout<<"Postorder Traversal: ";
```

```
    tree.postorderTraversal(tree.root);
```

```
    cout<< endl;
```

```
    return 0;
```

```
}
```

Discussion Section

Picture 1:

The screenshot shows a C++ IDE with two windows. The left window displays the source code for a binary tree implementation. The right window shows the command prompt output.

```
LAB7_Q1.cpp U LAB7_Q2.cpp U
LAB7 > LAB7_Q1.cpp > main()
17 class BinaryTree {
64 void postorderTraversal(TreeNode* node){
65     if (node == nullptr) return; // 已經到底層
66
67     postorderTraversal(node->left); // 先左邊
68     postorderTraversal(node->right); // 再右邊
69     cout << node->value << " "; // 最後自己
70 }
71
72 };
73
74
75
76 int main() {
77     BinaryTree tree; // 宣告二元樹
78
79     // 輸入陣列用於構建樹，NULL 表示空子節點
80     vector<int> arr = { 1, 2, 3, 4, 5, 6, 7 };
81
82     tree.buildTree(arr); // 建立樹
83
84     // 中序遍歷輸出
85     cout << "Inorder Traversal: ";
86     tree.inorderTraversal(tree.root);
87     cout << endl;
88     // 後序遍歷輸出
89     cout << "Postorder Traversal: ";
90     tree.postorderTraversal(tree.root);
91     cout << endl;
92
93
94
95
96     return 0;
97 }
```

```
C:\WINDOWS\system32\cmd.
Microsoft Windows [Version 10.0.26100.3775]
(c) Microsoft Corporation. All rights reserved.

C:\Users\user>cd C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7

C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>ls
'ls' is not recognized as an internal or external command,
operable program or batch file.

C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>main
Inorder Traversal: 4 2 5 1 6 3 7
Postorder Traversal: 4 5 2 6 7 3 1

C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>
```

Picture 2:

```
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>main
Inorder Traversal: 4 2 5 1 6 3 7
Postorder Traversal: 4 5 2 6 7 3 1
```

Lab07-Q2

Implement a method to find the maximum value in the left and right subtrees.

Code

```
#include <iostream>
#include <queue>
#include <vector>
using namespace std;

// 樹的節點
class TreeNode {
public:
    int value;           // 節點的值
    TreeNode* left;      // 左子節點
    TreeNode* right;     // 右子節點

    TreeNode(int val) : value(val), left(nullptr), right(nullptr) {} // 初始化節點
};

// 樹結構
class BinaryTree {
public:
    TreeNode* root;

    BinaryTree() : root(nullptr) {} // 初始化樹

    // 用陣列構建二元樹
    TreeNode* buildTree(vector<int>& arr) {
        if (arr.empty()) return nullptr;

        queue<TreeNode*> q; // 建立 queue 儲存待處理的節點
        root = new TreeNode(arr[0]); // 建立根節點 (陣列第一個元素)
        q.push(root); // 將根節點加入 queue

        size_t i = 1; // 陣列索引
        while (!q.empty() ) { // i < arr.size()
            TreeNode* current = q.front(); // 取出 queue 中的節點
            q.pop();
```

```

        // 添加左子節點
        if (i < arr.size()) {
            current->left = new TreeNode(arr[i]);
            q.push(current->left); // 將左子節點加入 queue
            i++;
        }

        // 添加右子節點
        if (i < arr.size()) {
            current->right = new TreeNode(arr[i]);
            q.push(current->right); // 將右子節點加入 queue
            i++;
        }
    }

    return root;
}

// 中序遍歷
void inorderTraversal(TreeNode* node) {
    if (node == nullptr) return; // 如果節點為空，忽略

    inorderTraversal(node->left); // 遍歷左子樹
    cout << node->value << " "; // 訪問當前節點
    inorderTraversal(node->right); // 遍歷右子樹
}

int findTheMax(TreeNode* node){
    int left_data, right_data; //宣告左右資料
    if (node == nullptr){
        return -1; //到達最末端
    }
    left_data = findTheMax(node->left); //往左邊找
    right_data = findTheMax(node->right); //往右邊找
    return max(node->value, left_data, right_data);
}
// 比較自己、左子節點與右子節點的大小

```

```
int main() {
    BinaryTree tree; // 宣告二元樹

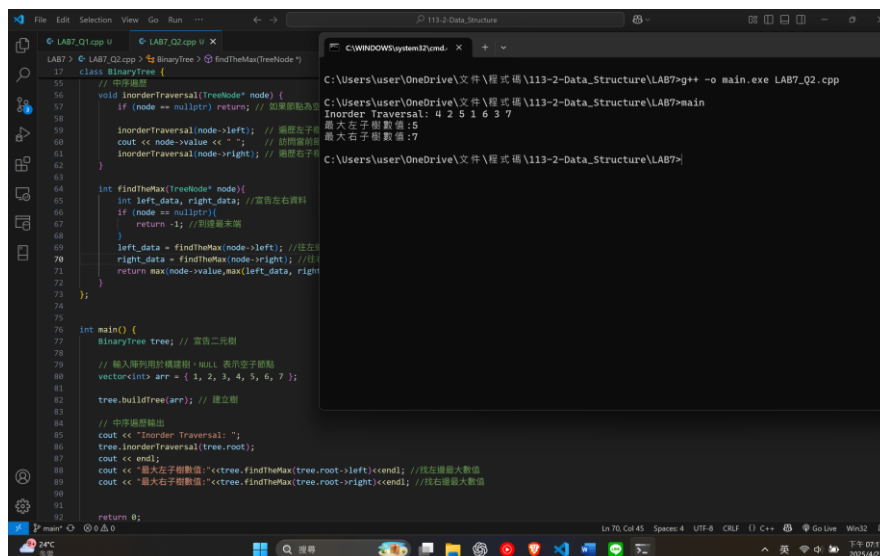
    // 輸入陣列用於構建樹，NULL 表示空子節點
    vector<int> arr = { 1, 2, 3, 4, 5, 6, 7 };

    tree.buildTree(arr); // 建立樹

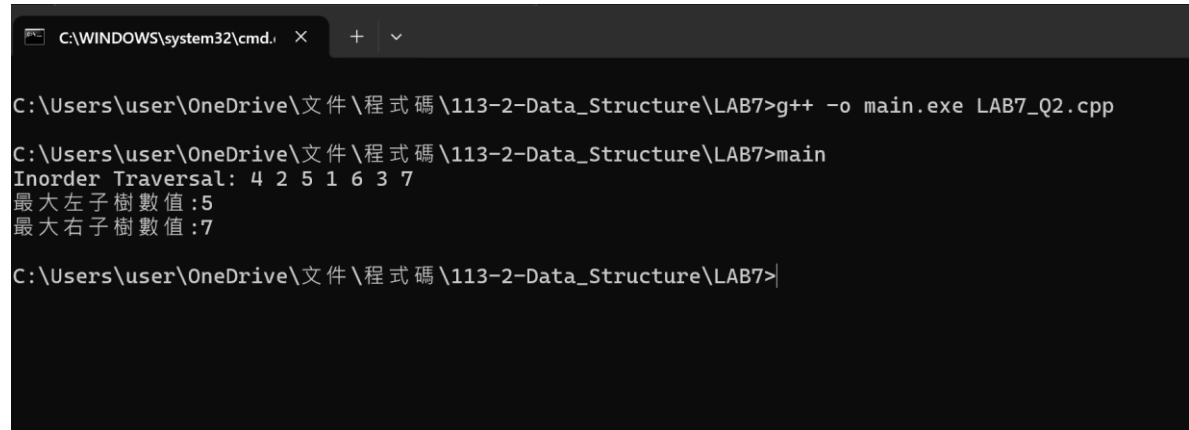
    // 中序遍歷輸出
    cout << "Inorder Traversal: ";
    tree.inorderTraversal(tree.root);
    cout << endl;
    cout << "最大左子樹數值:"<<tree.findTheMax(tree.root->left)<<endl;
    //找左邊最大數值
    cout << "最大右子樹數值:"<<tree.findTheMax(tree.root->right)<<endl;
    //找右邊最大數值

    return 0;
}
```

Picture 1:



Picture 2:



```
C:\WINDOWS\system32\cmd.exe X + v  
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>g++ -o main.exe LAB7_Q2.cpp  
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>main  
Inorder Traversal: 4 2 5 1 6 3 7  
最大左子樹數值:5  
最大右子樹數值:7  
C:\Users\user\OneDrive\文件\程式碼\113-2-Data_Structure\LAB7>|
```