同济大学

TONGJI UNIVERSITY

离散数学课程设计

项目名称	最优二元树的应用
学　　院	计算机科学与技术学院
专　　业	软件工程
学生姓名	杨瑞晨
学　　号	2351050
指导教师	唐剑锋
日　　期	2024 年 12 月 1 日

目 录

1 项目分析

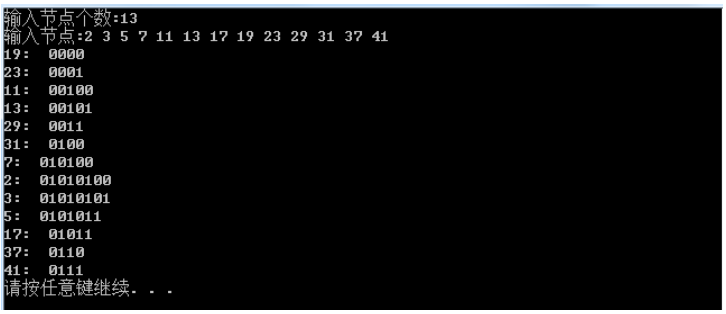
1.1 项目背景

在信息论和计算机科学中，编码选择至关重要。发送方必须思考如何按照特定规则有效转换所需传递的信息。霍夫曼编码是一种使用变长编码表对数据进行编码的方法，它是一种前缀编码方法，意味着没有任何一个编码是另一个编码的前缀。霍夫曼编码广泛应用于数据压缩领域，特别是在无损数据压缩中。这种编码方法通过为更频繁出现的字符分配较短的编码，以达到减少编码后数据长度的目的，从而提高存储效率。

1.2 项目要求

本项目的目标是实现一个霍夫曼编码算法，输入一组字符对应的使用频率，输出每个字符的霍夫曼编码。

1.3 项目示例



该示例存在每个前缀码前都有一个多余的 0 的问题，本项目解决了这个问题。

1.4 项目环境

使用 C++ 语言实现，开发环境为 Linux 下的 gcc 编译器。

2 项目设计

2.1 数据结构应用

在霍夫曼编码项目中，数据结构的选择对于算法的效率至关重要。我们需要一个能够快速比较节点频率并构建二叉树的数据结构。以下是选择和设计数据结构的分析过程：

- 节点频率存储：需要一个数据结构来存储节点的频率，以便后续构建霍夫曼树。
- 节点频率比较需求：霍夫曼编码的核心在于根据节点频率构建树，因此需要一个能够快速比较频率大小的数据结构。
- 自动排序：在添加新节点后，数据结构应能自动维持元素的有序状态，以便快速找到最小元

基于以上分析，我们确定了以下数据结构：

2.1.1 优先队列 *std :: priority_queue*

优先队列是一个能够自动排序的队列，它能够根据元素的优先级自动调整元素的顺序。在霍夫曼编码中，我们需要一个能够快速找到频率最小的节点的数据结构，优先队列正是这样一种数据结构。我们可以将霍夫曼树的节点放入优先队列中，队列会自动根据节点的频率排序，从而保证每次取出的节点都是频率最小的。以下是优先队列的一些关键特性：

（1）元素优先级：每个元素都有一个优先级，通常使用数值表示，数值越小优先级越高。

（2）出队顺序：元素出队时，总是优先级最高的元素先出队。

（3）入队操作：元素可以按照任意顺序入队。

（4）动态性：优先队列可以在运行时动态地添加和移除元素。

（5）无序性：优先队列内部元素的存储通常是无序的，即元素的存储顺序并不反映它们的优先级顺序。

2.1.2 HuffmanNode 结构体

HuffmanNode 结构体是霍夫曼树的基本单元，用于表示树中的每个节点。包括：

- int freq：存储节点的频率，即字符出现的频率。
- HuffmanNode *left：指向左子节点的指针。
- HuffmanNode *right：指向右子节点的指针。

节点的频率是霍夫曼编码中的关键信息，因此将其作为节点的首要属性。左右子节点指针用于构建二叉树结构，允许递归遍历和构建树。

2.1.3 compare 结构体

compare 结构体是一个比较器，用于优先队列中节点的比较。优先队列需要一个比较器来确定节点的顺序，compare 结构体提供了这种比较机制。通过重载 operator()，我们可以定义优先队列中

元素的排序规则，即频率较小的节点排在前面。

2.2 算法设计

2.2.1 算法思路

A. 初始化霍夫曼树节点（initializeTree）

该函数接收一个整数数组 freq[] 和其大小 size，初始化霍夫曼树的叶子节点，并将它们放入一个最小堆（优先队列）中。

B. 构建霍夫曼树（buildHuffmanTree）

该函数接收一个最小堆（优先队列）作为参数，通过贪心算法构建霍夫曼树。每次从堆中取出两个频率最小的节点，创建一个新的节点作为它们的父节点，并将新节点的频率设置为两个子节点频率之和。这个过程重复进行，直到堆中只剩下一个节点，这个节点就是霍夫曼树的根节点。

C. 打印霍夫曼编码（printHuffmanCodes）

该函数通过前序遍历 (preorder) 霍夫曼树，打印每个叶子节点的霍夫曼编码。对于每个叶子节点，它将节点的频率和对应的编码打印出来。

2.2.2 性能评估

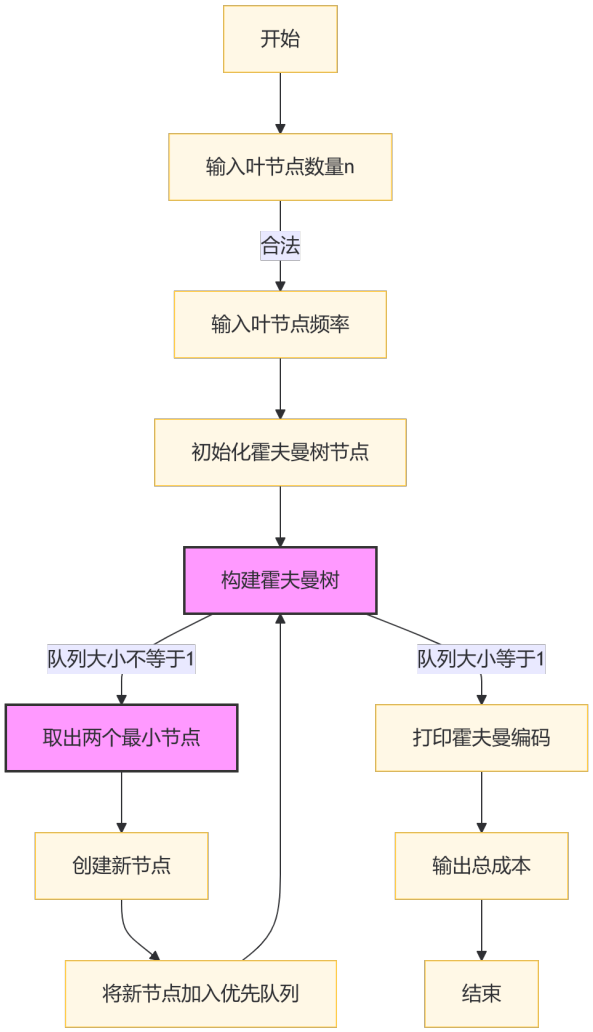
时间复杂度：

- 初始化霍夫曼树节点的时间复杂度为 $O(n)$，其中 n 为节点的数量。
- 构建霍夫曼树的时间复杂度为 $O(n \log n)$，这是因为每次从堆中取出两个节点并插入新节点后，需要进行堆的调整。
- 打印霍夫曼编码的时间复杂度为 $O(n)$，因为每个节点只被访问一次。

空间复杂度：

- 需要为每个节点分配空间，总空间复杂度为 $O(n)$。

2.2.3 流程图表示



2.2.4 代码实现

```cpp
#include <iostream>
#include <queue>
#include <vector>
#include <string>
#include <iomanip>
#include <limits>

using namespace std;

int total_cost = 0;

// 霍夫曼树节点
struct HuffmanNode {
    int freq;
    HuffmanNode *left, *right;
```

```
16
17      HuffmanNode(int freq) : freq(freq), left(nullptr), right(nullptr) {}
18  };
19
20  // 比较器，用于优先队列
21  struct compare {
22      bool operator()(HuffmanNode* l, HuffmanNode* r) {
23          return l->freq > r->freq;
24      }
25  };
26
27  // 递归前序遍历打印霍夫曼编码
28  void printCodes(HuffmanNode* root, string str) {
29      if (!root)
30          return;
31
32      if (!root->left && !root->right) // 叶子节点
33          cout << setw(2) << root->freq << ": " << str << "\n";
34
35      printCodes(root->left, str + "0");
36      printCodes(root->right, str + "1");
37  }
38
39  // 初始化霍夫曼树节点
40  priority_queue<HuffmanNode*, vector<HuffmanNode*>, compare> initializeTree(int freq[], int
    ↪  size) {
41      priority_queue<HuffmanNode*, vector<HuffmanNode*>, compare> minHeap;
42      for (int i = 0; i < size; ++i)
43          minHeap.push(new HuffmanNode(freq[i]));
44      return minHeap;
45  }
46
47  // 构建霍夫曼树
48  HuffmanNode* buildHuffmanTree(priority_queue<HuffmanNode*, vector<HuffmanNode*>, compare>&
    ↪  minHeap) {
49      HuffmanNode *left, *right, *top;
50
51      while (minHeap.size() != 1) {
52          left = minHeap.top();
53          minHeap.pop();
54
55          right = minHeap.top();
56          minHeap.pop();
57
58          top = new HuffmanNode(left->freq + right->freq);
59          top->left = left;
60          top->right = right;
61          total_cost += top->freq;
62          minHeap.push(top);
63      }
64
65      return minHeap.top();
66  }
67
68  // 打印霍夫曼编码
69  void printHuffmanCodes(HuffmanNode* root) {
70      printCodes(root, "");
71  }
72
```

```cpp
int main() {
    int n;
    cout << " 请输入叶节点的数量: ";
    while (true)
    {
        cin >> n;
        if (cin.fail() || n <= 0) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << " 输入错误, 请重新输入: ";
            continue;
        } else {
            break;
        }
    }

    int* freq = new int[n];

    cout << " 请依次输入叶节点: ";
    for (int i = 0; i < n; ++i) {
        cin >> freq[i];
        if (cin.fail() || freq[i] <= 0) {
            cerr << " 输入错误, 请重新输入第 " << i + 1 << " 个节点及以后的节点: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            --i;
        }
    }

    auto minHeap = initializeTree(freq, n);
    HuffmanNode* root = buildHuffmanTree(minHeap);
    cout << " 霍夫曼编码: \n";
    printHuffmanCodes(root);
//  cout << " 总花销为: " << total_cost << endl;
    delete[] freq;

    return 0;
}
```

3 项目测试

3.1 正常测试

```
请输入叶节点的数量：5
请依次输入叶节点：5 9 12 13 16
霍夫曼编码：
12: 00
13: 01
 5: 100
 9: 101
16: 11
```

```
请输入叶节点的数量：13
请依次输入叶节点：2 3 5 7 11 13 17 19 23 29 31 37 41
霍夫曼编码：
19: 000
23: 001
11: 0100
13: 0101
29: 011
31: 100
17: 1010
 7: 10110
 5: 101110
 2: 1011110
 3: 1011111
37: 110
41: 111
```

修复了之前的 bug，程序能够正确输出结果，而不包含多余的前缀 0。

只有一个节点的情况　　只有一个节点的情况下，节点即根节点，无霍夫曼编码，如下图所示：

```
请输入叶节点的数量：1
请依次输入叶节点：6
霍夫曼编码：
 6:
```

3.2 健壮性测试

程序具有良好的健壮性，对于输入的不合法数据，程序能够给出合理的提示，如下图所示：

```
请输入叶节点的数量：0
输入错误，请重新输入：-2
输入错误，请重新输入：q
输入错误，请重新输入：3
请依次输入叶节点：1 e
输入错误，请重新输入第 2 个节点及以后的节点：3 q
输入错误，请重新输入第 3 个节点及以后的节点：-2
输入错误，请重新输入第 3 个节点及以后的节点：5
霍夫曼编码：
 1: 00
 3: 01
 5: 1
```

4 心得体会

在开发霍夫曼编码项目的过程中，我不仅深入掌握了霍夫曼编码的理论知识，还通过实践加深了对其内在机制的理解。霍夫曼编码是一种基于字符频率的最优前缀编码方法，它通过构建霍夫曼树来为不同字符分配不同长度的编码，从而实现数据压缩。我学会了如何根据字符出现的频率来构建这棵树，以及如何利用树的结构来生成编码，这些都是信息论领域的重要概念。

通过使用 C++ 标准库中的 <queue> 和 <vector> ，我实现了一个优先队列来存储和处理霍夫曼树的节点。这个优先队列帮助我以效率极高的方式选择频率最低的节点，这是构建霍夫曼树的关键步骤。我还使用了递归遍历来生成霍夫曼编码，这种方法不仅代码简洁，而且直观地体现了树结构的深度优先搜索特性。

在这个项目中，我也提升了自己的编程技巧。我学会了如何更有效地管理内存，尤其是在处理动态分配的内存时。我意识到，对于每一个新创建的 'HuffmanNode'，都需要在适当的时候释放内存，以避免内存泄漏。此外，我还学习了如何通过递归函数来处理树结构，这是一种处理复杂数据结构的强大工具。

项目中，我还特别关注了输入验证的重要性。我设计了多个检查点来确保用户输入的数据是合法的，比如检查输入的叶节点数量是否为正数，以及每个叶节点的频率是否大于零。这些检查提高了程序的健壮性，确保了即使在面对错误或异常输入时，程序也能正常运行或给出合理的错误提示。

总的来说，这个项目不仅让我在技术层面上有所成长，也让我对软件开发的各个方面有了更深的认识。我学会了如何将理论知识应用到实际问题中，如何解决实际编程中遇到的问题，以及如何编写出既高效又健壮的代码。这个项目无疑是一个宝贵的学习经历，它不仅提升了我的技术能力，也增强了我对复杂问题解决能力的信心。