石 家 庄 铁 道 大 学

**实 验 报 告**

**实习三 二叉树应用**

题 目： 哈夫曼编译码

班 级： 信1901 –4

姓 名： 闫竞存

学 号： 20194127

日 期：­ 2020.10.30

**1. 实验题目**

哈夫曼编/译码器

[问题描述]

利用哈夫曼编码进行通信可以大大提高信道利用率，这要求在发送端通过一个编码系统对待传输预先编码，在接收端将传来的数据进行译码。对于双工通道，每端都需要一个完整的编/译码系统。

[基本要求]

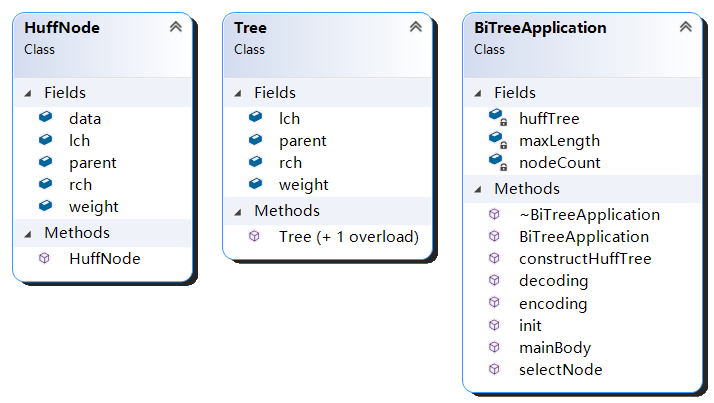
试为这样的信息收发站写一个哈夫曼码的编/译码系统。

**2. 需求分析**

设计一个哈夫曼编码译码系统，实现对字符串的编码和解码

**3. 概要设计**

**本程序的类图如下：**



**4. 关键代码的详细设计**

构建哈夫曼树

void BiTreeApplication::init() {

        cout << "----------------\n正在初始化二叉树\n----------------" << endl;

        // enter chars and its weight;

        cout << "输入字符和其权值，中间以空格隔开，EOF 代表结束" << endl;

        char tempChar;

        int tempWeight;

        for (this->nodeCount = 1; cin >> tempChar >> tempWeight; nodeCount++) {

                this->huffTree[nodeCount].data = tempChar;

                this->huffTree[nodeCount].weight = tempWeight;

        }

        this->nodeCount--; //减去在循环的时候多出来的那个

        this->maxLength = this->nodeCount \* 2 - 1;

        cout << "----------------\n提示\n\n输入字母代码成功\n----------------" << endl;

        cin.clear();

        this->constructHuffTree();

}

void BiTreeApplication::constructHuffTree() {

        cout << "正在构建二叉树..." << endl;

        for (int i = ++this->nodeCount; i <= this->maxLength; i++) {

                // 将s1和s2赋值为最大int值方便查找

                int s1 = 0, s2 = 0;

                this->selectNode(s1, i - 1, s2);

                this->huffTree[s1].parent = i;

                this->huffTree[s2].parent = i;

                this->huffTree[i].lch = s1;

                this->huffTree[i].rch = s2;

                this->huffTree[i].weight = this->huffTree[s1].weight + this->huffTree[s2].weight;

        }

        cout << "--------------\n提示：\n\n二叉树构建完毕\n--------------" << endl;

        system("pause");

        system("cls");

        this->hasConstruct = 1;

}

哈夫曼编码

void BiTreeApplication::encoding() {

    if (!this->hasConstruct) {

        cout << "---------------------------\n提示：\n\n二叉树尚未建立, 请执行选项2\n---------------------------" << endl;

        system("pause");

        system("cls");

        return;

    }

    cout << "输入字符(不含空格): ";

    string str;

    cin.clear();

    cin >> str;

    cout << "\n哈夫曼编码为 :" << endl;

    for (int i = 0; i < str.length(); i++) {

        // 遍历树，寻找数据节点

        string tempCode = "";

        int flag = 0; // 有没有找到这个字母所在的节点

        for (int j = 1; j < this->nodeCount; j++) {

            if (this->huffTree[j].data == str[i]) {

                // show

                flag = 1;

                int tempNumber = j;

                int parent = this->huffTree[j].parent;

                while (parent != 0) {

                    // left child is the node

                    if (this->huffTree[parent].lch == tempNumber) tempCode += "0";

                    if (this->huffTree[parent].rch == tempNumber) tempCode += "1";

                    tempNumber = parent;

                    parent = this->huffTree[parent].parent;

                }

                //cout << tempCode;

                break;

            }

        }

        // 在树中没找到这个节点

        if (!flag) {

            cout << "\n\n警告：\n\n未找到节点" << str[i] << " 程序即将返回上一层\n" << endl;

            system("pause");

            system("cls");

            return;

        }

        // 反转10序列然后输出

        reverse(tempCode.begin(), tempCode.end());

        cout << tempCode;

    }

    cout << endl;

    system("pause");

    system("cls");

}

哈夫曼解码

void BiTreeApplication::decoding() {

        if (!this->hasConstruct) {

                cout << "---------------------------\n提示：\n\n二叉树尚未建立, 请执行选项1\n---------------------------" << endl;

                system("pause");

                system("cls");

                return;

        }

        cout << "\n输入Huffman code：";

        string source;

        cin >> source;

        cout << "\n解码结果为 ：";

        int ptr = this->maxLength;

        for (int i = 0; i < source.length(); i++) {

                if (this->huffTree[ptr].lch != 0 && this->huffTree[ptr].rch != 0) {

                        if (source[i] == '0') {

                                ptr = this->huffTree[ptr].lch;

                        } else if (source[i] == '1') {

                                ptr = this->huffTree[ptr].rch;

                        } else {

                                cout << "\n----------------------\n警告：\n\n遇到解码错误，即将返回\n----------------------";

                                system("pause");

                                system("cls");

                                return;

                        }

                } else {

                        cout << this->huffTree[ptr].data;

                        ptr = this->maxLength;

                        i--; // 回退

                }

        }

        cout << endl;

        system("pause");

        system("cls");

}

**5. 遇到的问题以及解决方案**

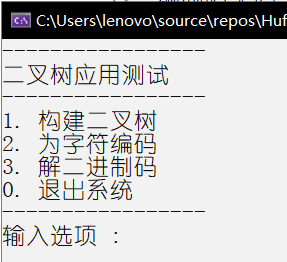
cin在遇到文件结束符的时候会停止读取缓存区的数据，此时应该使用cin.clear() 清空cin 的缓存区，否则容易出现读取数据失败的问题。

在写“筛选出权值最小的两个节点的下标”函数的时候，将进行比较的初始值赋值为了INT\_MIN，导致了每次程序运行的时候都只能筛选出来第一个值，后来通过查阅相关资料，改为了INT\_MAX。程序运行成功。

本程序的内容比较基础，设计的算法都可以在教科书中找到，没有遇到其他问题。

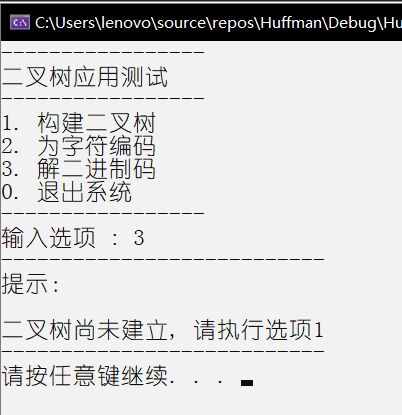
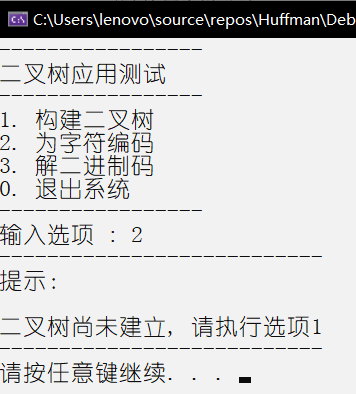
**6. 用户使用说明**

打开程序。



输入要执行的选项

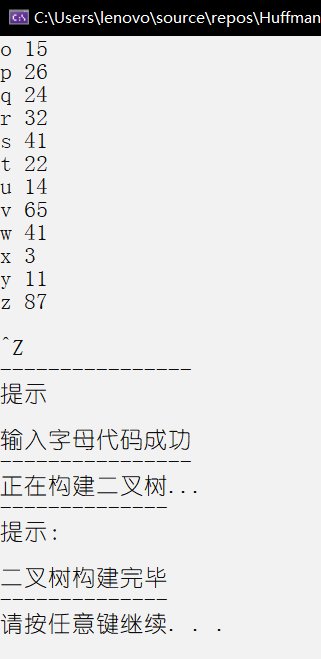
注意：在程序第一次运行的时候，直接执行选项2/3是不能用的，如下：



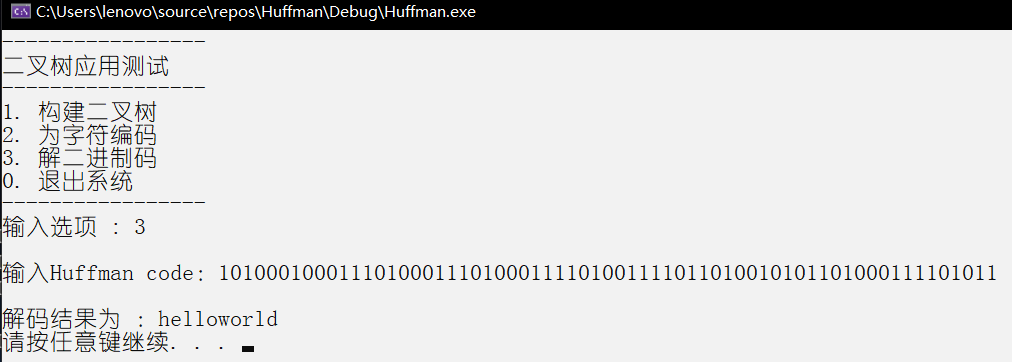
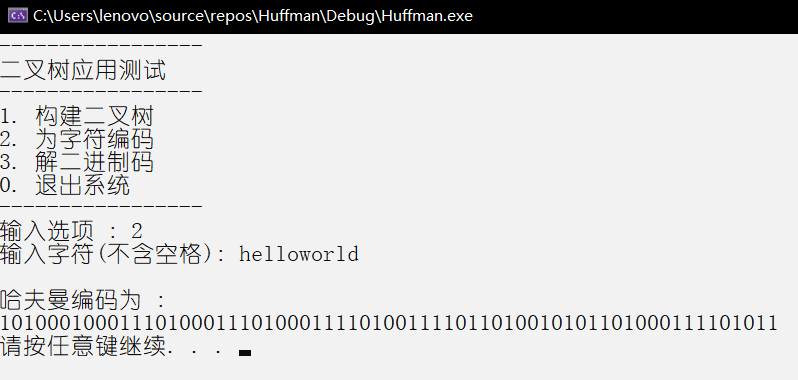
此时只能先执行选项1，进行二叉树的初始化 ：

输入字母和其权值，中间用空格隔开

输入完毕后输入文件结束符结束输入



然后测试选项2和3：

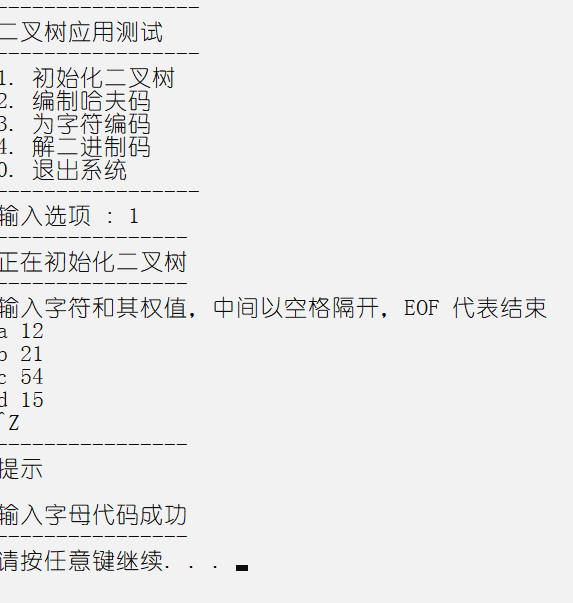


**7. 测试结果**

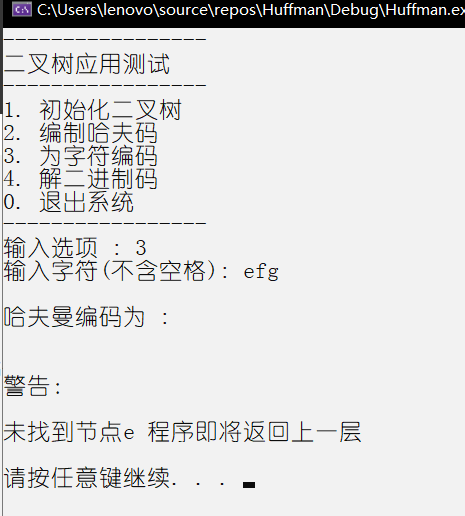
测试1.容错测试

数据:

|  |
| --- |
| a 12  b 21  c 54  d 15 |

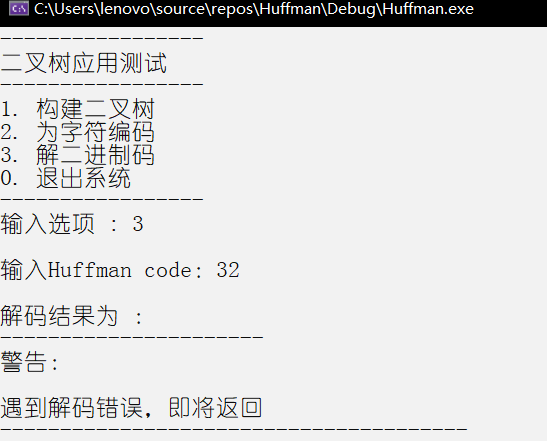


**输入没有编码的结点**

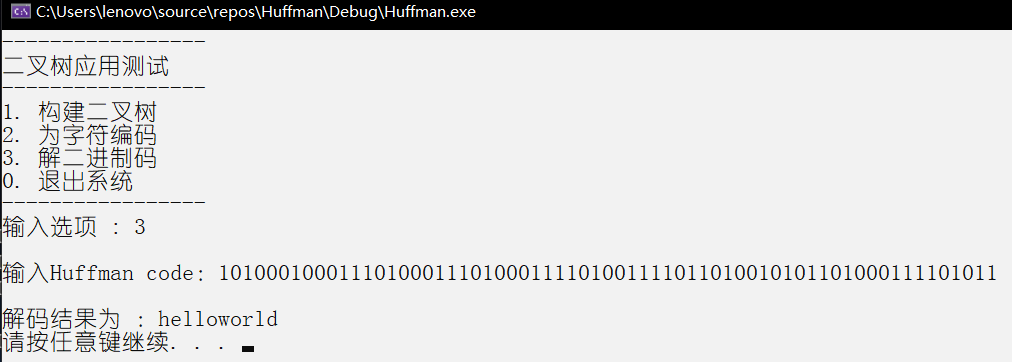
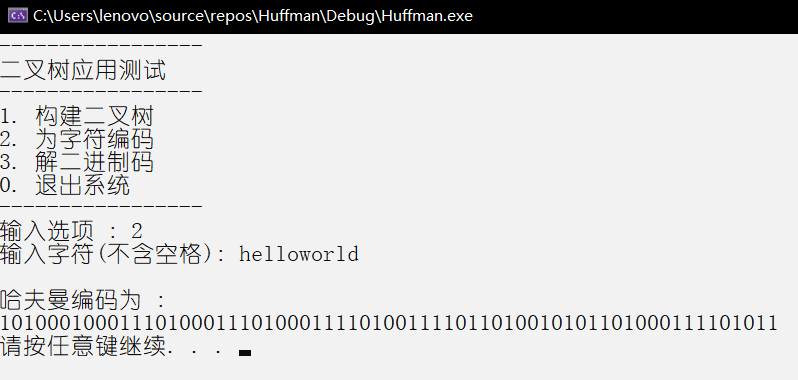


输入没有结点的编码

报错：



正常使用功能：



测试2（带有标点符号）

测试数据

|  |
| --- |
| i 1  l 2  o 3  v 4  y 5  u 8  , 9  e 10 |

