中山大学计算机学院本科生实验报告

(2025 学年第 1 学期)

课程名称：数据结构与算法实验　　任课老师：张子臻

年级	2024 级	专业（方向）	计算机科学与技术 (人工智能与大数据)
学号	24325155	姓名	梁桂铭
电话	15817681625	Email	lianggm8@mail2.sysu.edu.cn
开始日期	2025 年 11 月 12 日	完成日期	2025 年 11 月 12 日

第一题：实验题目

给你一棵以 root 为根的二叉树，二叉树中的交错路径定义如下：

- 选择二叉树中任意节点和一个方向（左或者右）。
- 如果前进方向为右，那么移动到当前节点的右子节点，否则移动到它的左子节点。
- 改变前进方向：左变右或者右变左。
- 重复第二步和第三步，直到你在树中无法继续移动。

交错路径的长度定义为：访问过的节点数目 - 1（单个节点的路径长度为 0）。
请你返回给定树中最长交错路径的长度。

实验目的

1. 理解二叉树的结构特性和遍历方法。
2. 掌握递归算法在二叉树问题中的应用。
3. 实现计算二叉树中最长交错路径长度的算法。
4. 分析算法的时间复杂度和空间复杂度。

算法设计

本算法采用递归方法计算二叉树中最长交错路径的长度。主要思路如下：

- 定义辅助函数 `getlength(TreeNode* T, int n)`，其中参数 `n` 表示当前方向（0 表示左，1 表示右）。
- 在 `getlength` 函数中，如果当前节点为空，返回 0；否则根据方向参数判断下一步移动方向，并递归计算路径长度。
- 在 `longestZigZag` 函数中，分别计算从根节点出发向左和向右的路径长度，然后递归计算左右子树的最长交错路径长度。
- 最终返回三者中的最大值作为整棵树的最长交错路径长度。

复杂度分析：

- **时间复杂度**：$O(n)$，其中 n 为二叉树节点数，每个节点被访问一次。
- **空间复杂度**：$O(h)$，其中 h 为二叉树的高度，主要是递归调用栈的空间。

程序运行与测试

```cpp
#include "TreeNode.h"
#include <iostream>
using namespace std;

int getlength(TreeNode* T, int n) {
    if (T == nullptr) return 0;
    else if (n == 0) {
        if (T -> left == nullptr) return 0;
        else return getlength(T -> left, 1) + 1;
    }
    else {
        if (T -> right == nullptr) return 0;
        else return getlength(T -> right, 0) + 1;
    }
}

int longestZigZag(TreeNode* root) {
    if (root == nullptr) return 0;
    else {
        int l1 = 0, l2 = 0, m = 0;
        l1 = getlength(root, 0);
        l2 = getlength(root, 1);
        if (m < max(l1, l2)) {
            m = max(l1, l2);
        }
        m = max(m, max(longestZigZag(root -> right), longestZigZag(root -> left)));
        return m;
```

```
    }
}
```

实验总结

通过本次实验，我深入理解了二叉树的结构特性和递归算法的应用。在实现最长交错路径长度计算的过程中，我掌握了如何通过递归遍历二叉树并维护路径状态。算法通过方向参数的巧妙设计，实现了交错路径的模拟计算。递归方法虽然简洁高效，但需要注意栈空间的使用。本次实验让我对树形结构的递归处理有了更深刻的认识，为后续学习更复杂的树算法打下了坚实基础。我在实验中也遇到一些问题，一开始我设置了全局变量 m 用来存储最大值，但这样会导致在主函数当中每次调用函数后 m 无法更新而导致出错，这件事告诉我在设置变量时需要考虑周全。

第二题：实验题目

给定一颗二叉查找树，其中结点上存储整数关键字，请你判断它是否一棵平衡的二叉查找树，即每个内部结点的两颗子树高度差小于等于 1。

实验目的

1. 掌握二叉查找树的基本性质和先序遍历序列的特点
2. 理解平衡二叉树（AVL 树）的定义和判断条件
3. 学习通过递归方法计算二叉树的高度并判断平衡性

算法设计

核心思路

根据二叉查找树的性质，先序遍历序列的第一个元素为根节点，后续序列中所有小于根节点的元素构成左子树，大于根节点的元素构成右子树。通过递归计算左右子树的高度，并判断高度差是否满足平衡条件。

算法步骤

1. **输入处理**：读取测试样例数，对每个样例读取节点数和先序遍历序列
2. **递归计算高度**：函数 h 接收数组、起始索引、结束索引和平衡标志 flag
   - 基准情况：当起始索引等于结束索引时，返回高度 1
   - 递归情况：根据根节点值划分左右子树，分别递归计算高度
   - 平衡判断：比较左右子树高度差，若大于 1 则设置平衡标志 flag = 1
3. **结果输出**：根据平衡标志输出”YES” 或”NO”

复杂度分析

时间复杂度

- **最坏情况**：当二叉查找树退化为链表时，每次递归都需要遍历整个子树来划分左右子树，时间复杂度为 $O(n^2)$
- **最好情况**：当二叉查找树为完全平衡树时，时间复杂度为 $O(n \log n)$
- **平均情况**：时间复杂度为 $O(n \log n)$

空间复杂度

- **递归栈空间**：递归深度为树的高度，最坏情况下（退化为链表）为 $O(n)$，平衡情况下为 $O(\log n)$
- **辅助空间**：除了输入数组外，算法只使用了常数级别的额外空间
- **总空间复杂度**：$O(n)$（主要取决于递归栈深度）

程序运行与测试

程序代码

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int h(int* arr, int s, int e, int& flag) {
    if (s == e) return 1;
    if (s < e) {
        int root = arr[s];
        int l1 = 0, l2 = 0;
        for (int i = s + 1; i < e; i++) {
            if (arr[i] > root) break;
            l1++;
        }
        l2 = e - s - l1 - 1;
        int h1 = h(arr, s + 1, s + 1 + l1, flag) + 1;
        int h2 = h(arr, s + 1 + l1, e, flag) + 1;
        if (abs(h1 - h2) > 1) flag = 1;
        return max(h1, h2);
    }
}

int main() {
    int N, n;
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> n;
        int* arr = new int[n];
        for (int j = 0; j < n; j++) cin >> arr[j];
```

```
        int f = 0;
        int& flag = f;
        h(arr, 0, n, flag);
        if (flag == 0) cout << "YES" << endl;
        else cout << "NO" << endl;
    }
    return 0;
}
```

实验总结

- 成功实现了通过先序遍历序列判断二叉查找树平衡性的算法
- 掌握了利用递归同时计算高度和判断平衡性的技巧
- 加深了对二叉查找树性质和平衡二叉树定义的理解
- 通过引用传递平衡标志的方法实现了在递归过程中记录状态信息

第三题：实验题目

给定一棵由正整数 1,2,3,... 组成的无限大满二叉树，求两个结点 X 和 Y 的最近公共祖先。从某个结点到根结点的路径是唯一的，要求找到两个路径从根节点开始最后一个相同的结点。

实验目的

1. 理解完全二叉树的结构性质和编号规律
2. 掌握二叉树中最近公共祖先 (LCA) 问题的求解方法
3. 学习利用数学方法计算二叉树结点的深度
4. 提高对二叉树路径操作的理解和应用能力

算法设计

核心思路

利用完全二叉树的性质：对于任意结点 i，其父结点为 i/2。通过计算两个结点到根结点的深度，先将较深的结点向上移动至同一深度，然后同时向上移动两个结点直到相遇，相遇点即为最近公共祖先。

算法步骤

1. **输入处理**：读取测试用例个数，对每个测试用例读取两个整数 X 和 Y
2. **计算深度**：利用对数公式计算两个结点在二叉树中的深度
3. **深度对齐**：将较深的结点向上移动，直到两个结点处于同一深度
4. **同时上移**：同时向上移动两个结点，直到它们相遇
5. **输出结果**：输出相遇的结点编号，即为最近公共祖先

复杂度分析

时间复杂度

- 每个测试用例的时间复杂度为 $O(\log n)$，其中 n 为结点编号的最大值
- 总时间复杂度为 $O(T \log n)$，T 为测试用例个数

空间复杂度

- 算法只使用了常数级别的额外空间，空间复杂度为 $O(1)$

程序运行与测试

程序代码

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    int N, x, y, xl, yl;
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> x >> y;
        xl = log(x) / log(2) + 1;
        yl = log(y) / log(2) + 1;
        while (xl < yl) {
            y = y / 2;
            yl--;
        }
        while (yl < xl) {
            x = x / 2;
            xl--;
        }
        while (x != y) {
            x /= 2;
            y /= 2;
        }
```

```
        cout << x << endl;
    }
    return 0;
}
```

实验总结

- 成功实现了在完全二叉树中求解最近公共祖先的高效算法
- 掌握了利用完全二叉树编号规律和数学计算的方法
- 该算法利用了完全二叉树的特殊性质，不适用于一般的二叉树结构

附录、提交文件清单

第一题

```
#include "TreeNode.h"
#include <iostream>
using namespace std;

int getlength(TreeNode* T, int n) {
    if (T == nullptr) return 0;
    else if (n == 0) {
        if (T -> left == nullptr) return 0;
        else return getlength(T -> left, 1) + 1;
    }
    else {
        if (T -> right == nullptr) return 0;
        else return getlength(T -> right, 0) + 1;
    }
}

int longestZigZag(TreeNode* root) {
    if (root == nullptr) return 0;
    else {
        int l1 = 0, l2 = 0, m = 0;
        l1 = getlength(root, 0);
        l2 = getlength(root, 1);
        if (m < max(l1, l2)) {
            m = max(l1, l2);
        }
        m = max(m, max(longestZigZag(root -> right), longestZigZag(root -> left)));
        return m;
    }
}
```

第二题

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int h(int* arr, int s, int e, int& flag) {
    if (s == e) return 1;
    if (s < e) {
        int root = arr[s];
        int l1 = 0, l2 = 0;
        for (int i = s + 1; i < e; i++) {
            if (arr[i] > root) break;
            l1++;
        }
        l2 = e - s - l1 - 1;
        int h1 = h(arr, s + 1, s + 1 + l1, flag) + 1;
        int h2 = h(arr, s + 1 + l1, e, flag) + 1;
        if (abs(h1 - h2) > 1) flag = 1;
        return max(h1, h2);
    }
}

int main() {
    int N, n;
    cin >> N;
    for (int i = 0; i < N; i++) {
        cin >> n;
        int* arr = new int[n];
        for (int j = 0; j < n; j++) cin >> arr[j];
        int f = 0;
        int& flag = f;
        h(arr, 0, n, flag);
        if (flag == 0) cout << "YES" << endl;
        else cout << "NO" << endl;
    }
    return 0;
}
```

第三题

```cpp
#include <iostream>
#include <cmath>
using namespace std;

int main() {
    int N, x, y, xl, yl;
    cin >> N;
    for (int i = 0; i < N; i++) {
```

```
        cin >> x >> y;
        xl = log(x) / log(2) + 1;
        yl = log(y) / log(2) + 1;
        while (xl < yl) {
            y = y / 2;
            yl--;
        }
        while (yl < xl) {
            x = x / 2;
            xl--;
        }
        while (x != y) {
            x /= 2;
            y /= 2;
        }
        cout << x << endl;
    }
    return 0;
}
```