《离散数学》课程实验报告文档

题目：最优 2 元树的应用

姓名：　赵卓冰

学号：　2252750

专业：　软件工程

年级：　2023 级

指导教师：唐剑锋

2024 年 12 月 1 日

1. 实验内容

输入一组通信符号的使用频率，构造霍夫曼树，计算每个通信符号的前缀码，最终输出所有符号的前缀码以及霍夫曼树的结构。

2. 实验原理和方法

霍夫曼编码是一种基于二叉树的无损压缩算法，广泛用于文件压缩和通信领域。通过构造霍夫曼树，可以为符号分配长度可变的前缀码，权重高的符号分配较短的编码，权重低的符号分配较长的编码，从而实现数据压缩。

2.1. 原理

霍夫曼树：

- 是一棵带权路径长度最小的二叉树。
- 路径长度是从根节点到叶子节点的路径上所有边的权值之和。
- 权重较大的符号离根较近，权重较小的符号离根较远。

前缀码：

- 霍夫曼编码是一种前缀编码，没有任何一个码是另一个码的前缀，保证了解码的唯一性。

2.2. 方法

输入通信符号和频率：

- 用一维数组存储通信符号和对应频率。

构造霍夫曼树：

- 使用优先队列（最小堆）存储树节点。
- 每次取出频率最小的两个节点合并，生成新节点并加入优先队列，直到队列中只剩下一个节点，作为霍夫曼树的根。

生成前缀码：

- 通过递归遍历霍夫曼树，从根节点到每个叶子节点的路径即为对应符号的前缀码。

输出前缀码和树结构：

- 按符号输出前缀码。
- 使用递归前序遍历输出霍夫曼树结构。

3. 实验步骤

3.1. 输入通信符号及频率

用户输入通信符号及其使用频率，程序以键值对的形式存储这些信息。

3.2. 构造霍夫曼树

1. 将所有符号作为叶子节点，存入最小堆。
2. 每次从堆中取出频率最小的两个节点，合并成一个新节点，并将其频率设为两子节点频率之和。
3. 将新节点插入堆中，重复操作，直到堆中只剩下一个节点，即霍夫曼树的根节点。

3.3. 生成前缀码

从根节点开始递归遍历：

1. 向左分支递归时在路径上加“0”。
2. 向右分支递归时在路径上加“1”。
3. 到达叶子节点时，将路径保存为该符号的前缀码。

3.4. 输出结果

1. **输出前缀码**：
    - 格式为“符号(频率)：前缀码”。
2. **输出霍夫曼树结构**：
    - 使用缩进展示树的层级结构，非叶子节点用“*”表示。

4. 数据结构设计

4.1. 霍夫曼树节点

```
struct HuffmanNode {
    char symbol;              // 符号
    int frequency;            // 频率
    HuffmanNode* left, * right; // 左右子树指针
};
```

4.2. 优先队列

使用 C++ STL 的 `priority_queue` 构造最小堆：

```
priority_queue<HuffmanNode*, vector<HuffmanNode*>, Compare>
```

4.3. 存储结果

- **符号及其前缀码**：
  使用 `unordered_map<char, string>` 保存每个符号的前缀码。
- **符号及其频率**：
  使用 `unordered_map<char, int>` 保存每个符号的频率。

5. 核心代码实现

5.1. 构造霍夫曼树

```
HuffmanNode* BuildTree() {
    priority_queue<HuffmanNode*, vector<HuffmanNode*>, Compare>
min_heap;

    for (const auto& entry : frequencies) {
        min_heap.push(new HuffmanNode(entry.first, entry.second));
    }

    while (min_heap.size() > 1) {
        HuffmanNode* left = min_heap.top();
        min_heap.pop();
        HuffmanNode* right = min_heap.top();
        min_heap.pop();

        HuffmanNode* merged = new HuffmanNode('*', left->frequency +
right->frequency);
        merged->left = left;
        merged->right = right;

        min_heap.push(merged);
```

```
    }

    return min_heap.top();
}
```

5.2. 生成前缀码

```
void GenerateCodesRecursive(HuffmanNode* node, string code) {
    if (!node)
        return;

    if (!node->left && !node->right) {
        symbol_codes_map[node->symbol] = code;
    }

    GenerateCodesRecursive(node->left, code + "0");
    GenerateCodesRecursive(node->right, code + "1");
}
```

5.3. 打印霍夫曼树

```
void PrintTreeRecursive(HuffmanNode* node, int depth) {
    if (!node)
        return;

    for (int i = 0; i < depth; ++i) {
        cout << "    ";
    }

    if (node->symbol == '*') {
        cout << "[*, " << node->frequency << "]" << endl;
    } else {
        cout << "[" << node->symbol << ", " << node->frequency << "]"
<< endl;
    }

    PrintTreeRecursive(node->left, depth + 1);
    PrintTreeRecursive(node->right, depth + 1);
}
```

6. 实验结果

```
输入节点个数:
13
请输入字符:
a b c d e f g h i j k l m
请输入字符对应的频率
2 3 5 7 11 13 17 19 23 29 31 37 41
------------------------------------------------
              霍夫曼树的目录结构:
'*'代表非叶子结点，括号里面是频率，冒号后是前缀码
------------------------------------------------
|*(238)
        |*(95)
                |*(42)
                        |h(19): 000
                        |i(23): 001
                |*(53)
                        |*(24)
                                |e(11): 0100
                                |f(13): 0101
                        |j(29): 011
        |*(143)
                |*(65)
                        |k(31): 100
                        |*(34)
                                |g(17): 1010
                                |*(17)
                                        |d(7): 10110
                                        |*(10)
                                                |c(5): 101110
                                                |*(5)
                                                        |a(2): 1011110
                                                        |b(3): 1011111
                |*(78)
                        |l(37): 110
                        |m(41): 111
------------------------
字符(频率): 前缀码
------------------------
h(19):    000
i(23):    001
e(11):    0100
f(13):    0101
j(29):    011
k(31):    100
g(17):    1010
d(7):     10110
c(5):     101110
a(2):     1011110
b(3):     1011111
l(37):    110
m(41):    111
```

7. 心得体会

通过本实验，我深入学习了霍夫曼编码的原理与实现，掌握了优先队列在构造最优二叉树中的应用，并通过递归生成前缀码与打印霍夫曼树结构。以下是我的收获：

1. **数据结构的选择**：
    - 使用优先队列高效地选择最小频率节点。
    - 通过链表形式保存树结构，便于递归操作。
2. **递归与贪心思想**：
    - 霍夫曼编码基于贪心策略，构造了最优二叉树。
    - 利用递归思想简化了树的遍历与操作。
3. **实践能力**：
    - 将理论知识应用到实际代码实现中，解决了符号编码问题。

本实验让我对霍夫曼编码的理论与实现有了更深刻的理解，同时也提升了我的的编程能力。