**项目说明文档**

**离散数学课程实验**

**——最优二元树**

作 者 姓 名 刘淑仪

学 号 2251730

指 导 教 师 唐剑锋

学 院 专 业 软件学院 软件工程



二〇二三 年 十二 月 二十二 日

目录

1 项目分析 1

1.1 实验目的分析 1

1.2 实验内容概述 1

2 项目设计与实现 1

2.1 实验原理 1

2.1.1 题目涉及知识 1

2.1.2 具体实验原理 2

2.2 实验过程设计 2

2.2.1 实验思路简述 2

2.2.2 实验设计流程图 3

2.3 项目具体实现 4

3 项目实现与测试 4

4 源代码展示 5

5 集成开发环境与编译运行环境 8

# 项目分析

## 实验目的分析

该实验课程的目标是培养学生对数据压缩技术和信息编码的深刻理解。通过实现哈夫曼编码，学生将学习如何根据通信符号的使用频率构建最优二元树（哈夫曼树），这是一种高效的数据压缩方法。实验不仅让学生掌握如何将理论应用于实际问题，还增进了他们对贪心算法、二叉树等数据结构的理解。

## 实验内容概述

输入一组通信符号的使用频率，求各通信符号对应的前缀码。

Ⅰ、输入结点数量（要求为大于1的正整数）。

Ⅱ、输入各个结点的权值。

Ⅲ、根据输入的信息构建哈夫曼树（即最优二元树），并打印。

Ⅳ、程序结束。

# 项目设计与实现

## 实验原理

### 题目涉及知识

1. 哈夫曼树（Huffman Tree）：

在给定N个权值作为N个叶子节点的情况下，构造一棵二叉树。当该树的带权路径长度（Weighted Path Length）达到最小时，这样的二叉树被称为最优二叉树，亦即哈夫曼树。哈夫曼树特征在于其是带权路径长度最短的树形结构，其中权值较大的节点离根节点更近。

2.哈夫曼编码（Huffman Coding）：

哈夫曼编码是一种编码机制，属于可变字长编码（Variable-Length Coding, VLC）的一种形式。1952年，Huffman提出了这种依据字符出现概率构造的编码方法，旨在构建平均码长最短的不同前缀码字。这种方法通常被认为是最优的编码方式，因此也被称作哈夫曼编码，有时亦称霍夫曼编码。

3.最小堆（Min Heap）：

堆是一种特殊的完全二叉树，其中每个非叶子节点的值都不大于其左右子节点的值。最大堆（Max Heap）和最小堆是二叉堆的两种主要形态。在最小堆中，根节点的键值是所有堆节点键值中的最小值，确保树的根始终包含最小元素。这种数据结构在多种算法，包括哈夫曼编码的实现中，扮演着关键角色。

### 具体实验原理

Ⅰ、构建哈夫曼树

将给定的N个权值作为N个叶子节点，每个节点可视为一棵独立的二叉树。在当前森林（所有树的集合）中找出两个根节点权值最小的树。将这两棵树合并为一棵新的二叉树，新树的根节点权值为两个子树根节点权值之和。重复此过程，直到森林中只剩下一棵树，这棵树就是哈夫曼树。

Ⅱ、生成哈夫曼编码

每个字符的哈夫曼编码可以通过从哈夫曼树根节点到该字符所在叶子节点的路径来确定：沿树向下移动时，每经过一个左子节点，就在编码中添加一个“0”；每经过一个右子节点，就添加一个“1”。到达叶子节点时，所形成的一串0和1的序列就是该节点字符的哈夫曼编码。

## 实验过程设计

### 2.2.1 实验思路简述

Ⅰ.进行数据结构的定义：

1. Node结构：定义了树的节点，包括存储字符（data），字符频率（freq），以及左右子节点指针（left, right）。
2. compare结构：用于优先队列，定义了节点比较的规则。这里规定了频率越小的节点在优先队列中优先级越高。
3. HuffmanTree类：封装了哈夫曼树的构建和编码输出功能。

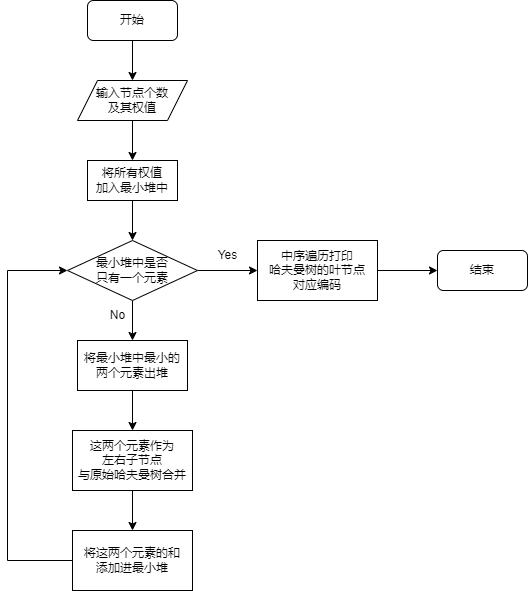
Ⅱ.进行哈夫曼树的构建：

在HuffmanTree类中，buildTree方法通过接受一组字符及其频率，构建哈夫曼树。使用最小堆（优先队列）来存储和操作节点，保证了每次都能高效地找到频率最小的节点。通过合并频率最小的两个节点构建新节点，并重复此过程直到只剩下一个节点（树的根节点）。

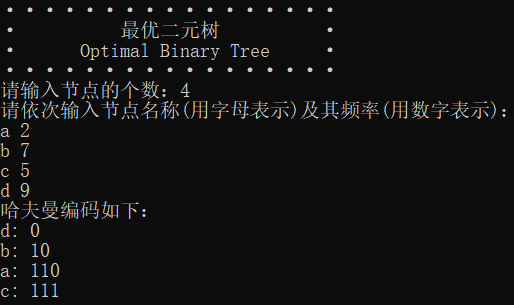
Ⅲ.输出哈夫曼编码：

printCodes方法用于递归中序遍历哈夫曼树，并打印从根到每个叶节点的路径（即哈夫曼编码）。对于每个叶节点（代表一个字符），方法会输出该字符及其对应的哈夫曼编码。对于权值相同的结点，将其位置调换也仍然是最优二叉树，都满足哈夫曼树的定义。因此，哈夫曼树不唯一，哈夫曼编码不唯一，以上流程只是给出其中一种树的构建过程便于理解哈夫曼树的构建。

### 2.2.2 实验设计流程图



## 项目具体实现



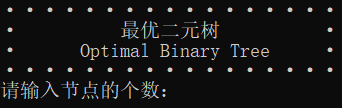
# 项目实现与测试

结果分析：

输入阶段控制输入节点个数为大于1的正整数，若个数小于等于0，为不合理输入；若输入为1，则只有一个节点需要进行哈夫曼编码，但只有一个节点时没有哈夫曼编码压缩的意义。

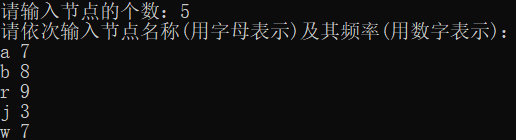
正确输入后，根据输入的节点数量及其权值，构建哈夫曼树，按中序遍历打印其叶结点的编码（输出为这串数字序列的其中一种哈夫曼编码结果）。

1.进入界面

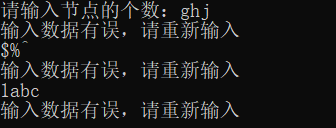


2.输入节点个数及其对应频率

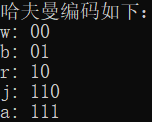
2.1.正常输入



2.2.错误输入



3.输出边及其权重



# 源代码展示

#include <iostream>

#include <queue>

#include <vector>

#include <string>

using namespace std;

// 树节点的结构

template <class D,class F>

struct Node {

    D data; // 存储字符

    F freq; // 存储频率

    Node\* left, \* right; // 左右子节点

    Node(D data, F freq) {

        left = right = nullptr;

        this->data = data;

        this->freq = freq;

    }

};

// 用于优先队列的比较函数

template <class D, class F>

struct compare {

    // 重载括号运算符

    bool operator()(Node<D,F>\* l, Node<D,F>\* r) {

        // 高频率的节点应该排在队列的后面

        return (l->freq > r->freq);

    }

};

// 哈夫曼树类，包含构建和打印树的方法

template <class D, class F>

class HuffmanTree {

private:

    Node<D, F>\* root; // 哈夫曼树的根

    // 递归函数，用于打印从根到每个叶子节点的路径（哈夫曼编码）

    void printCodes(Node<D, F>\* root, string str) {

        if (!root) return;

        // 如果是叶子节点，打印字符和对应的编码

        if (root->data != '$')

            cout << root->data << ": " << str << "\n";

        // 递归遍历左右子树

        printCodes(root->left, str + "0");

        printCodes(root->right, str + "1");

    }

public:

    // 构造函数

    HuffmanTree() : root(nullptr) {}

    // 构建哈夫曼树

    void buildTree(const vector<char>& data, const vector<int>& freq, int size) {

        // 创建一个优先队列（最小堆）

        priority\_queue<Node<D, F>\*, vector<Node<D, F>\*>, compare<D,F>> minHeap;

        // 将所有字符及其频率添加到最小堆中

        for (int i = 0; i < size; ++i) {

            minHeap.push(new Node<D,F>(data[i], freq[i]));

        }

        // 循环，直到堆中只剩一个节点（树的根）

        while (minHeap.size() != 1) {

            Node<D, F>\* left = minHeap.top(); minHeap.pop();

            Node<D, F>\* right = minHeap.top(); minHeap.pop();

            // 创建一个新节点，其频率是两个最小节点的总和

            Node<D, F>\* top = new Node<D, F>('$', left->freq + right->freq);

            top->left = left;

            top->right = right;

            minHeap.push(top);

        }

        root = minHeap.top(); // 根节点

    }

    // 打印哈夫曼树的编码

    void printHuffmanCodes() {

        cout << "哈夫曼编码如下：\n";

        printCodes(root, "");

    }

};

// 输入处理函数

static int inputNumber() {

    int inputNum = 0;

    while (true) {

        std::cin >> inputNum;

        if (std::cin.fail() || inputNum < 0 || inputNum > 65536) {

            // 输入验证

            std::cin.clear();

            std::cin.ignore(65536, '\n');

            std::cout << "输入数据有误，请重新输入\n";

        }

        else {

            if (std::cin.get() == '\n')

                break;

            else {

                std::cin.clear();

                std::cin.ignore(65536, '\n');

                std::cout << "输入数据有误，请重新输入\n";

            }

        }

    }

    return inputNum;

}

void menu()

{

    std::cout << "·················\n";

    std::cout << "·          最优二元树          ·\n";

    std::cout << "·      Optimal Binary Tree     ·\n";

    std::cout << "·················\n";

}

int main()

{

    HuffmanTree<char, int> huffmanTree;

    int nodeNum = 0;

    char inputName;

    int inputFreq;

    vector<char> nodeName;

    vector<int> nodeFreq;

    menu();

    std::cout << "请输入节点的个数：";

    nodeNum = inputNumber();

    std::cout << "请依次输入节点名称(用字母表示)及其频率(用数字表示)：\n";

    for (int i = 0; i < nodeNum; i++) {

        while (true) {

            std::cin >> inputName >> inputFreq;

            if ((inputName >= 'a' && inputName <= 'z') || (inputName >= 'A' && inputName <= 'Z')) {

                if (inputFreq >= 0)

                    break;

                else

                    std::cout << "节点频率不可以为负数，请重新输入：";

            }

            else

                std::cout << "节点名称输入错误，请重新输入：";

        }

        nodeName.push\_back(inputName);

        nodeFreq.push\_back(inputFreq);

    }

    huffmanTree.buildTree(nodeName, nodeFreq, nodeName.size());

    huffmanTree.printHuffmanCodes();

    return 0;

}

# 集成开发环境与编译运行环境

Windows系统：Windows 11 x64

Windows集成开发环境：Microsoft Visual Studio 2022 (Debug模式)

Windows编译运行环境：本项目适用于x86架构和x64架构