《离散数学》课程实验报告 5 最优 2 元树在通信编码中的应用

2254321 陆诚彬

1. 实验内容

本实验的目标是实现最优 2 元树（二叉树）在通信编码中的应用。具体任务是根据输入的一组通信符号使用频率，为每个通信符号计算出对应的前缀码。前缀码是一种特殊类型的编码，它保证任何一个编码不是其他编码的前缀，这在数据传输和压缩中非常重要，因为它消除了歧义，使得每个编码都是唯一可识别的。

2. 解题思路

2.1. 实验原理和方法

存储通信符号频率：实验首先利用一维数组 f[N] 存储通信符号的使用频率。这一步骤是为了后续的树构建提供基础数据。

构建最优 2 元树：使用给定的频率数据构建最优 2 元树。在这个过程中，根据频率合并节点，确保频率低的符号在树的更深层，频率高的符号在树的较浅层。这样做的目的是为了最小化整体编码长度。

使用链表保存树结构：树的结构通过链表形式保存，每个节点包含其数值和指向左右子节点的指针。

遍历树以输出前缀码：通过树的前序遍历，可以得到每个通信符号的前缀码。在遍历过程中，向左子节点遍历时添加 '0'，向右子节点遍历时添加 '1'，直到到达叶节点。到达叶节点时，遍历路径上的字符序列即构成了该节点的前缀码。

2.2. 实验代码分析

排序函数 sort：对构建过程中的节点数组进行排序，确保节点合并时总是选择频率最低的两个节点。

树构建函数 constructTree：利用频率数组 f 构建最优 2 元树。每次迭代合并两个频率最低的节点，直到构建出完整的树。

前序遍历函数 preorder：实现树的前序遍历，过程中构建前缀码，并在遇到叶节点时输出对应的通信符号和其前缀码。

主函数 main：实现用户输入的接口，接收通信符号的频率数据，构建树，并调用前序遍历函数输出所有通信符号的前缀码。

3. 数据结构设计

在实验代码中，为了实现最优 2 元树在通信编码的应用，采用了以下关键的数据结构设计：

1. TreeNode 结构

TreeNode 是构建最优 2 元树的基础结构单元。它具有以下特点：

属性：

- num：整型变量，用于存储节点的数值，这里代表通信符号的使用频率或其合并节点的频率总和。
- left 和 right：std::shared_ptr<TreeNode> 类型，指向左右子节点。这种智能指针的使用是为了自动管理内存，防止内存泄漏。

构造函数： TreeNode(int n) 用于创建一个新的树节点，初始化 num 为给定的频率，左右子节点为 nullptr。

2. 频率数组

类型： std::vector<int> frequencies(N)

用途： 存储输入的通信符号使用频率。这是构建最优 2 元树的起点，每个元素代表一个通信符号的使用频率。

3. 节点数组

类型： std::vector<std::shared_ptr<TreeNode>> nodes(N)

用途： 存储 TreeNode 类型的智能指针，用于构建最优 2 元树。每个 TreeNode 元素初开始代表一个单独的通信符号节点，随着树的构建过程，这些节点会根据频率被合并。

4. 字符串 s

类型： std::string s(2 * N, ' ')

用途： 在前序遍历过程中，用于构建并存储前缀码。字符串的长度设置为 2 * N 是为了确保有足够的空间存储可能的最长前缀码。

4. 项目实现

在本项目中，关键的函数包括树的构建、排序、前序遍历等。以下是这些重要函数的实现细节：

4.1. 树的构建：constructTree

功能： 构建最优 2 元树，用于生成通信符号的前缀码。

输入： 频率数组 f。

过程：

1) 初始化节点数组 nodes，每个节点初开始代表一个通信符号。
2) 通过循环，每次迭代合并频率最低的两个节点（即数组的前两个节点）。
3) 创建新节点作为这两个节点的父节点，新节点的频率是这两个子节点频率之和。
4) 更新节点数组，并调用 sort 函数以保持频率的递增顺序。
5) 重复以上步骤，直至只剩下一个节点，即构建出了完整的最优 2 元树。

输出： 返回构建完成的最优 2 元树的根节点。

```
1.   std::shared_ptr<TreeNode> constructTree(const std::vector<int>& f)
     {
2.       std::vector<std::shared_ptr<TreeNode>> nodes(N);
3.
4.       for (int i = 0; i < N; ++i) {
5.           nodes[i] = std::make_shared<TreeNode>(f[i]);
6.       }
7.
8.       for (int i = 1; i < N; ++i) {
9.           auto combined = std::make_shared<TreeNode>(nodes[i - 1]->
    num + nodes[i]->num);
10.          combined->left = nodes[i - 1];
11.          combined->right = nodes[i];
12.          nodes[i] = combined;
13.          sort(nodes, N - i);
14.      }
15.
16.      return nodes[N - 1];
17.  }
```

4.2. 节点排序：sort

功能： 对节点数组按照节点的 num（频率）进行排序，确保频率最小的节点在数组前面。

输入： 节点数组 array 和需要排序的元素个数 n。

过程：

1) 使用简单的冒泡排序法对数组进行排序。
2) 对于 array[i] 和 array[i+1]，如果 array[i] 的频率大于 array[i+1]，则交换它们的位置。
3) 重复此过程，直到数组中的元素按频率从小到大排列。

```
1.  void sort(std::vector<std::shared_ptr<TreeNode>>& array, int n) {
2.      for (int i = N - n; i < N - 1; i++) {
3.          if (array[i]->num > array[i + 1]->num) {
4.              std::swap(array[i], array[i + 1]);
5.          }
6.      }
7.  }
```

4.3. 前序遍历：preorder

功能： 前序遍历最优 2 元树，构建并输出前缀码。

输入： 树的根节点 node，当前深度 depth，字符 c，用于构建前缀码的字符串 s。

过程：

1) 检查当前节点是否为空。
2) 根据前序遍历的规则，首先处理当前节点，然后递归遍历左子节点和右子节点。
3) 在遍历过程中，通过添加字符 '0' 或 '1' 来构建前缀码。
4) 当到达叶节点时，输出该节点（通信符号）的前缀码。
5) 递归地调用 preorder 函数，首先对左子节点（添加 '0'），然后对右子节点（添加 '1'）。

```
1.  void preorder(const std::shared_ptr<TreeNode>& node, int depth, c
    har c, std::string& s) {
2.      if (node) {
3.          s[depth] = (c == '1') ? '0' : '1';
4.          if (!node->left && !node->right) {
5.              std::cout << node->num << ": ";
6.              for (int j = 0; j <= depth; ++j) {
7.                  std::cout << s[j];
8.              }
9.              std::cout << '\n';
```

```
            }
            preorder(node->left, depth + 1, '1', s);
            preorder(node->right, depth + 1, '0', s);
        }
    }
```

5. 设计小结

本项目成功实现了最优 2 元树在通信编码中的应用，主要通过构建一个特殊类型的二叉树来生成前缀码。这些前缀码在数据传输和压缩中极为重要，因为它们确保了编码的唯一性和非歧义性。整个实验的核心在于理解和应用二叉树的特性，以及如何利用这些特性在通信编码中找到最优解。

关键点回顾：

数据结构的有效应用：使用了 TreeNode 结构、频率数组、节点数组和字符串，这些数据结构紧密协作，完成了最优 2 元树的构建和前缀码的生成。

算法实现的高效性：通过排序函数和前序遍历算法，我们能够有效地构建树并提取所需的编码，这显示了算法在解决实际问题中的重要性。

代码设计的简洁性：整个项目的代码清晰易懂，功能分离明确，易于维护和理解。

通过本项目的设计和实现，我们不仅加深了对离散数学和数据结构的理解，还获得了实际应用理论知识解决实际问题的经验。

6. 实验心得

在完成《离散数学》课程的这次实验后，我获得了宝贵的学习经验和深刻的理解。

理论与实践的结合：这次实验让我明白了理论知识与实际编程之间的桥梁。通过将离散数学的理论应用于实际的编码问题，我能够更好地理解和记忆这些概念。

问题解决能力的提升：在实验过程中，我遇到了多个挑战，比如如何有效地构建二叉树、如何生成有效的前缀码等。通过研究和解决这些问题，我的问题解决能力和编程技能都得到了提升。

团队协作的重要性：虽然这是一个个人项目，但在实验过程中，我与同学们进行了讨论和交流，这让我认识到团队合作在解决复杂问题时的重要性。

终身学习的态度：这次实验也让我意识到，技术不断进步，作为计算机科学

的学生和未来的从业者，我们需要不断学习和适应新的技术和理论。

总之，这次实验不仅加深了我对离散数学和数据结构的理解，还提高了我的编程能力，增强了我解决实际问题的信心和能力。