沈阳航空航天大学

课 程 设 计 报 告

课程设计名称：数据结构课程设计

课程设计题目：通讯发报应用

学　　院	计算机学院
专　　业	计算机科学与技术
班　　级	计算机 1803
学　　号	183424080320
姓　　名	刘上
指导教师	王丹

2021 年 1 月

目　录

1 题目介绍

1.1 问题描述

在通讯发报应用中，需要让应用对输入的字符集以及字符频度进行处理，构造哈夫曼树，并进行哈夫曼编码。比如输入以下内容：

字符集：A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z

字符频度：186，64，13，22，32，103，21，15，47，571，5，32，20，57，63，15，1，48，51，80，23，7，18，2，16，38

1.1.1 问题背景

在通讯发报中，需要对发出的字符串进行缩短，进行最大化的无损压缩，由此想到构造哈夫曼树，并进行哈夫曼编码，频度最高的字母使用较短的编码，频度低的字母使用较长的编码。

1.1.2 主要任务

(1) 由用户来输入初始字符集、相应字符及字符频度。

(2) 输入一个要发报的字符串，将其编码后发报。

(3) 接收一组发报信息，将其还原为字符序列。

1.2 问题分析

需要解决的问题：

(1)如何使用户输入数据并进行有效的存储？

(2)如何构造一棵哈夫曼树并对结点进行哈夫曼编码？

(3)如何对输入的错误数据进行容错性处理？

求解方法：

(1)使用基本的 cin，getchar()获取输入并存储于 vector 中。

(2)使用递归来构造哈夫曼树，使用层次遍历的方法来对结点进行哈夫曼编码。

(3)对待错误的信息进行直接解码，然后对解码信息进行判断。

2　系统总体设计

2.1　系统总体功能

通过问题分析后，设计通讯发报应用分为 5 个模块，分别为输入字符和字符频度，展示字符和字符频度，编码，解码，退出程序。

输入字符和字符频度模块中包含输入数据，哈希表存储，构造哈夫曼树。构造哈夫曼树中包含入队，出队，获取元素个数操作。

展示字符和字符频度包含哈希表遍历。

编码中包含输入编码值，树的先序遍历，输出解码值。

解码中包含输入编码值，树的遍历，输出解码值，其中树的遍历依据二进制编码，从而选择向左还是向右对哈夫曼树进行遍历。

最后是退出程序。如图 2.1 所示。

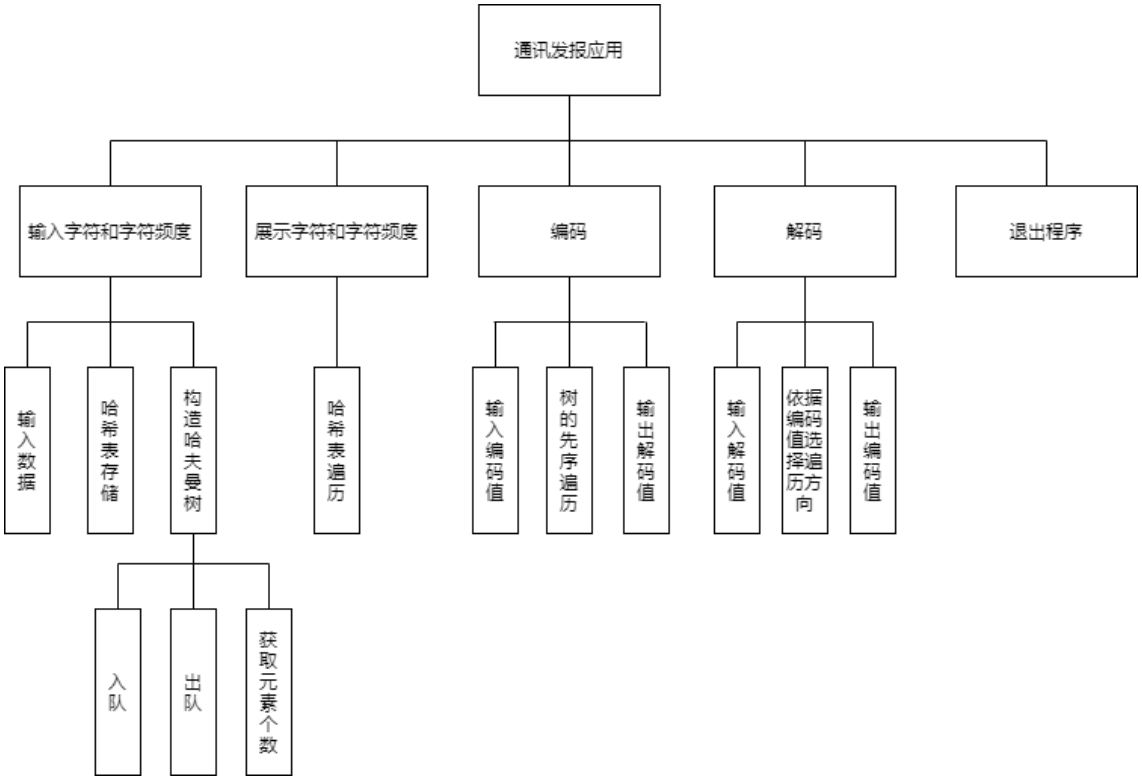

图 2.1 系统总体功能模块

2.2 系统总体流程

程序总体分为 5 个模块，分别是输入字符和字符频度，展示字符和字符频度，编码，解码，退出程序。

程序最开始执行的时候可以通过 switch 选择来决定进入哪个模块,选择后则执行对应模块的程序。

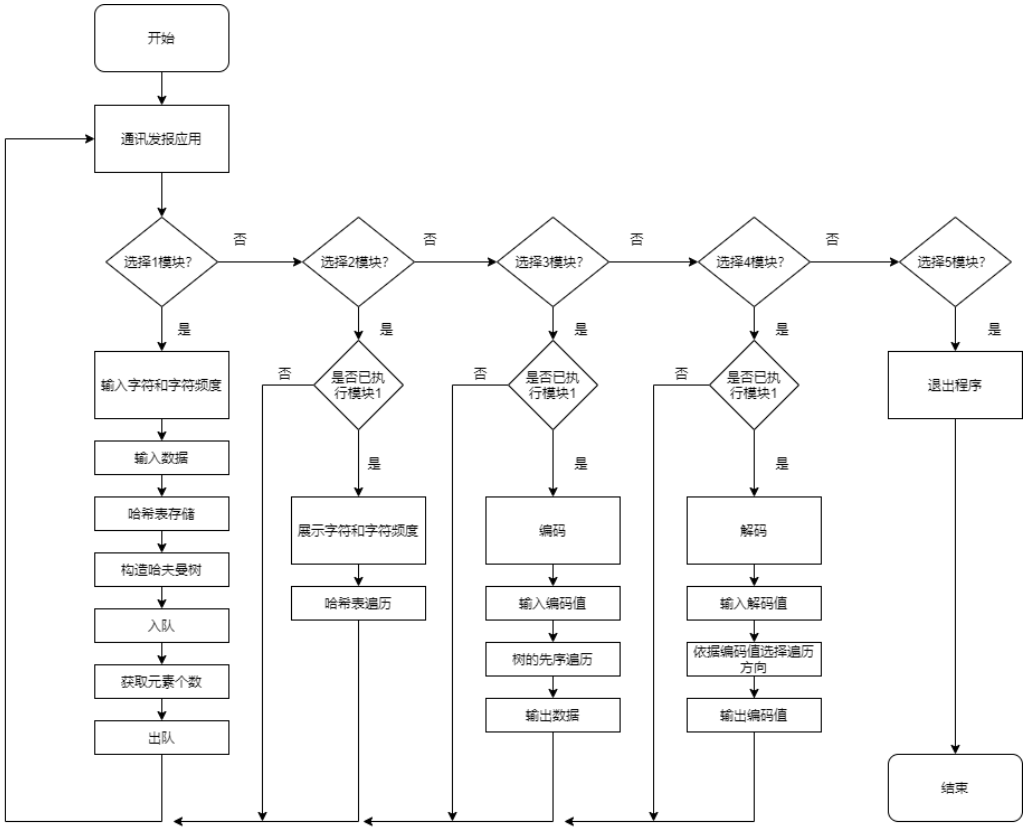

图 2.2 系统总体流程

3 数据结构设计

3.1 Huffman 类

```
class Huffman
{
public:
    Huffman();   // 构造函数
    ~Huffman(); // 析构函数

    void getMap();   // 输入 Huffman 映射表
    void showMap(); // 显示 Huffman 映射表

    void encodeElement();   // 编码
    void decodeElement();   // 解码

    void setEncode(HuffNode * head, char data);         // 输出编码后的信息
    void setDecode(HuffNode * head, string encode);   // 输出解码后的信息

    void createTree();                          // 创建 Huffman

    void preOrder(HuffNode * head);         // 先序遍历

    void clearData();                          // 清除特定数据

private:
    map<char, int> huffman_map;         // Huffman 映射表
    map<char, int>::iterator iter;             // Huffman 映射表遍历
```

```
    vector<char> huffdata;              // 输入的元素
    vector<HuffNode *> huffnodes;  // 输入的元素

    string encode;   // 编码后的信息
    string decode;   // 解码后的信息

    HuffNode * huffHeadPtr;              // Huffman 头节点
};
```

3.2 Huffman Node 结构体

```
struct HuffNode
{
    char data;  // 存储的信息
    int weight; // 本结点对应的权值
    string huff_code;   // Huffman 编码值
    HuffNode * left;    // 左结点
    HuffNode * right;   // 右结点
};
```

定义了 6 个变量，分别为字符类型存储的数据，整型本结点对应的权值，字符串类型 Huffman 编码值，HuffNode 指针类型的左结点，HuffNode 指针类型的右结点。

4 功能模块设计

4.1 输入字符以及对应的频度模块

本模块中主要用到的是 void getMap()和 void createTree()函数，使用 void getMap()来调用 void createTree()。其次，还包含了对哈希表插入和遍历，对队列的入队，出队以及获取元素个数。

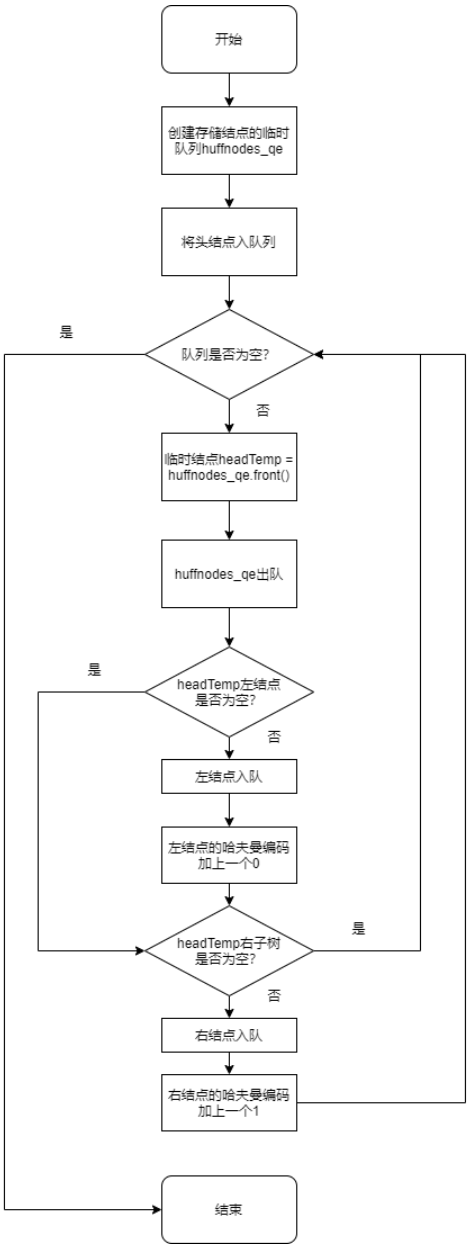

图 4.1 输入字符以及对应的字符频度模块流程图

4.2 解码模块

解码模块中主要用到 void decodeElement()和 void setDecode(HuffNode * head, string encode)函数，使用 void decodeElement()来调用 void setDecode(HuffNode * head, string encode)，从而启动解码流程。

在解码中，对输入的二进制编码进行逐次遍历，如何遇到 0，则将头指针向左结点移动；如果遇到 1，则将头指针向右结点移动；如果遇到叶子结点，则取出叶子结点存储的数据，并将头指针移动到根节点，准备下一次哈夫曼树的遍历。

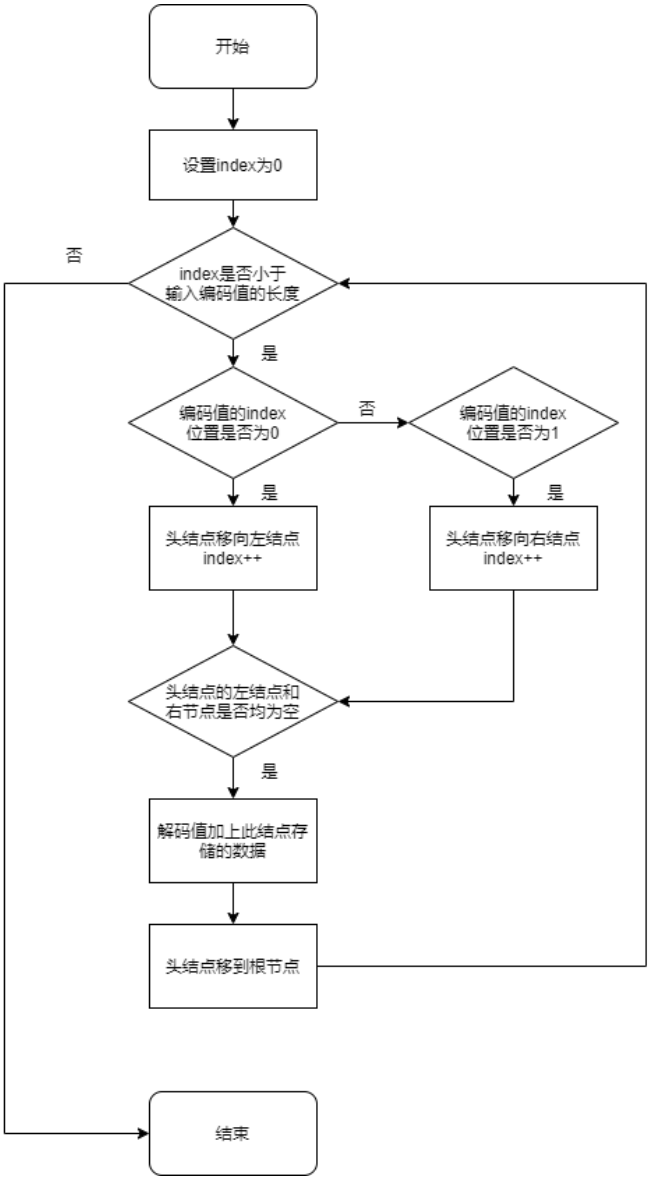

图 4.2 解码模块流程图

5 系统测试与运行结果

5.1 调试及调试分析

(1) 逗号残留问题

问题描述：输入数据时，两个数字中间的逗号不好处理，影响数据的输入。

解决方法：每输入一个数据，就用一个 flag 变量来 getchar()逗号，将逗号吸收，不影响下一个数据的输入。

(2) 如何给字符编码问题

问题描述：在构造完哈夫曼树并存储后，应当采取怎样的算法对每个字符进行编码？

解决方法：使用层次遍历，将根节点压入队列，并依次压入左结点和右结点，并对左结点和右结点进行编码。

(3) 如何找出输入字符的编码值问题

问题描述：输入字符后，应该使用什么算法对字符进行编码？

解决方法：使用先序遍历，将每个输入的字符与结点存储的字符进行比较，相同则录入将要输出的编码值中。

5.2 测试用例

设计了三组测试用例，分别为：

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:3
C,B,D
编码后的信息:0000000011011010
```

图 5.2.1 编码测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:3
Z,Z,S
编码后的信息:010010100110110
```

图 5.2.2 编码测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:4
0000000011011010
解码后的信息:CBD
```

图 5.2.3 解码测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:4
010010100110110
解码后的信息:ZZS
```

图 5.2.4 解码测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:4
110100
解码后的信息:J
```

图 5.2.5 容错性测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:4
010011
解码后的信息:Z
```

图 5.2.6 容错性测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:4
24689
非法数据输入！程序退出
```

图 5.2.7 非法数据测试

```
D:\Download\Huffman-main>main.o
[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:1
A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z
186,64,13,22,32,103,21,15,47,571,5,32,20,57,63,15,1,48,51,80,23,7,18,2,16,38

[1] Input Huffman hash map
[2] Show out Huffman hash map
[3] Encode element
[4] Decode element
[5] Exit

Plz choose:4
000010100
解码后的信息:Q
```

图 5.2.8 边界数据解码测试

表 5.1 系统测试

测试项	测试用例	预期输出	实际输出
编码测试	C,B,D	00000000110111010	000000001101 1010
编码测试	Z,Z,S	0100101001101 10	010010100110 110
解码测试	000000001101 1010	CBD	CBD
解码测试	010010100110 110	ZZS	ZZS
容错性测试	110100	J	J
容错性测试	010011	Z	Z
非法数据测试	24689	程序退出	程序退出
边界数据解码测试	000010100	Q	Q

6 总 结

两周的课设基本结束了，在这次的课设中不仅检验了我所学的知识，也培养了我如何在课设中运用到课堂上所学的知识，培养了我解决问题的能力。在设计过程中，对于面向对象编程和模块化编程有了更加充分的理解。

在本次的课设过程中，我更加熟练的掌握了 Linux 下 C++程序的开发，对于 C++ STL 中的哈希表，队列，容器有了更加充分的理解和更加熟练的运用。本次的程序基本上符合课设的要求，对于输入的字符能够进行准确的哈夫曼树的建立并进行哈夫曼编码，模块化的设计使得程序在往后也能够方便地进行维护。同时也对于树的遍历有了更深的理解和运用，使用层次遍历可以更好的对结点进行哈夫曼编码。

但是程序也存在着一些不足的地方，比如数据还暂时不能保存在硬盘上，对于错误数据的处理不具备较高的容错度，对于错误数据的输入也存在处理不佳的情况。这些问题在以后的编程设计过程中都应当注意。

参考文献

[1] 李四老师，哈夫曼树和哈夫曼编码的详细介绍以及代码实现，CSDN：https://blog.csdn.net/qq_29542611/article/details/79334308，2018

[2] Wiki，霍夫曼编码，Wiki：https://zh.wikipedia.org/wiki/%E9%9C%8D%E5%A4%AB%E6%9B%BC%E7%BC%96%E7%A0%81，2020

[3] 严蔚敏，数据结构 C 语言版，北京：清华大学出版社，2007

[4] 谭浩强，C 程序设计，北京：清华大学出版社，2017

附　录

（程序清单）

Main.cpp

```
#include "Huffman.h"

int main()
{
    int choose;
    Huffman huffman;

    while(true)
    {
        cout << "[1] Input Huffman hash map" << endl;
        cout << "[2] Show out Huffman hash map" << endl;
        cout << "[3] Encode element" << endl;
        cout << "[4] Decode element" << endl;
        cout << "[5] Exit" << endl << endl;

        cout << "Plz choose:";
        cin >> choose;

        switch(choose)
        {
            case 1:
                huffman.getMap();
                break;
            case 2:
                huffman.showMap();
                break;
            case 3:
                huffman.encodeElement();
                break;
            case 4:
                huffman.decodeElement();
                break;
            case 5:
                exit(0);
                break;
```

```
            default:
                cout << "Plz consider another choose!" << endl;
                break;
        }

        cout << endl;
        huffman.clearData();    // 清除特定数据
    }
    return 0;
}
```

Huffman.h

```
#ifndef MAIN_HUFFMAN_H
#define MAIN_HUFFMAN_H

#include <iostream>
#include <cstdio>
#include <vector>
#include <queue>
#include <map>
#include <string>
#include <algorithm>

using namespace std;

struct HuffNode
{
    char data;  // 信息
    int weight; // 权值
    string huff_code;   // Huffman 编码值
    HuffNode * left;    // 左节点
    HuffNode * right;   // 右节点
};

class Huffman
{
public:
    Huffman();  // 构造函数
    ~Huffman(); // 析构函数
```

```
    void getMap();  // 输入 Huffman 映射表
    void showMap(); // 显示 Huffman 映射表

    void encodeElement();  // 编码
    void decodeElement();  // 解码

    void setEncode(HuffNode * head, char data);     // 输出编码后的信息
    void setDecode(HuffNode * head, string encode); // 输出解码后的信息

    void createTree();                  // 创建 Huffman

    void preOrder(HuffNode * head);     // 先序遍历

    void clearData();   // 清除特定数据

private:
    map<char, int> huffman_map;     // Huffman 映射表
    map<char, int>::iterator iter;  // Huffman 映射表遍历

    vector<char> huffdata;  //  输入的元素
    vector<HuffNode *> huffnodes; // 输入的元素

    string encode;  // 编码后的信息
    string decode;  // 解码后的信息

    HuffNode * huffHeadPtr;          // Huffman 头节点
};

#endif //MAIN_HUFFMAN_H
```

Huffman.cpp

```
#include "Huffman.h"

// @brief 排序函数
bool sortByWeight(HuffNode * first, HuffNode * second)
{
    return first->weight < second->weight;
}

// @brief 构造函数
Huffman::Huffman()
{
```

```
    huffHeadPtr = nullptr; // 头节点指向空
}

// @brief 析构函数
Huffman::~Huffman()
{
    // TODO:递归删除哈弗曼树
    delete huffHeadPtr;    // 删除头节点
    encode = "";
}

// @brief 输入 Huffman 映射表
void Huffman::getMap()
{
    char flag;  // 判断回车标志
    char input_char;    // 输入的字符
    int input_weight;  // 输入的权重

    while(true)
    {
        cin >> input_char;
        huffman_map.insert(pair<char, int>(input_char, -1));    // 插入字符

        if((flag = getchar()) == '\n')
            break;
    }

    for(iter = huffman_map.begin(); iter != huffman_map.end(); iter++)
    {
        cin >> input_weight;
        iter->second = input_weight;

        flag = getchar();   // 吸收逗号
    }

    // 创建初始化容器
    for(iter = huffman_map.begin(); iter != huffman_map.end(); iter++)
    {
        HuffNode * temp = new HuffNode;

        temp->data = iter->first;
        temp->weight = iter->second;
        temp->left = nullptr;
```

```
        temp->right = nullptr;

        huffnodes.push_back(temp);
    }

    createTree();   // 构造 huffman 树

    // 使用层次遍历 构造每一个节点 Huffman 编码
    if(huffHeadPtr == nullptr)
        return;

    HuffNode * headTemp = huffHeadPtr;
    queue<HuffNode *> huffnodes_qe;
    huffnodes_qe.push(headTemp);

    while(huffnodes_qe.size() > 0)
    {
        headTemp = huffnodes_qe.front();
        huffnodes_qe.pop();

        if(headTemp->left)
        {
            huffnodes_qe.push(headTemp->left);
            headTemp->left->huff_code = headTemp->huff_code + "0";
        }
        if(headTemp->right)
        {
            huffnodes_qe.push(headTemp->right);
            headTemp->right->huff_code = headTemp->huff_code + "1";
        }
    }

    preOrder(huffHeadPtr);    // 使用遍历来验证 Huffman
}

// @brief 打印 Huffman 映射表
void Huffman::showMap()
{
//    cout << huffman_map.size() << endl;
    for(iter = huffman_map.begin(); iter != huffman_map.end(); iter++)
    {
        cout << iter->first << " " << iter->second << endl;
    }
```

```
    cout << endl;
}

// @brief 编码
void Huffman::encodeElement()
{
    char flag;
    char input;

    // input elements
    while(true)
    {
        cin >> input;
        huffdata.push_back(input);

        if((flag = getchar()) == '\n')
            break;
    }

    for(auto data : huffdata)
        setEncode(huffHeadPtr, data);

    cout << "编码后的信息:" << encode << endl;
}

// @brief 解码
void Huffman::decodeElement()
{
    cin >> encode;

    setDecode(huffHeadPtr, encode);

    cout << "解码后的信息:" << decode << endl;
}

// @brief 设置编码
void Huffman::setEncode(HuffNode * head, char data)
{
    if(head)
    {
        if(head->data == data)
            encode += head->huff_code;
        setEncode(head->left, data);
```

```
        setEncode(head->right, data);
    }
}

// @brief 设置解码
void Huffman::setDecode(HuffNode * head, string encode)
{
    int index = 0;
    HuffNode * temp = huffHeadPtr;
    while(index < encode.size())
    {
        if(encode.at(index) == '0')
        {
            temp = temp->left;
            index++;
        }
        else if(encode.at(index) == '1')
        {
            temp = temp->right;
            index++;
        }
        else
        {
            cout << "非法数据输入，程序退出" << endl;
            exit(0);
        }

        if(temp->left == nullptr && temp->right == nullptr)
        {
            decode += temp->data;
            temp = huffHeadPtr;
        }
    }
}

// @brief 创建 Huffman 树
void Huffman::createTree()
{
    while(huffnodes.size() > 0)
    {
        // 按照 weight 从小到大进行排序
        sort(huffnodes.begin(), huffnodes.end(), sortByWeight);
```

```
        // 取出前两个节点
        if(huffnodes.size() == 1)
        {
            huffHeadPtr = huffnodes.at(0);
            huffnodes.erase(huffnodes.begin()); // 删除
        }
        else
        {
            // 取出前两个
            HuffNode * node_1 = huffnodes.at(0);
            HuffNode * node_2 = huffnodes.at(1);
            // 删除
            huffnodes.erase(huffnodes.begin());
            huffnodes.erase(huffnodes.begin());
            // 生成新的节点
            HuffNode * temp = new HuffNode;
            temp->weight = node_1->weight + node_2->weight;

            (node_1->weight < node_2->weight) ?
            (temp->left = node_1, temp->right = node_2) :
            (temp->left = node_2, temp->right = node_1);

            huffnodes.push_back(temp);
        }
    }
}

// @brief 使用先序遍历出叶子节点的Huffman值
void Huffman::preOrder(HuffNode *head)
{
    if(head)
    {
        if(head->left == nullptr && head->right == nullptr)
            cout << head->data << " " << head->huff_code << endl;
        preOrder(head->left);
        preOrder(head->right);
    }
}

// @brief 清除特定数据
void Huffman::clearData()
{
    huffdata.clear();
```

```
    huffnodes.clear();

    encode = "";
    decode = "";
}
```