数据结构作业
2025.10.29

1. 选择题

(3) 一颗完全二叉树上有1001个节点，其中叶子节点的个数是（ ）。

A. 250　　B. 254　　C. 500　　D. 501

(7) 对二叉树的节点从1开始进行连续编号，要求每个节点的编号大于其左、右孩子的编号，同一节点的左、右孩子中，其左孩子的编号小于其右孩子的编号，可采用（ ）遍历实现编号。

A. 先序　　B. 中序　　C. 后序　　D. 从根开始按层次

(8) 在一棵度为4的树T中，若有20个度为4的节点，10个度为3的节点，1个度为2的节点，10个度为1的节点，则树T的叶节点个数是（ ）。

A. 41　　B. 82　　C. 113　　D. 122

(14) 设F是森林，B是由F变换而得的二叉树。若F中有n个非终端节点，则B中右指针域为空的节点有（ ）个。

A. n-1　　B. n　　C. n+1　　D. n+2

2. 应用题

(2) 设一棵二叉树的先序序列为ABDFCEGH，中序序列为BFDAGEHC。

① 画出这棵二叉树。

② 画出这棵二叉树的后序线索树。

③ 将这棵二叉树转换成对应的树（或森林）。

(3) 假设用于通信的电文仅由8个字母组成，字母在电文中出现的频率分别为0.07、0.19、0.02、0.06、0.32、0.03、0.21、0.10。

① 试为这8个字母设计哈夫曼编码。

② 试设计另一种由二进制表示的等长编码方案。

③ 对于上述实例，比较两种方案的优缺点。

3. 算法设计题

以二叉链表作为二叉树的存储结构，设计以下算法

(2) 判别两棵树是否相等。

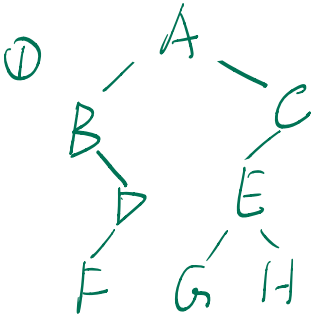
(5) 计算二叉树最大的宽度（二叉树的最大宽度是指二叉树所有层中节点个数的最大值）。

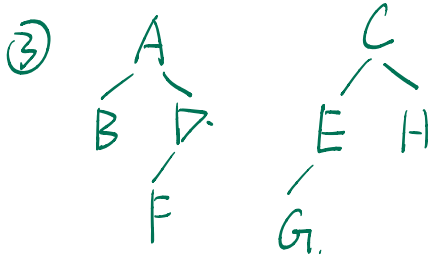
(2) 设一棵二叉树的先序序列为ABDFCEGH，中序序列为BFDAGEHC。

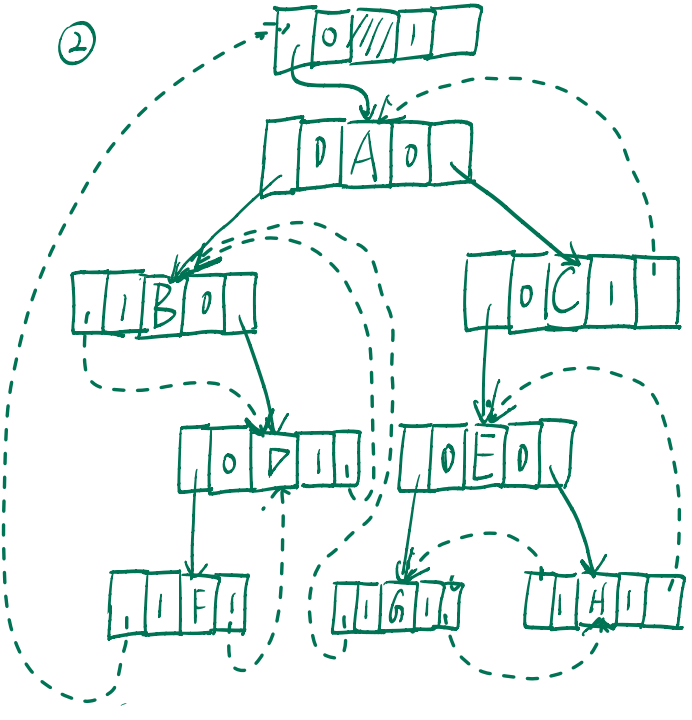
① 画出这棵二叉树。

② 画出这棵二叉树的后序线索树。

③ 将这棵二叉树转换成对应的树（或森林）。







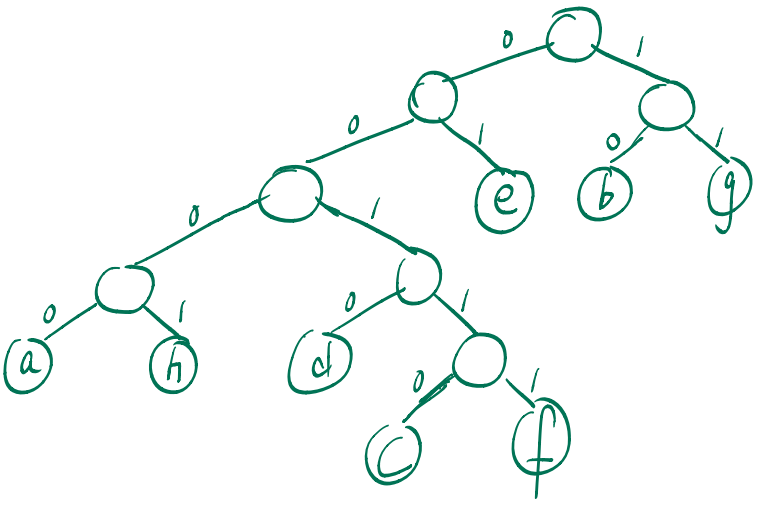
(3) 假设用于通信的电文仅由8个字母组成，字母在电文中出现的频率分别为0.07、0.19、0.02、0.06、0.32、0.03、0.21、0.10。

① 试为这8个字母设计哈夫曼编码。

② 试设计另一种由二进制表示的等长编码方案。

③ 对于上述实例，比较两种方案的优缺点。

① 不妨设这8个字母分别为 a, b, c, d, e, f, g, h



则 a 0000　b 010　c 00110　d 0010

e 01　f 00111　g 11　h 0001

② a 000　b 001　c 010　d 011

e 100　f 101　g 110　h 111

③ 对于哈夫曼编码，查找速度变快，但数据消耗空间变大

平均查找速度为 $4\times0.07+3\times0.19+5\times0.02+4\times0.06+2\times0.32+5\times0.03+2\times0.21+4\times0.1=2.8$

对于等长编码，平均查找次数为8次，但数据占空间小

100. 相同的树

已解答

C++ 智能模式

```cpp
/**
 * 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:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        /*实现思路：先序遍历 + 中序遍历 序列都相同即相同*/
        return DLR(p) == DLR(q) && LDR(p) == LDR(q);
    }
    bool isLeaf(TreeNode* t){
        return (t->left == nullptr  && t->right == nullptr);
    }
    string DLR(TreeNode* t){
        /*先序遍历算法*/
        if(!t) return "";
        string res = std::to_string(t -> val);
        if(t->left){
            res += DLR(t->left);
        }else{
            res += "*";
        }
        if(t->right){
            res += DLR(t->right);
        }else{
            res += "*";
        }
        return res;
    }

    string LDR(TreeNode* t){
        /*中序遍历算法*/
        if(!t) return "";
        string res;
        if(t->left){
            res += DLR(t->left);
        }else{
            res += "*";
        }
        res += std::to_string(t -> val);
        if(t->right){
            res += DLR(t->right);
        }else{
            res += "*";
        }
        return res;
    }
```

测试用例 | >_ 测试结果

通过 执行用时：0 ms

Case 1 Case 2 Case 3 Case 4 Case 5

输入

p =
[2,2,2,null,2,null,null,2]

q =
[2,2,2,2,null,2,null]

输出

false

预期结果

false

测试用例 | >_ 测试结果

通过 执行用时：0 ms

Case 1 Case 2 Case 3 Case 4 Case 5

输入

p =
[]

q =
[]

输出

true

预期结果

true

测试用例 | >_ 测试结果

通过 执行用时：0 ms

Case 1 Case 2 Case 3 Case 4 Case 5

输入

p =
[2,2,2,null,2,null,null,2]

q =
[2,2,2,2,null,2,null]

输出

false

预期结果

false

贡献测试用例

```cpp
#include <iostream>
#include <queue>
#include <vector>
#include <utility>

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 widthOfBinaryTree(TreeNode* root) {
        /*
            1.实现思路：队列+广度优先搜索 基于地址的算法
                地址计算：left -> idx*2, right -> idx*2+1.
                每层宽度计算：width = last_index - first_index + 1.
            2.难点：层数的确定
        */
        if (root == nullptr) return 0;
        unsigned long long maxWidth = 0;
        std::queue<std::pair<TreeNode*, unsigned long long>> q; //采用ull存储数字 防止地址过大溢出
        q.push({root, 1ULL});
        while (!q.empty()) {
            size_t levelSize = q.size(); // 当前层的节点数
            unsigned long long firstIndex = q.front().second;
            unsigned long long lastIndex = firstIndex;
            for (size_t i = 0; i < levelSize; ++i) {
                auto p = q.front(); q.pop();
                TreeNode* node = p.first;
                unsigned long long idx = p.second;
                lastIndex = idx;
                if (node->left)  q.push({node->left,  idx * 2ULL});
                if (node->right) q.push({node->right, idx * 2ULL + 1ULL});
            }
            unsigned long long width = lastIndex - firstIndex + 1ULL;
            if (width > maxWidth) maxWidth = width;
        }
        return static_cast<int>(maxWidth);
    }
};
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS D:\05_fourteen\ustc-life\2025fall\data-structure\hw5> g++ -std=c++17 -O2 "d:\05_fourteen\ustc-life
\2025fall\data-structure\hw5\tree.cpp" -o "d:\05_fourteen\ustc-life\2025fall\data-structure\hw5\tree.
exe"; if ($LASTEXITCODE -eq 0) { & "d:\05_fourteen\ustc-life\2025fall\data-structure\hw5\tree.exe" }
else { Write-Error "Compilation failed" }
d:\05_fourteen\ustc-life\2025fall\data-structure\hw5\tree.cpp:1:9: warning: #pragma once in main file
 #pragma once
         ^~~~
empty tree: expected=0, got=0 [PASS]
single node: expected=1, got=1 [PASS]
full tree height 3: expected=4, got=4 [PASS]
sparse example [1,3,2,5,3,null,9]: expected=4, got=4 [PASS]
skewed-left tree: expected=1, got=1 [PASS]

Tests passed: 5 / 5
```