PA3 实验报告

——哈夫曼编码和译码

姓名：刘博洋 学号：2153538 日期：2022 年 11 月 10 日

1. 涉及数据结构和相关背景

树型结构是一种非常重要的非线性数据结构，其中树和二叉树是最为常用的数据结构。树和二叉树采用分支关系来定义层次，可以较为清晰的表示事物的组成结构或者逻辑结构，也是信息的重要组成形式之一，许多广泛被人们使用的算法例如决策树、哈夫曼树等等都是建立在基本的树和二叉树结构之上，MySQL 数据库生成索引的数据结构，也是应用了排序二叉树。对树进行分类，主要可以分为有序树和无序树，常见的有序树主要有：完全二叉树、满二叉树、二插搜素树、平衡二叉树等等。哈夫曼树（Huffman）又称为最优树，是一类带权路径长度最短的树，有着广泛的应用。

本次实验主要研究哈夫曼树在通信编码中的应用。在传送电文时，一方面希望对发出的电报进行加密，另一方面又希望电文总长尽可能短，所以可以对每个字符设计长度不等的编码，并且让电文中出现频率较多的字符采用尽可能短的编码，则传送电文的总长便可以减少。而由哈夫曼树的性质，所有数据节点都是叶子节点，即任意一个字符的编码都不是另一个字符编码的前缀，否则它必然在从根节点到叶子节点的路径上，且它不会是叶子节点。这种编码称为前缀编码，哈夫曼编码就是一种前缀编码。

树的抽象数据类型定义：

ADT BiTree{

数据对象：D={ai |1≤i≤n，n≥0，ai 属 ElemType 类型}

数据关系：R={<ai ,aj>|ai ,aj∈D,1≤i≤n, 1≤j≤n,其中每个元素

只有一个直接前驱，可以有至多两个直接后继，有且仅

有一个元素没有直接前驱}

基本运算：

(1)Binarylnit()：创建一棵空二叉树。

(2) BinaryEmpty(T)：判断一棵二叉树 T 是否为空。

(3) Root(T) ：返回二叉树 T 的根结点标号。

(4) MakeTree(x,T,L,R) ：以 x 为根结点元素，分别以 L 和 R 为左、右子树构建一棵新的二叉树 T。

(5) Break Tree(T,L,R) ：函数 MakeTree 的逆运算将二叉树 T 拆分为根结点元素，左子树 L 和右子树尺等三部分。

(6) PreOrder(visit,t) ：前序遍历二叉树。

(7) InOrder(visit,t) ：中序遍历二叉树。

(8) PostOrder(visit,t) ：后序遍历二叉树。

(9) PreOut(T) ：二叉树前序列表。

(10)InOut(T) ：二叉树中序列表。

(11)PostOut(T) ：二叉树后序列表。

(12)Delete(t) ：删除二叉树。

(13)Height(f) ：二叉树的高度。

(14)Size(t) ：二叉树的结点数。

}ADT BiTree

2. 实验内容

2.1 问题描述

实现对 ASCII 字符文本进行 Huffman 压缩，并且能够进行解压。将给定的文本文件使用哈夫曼树进行压缩，并解压。

2.2 基本要求

能将文本文件压缩、打印压缩后编码、解压压缩后的文件。对实际使用的具体数据结构除必须使用二叉树外不做要求。

用一个二叉树表示哈夫曼树，因为 ASCII 表一共只有 127 个字符，可以直接使用数组来构造 Huffman 树。

(1) 程序要添加适当的注释，程序的书写要采用缩进格式 。

(2) 程序要具在一定的健壮性，即当输入数据非法时，程序也能适当地做出反应，如插入删除时指定的位置不对等等。

(3) 程序要做到界面友好，在程序运行时用户可以根据相应的提示信息进行操作。

(4) 根据实验报告模板详细书写实验报告，在实验报告中给出主要算法的复杂度分析。

2.3 数据结构设计

2.3.1 哈夫曼编码的原理

哈夫曼编码是一种可变长编码方式，比起定长编码的 ASCII 编码来说，哈夫曼编码能节省很多的空间，因为每一个字符出现的频率不是一致的，它是一种用于无损数据压缩的熵编码算法，通常用于压缩重复率比较高的字符数据。

例如：对于字符串 ABCDEFG,

如果采用等长编码，如下表

字符	等长编码值
A	000
B	001
C	010
D	011
E	100
F	101
G	110

表 1

等长编码的缺点是浪费空间，一共 7 个数，却必须每个数都要用三位二进制表示，并且等长编码不是前缀编码，所以在编码的同时对每个字符之间还要进行分隔，否则会造成误读。

如果联想到哈夫曼树的特点，每个节点都有权值，并且整棵树能够实现 WPL 最小，则可以采用哈夫曼编码，如下图对 ABCDEFGH 创建一颗哈夫曼树，对于每一个节点，左孩子上的路径标记为 0，右孩子上的路径标记为 1。每颗字符节点都是叶子节点，从根节点到叶子节点连接路径的字符编码就是该字符的哈夫曼编码。

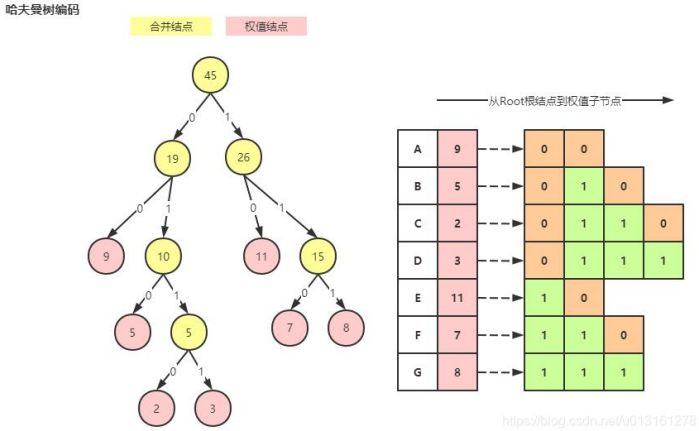

图 1

字符	不等长编码值
A	00
B	010
C	0110
D	0111
E	10
F	110
G	111

表 2

而由哈夫曼树的性质，所有数据节点都是叶子节点，即任意一个字符的编码都不是另一个字符编码的前缀，否则它必然在从根节点到叶子节点的路径上，且它不会是叶子节点。所以哈夫曼编码是前缀编码，可以节省许多分隔符的空间。

哈夫曼树的建立有一种一般规律的算法，即哈夫曼算法，叙述如下：

（1）根据给定或计算的 n 个权值{w1,w2,w3....}构成 n 颗二叉树，组成森林，每颗二叉树 Ti 中只有一个带权为 wi 的根节点，左右子树均为空

（2）在森林中选取两棵树根节点权值最小的树作为左右子树构造一颗新的二叉树，并且使新二叉树的根节点权值为左右子树权值之和。

（3）在森林中删除这两棵树，并且把新得到的二叉树加入森林。

（4）重复（2）和（3）直至森林中只剩下一颗二叉树，这棵树就是哈夫曼树

2.3.2 本题思路

想到得到电文总长最短的二进制前缀编码，便可以把每种字符在电文中出现的频率或者概率作为其权值，将权值最大的字符赋以最短的编码长度，设计一颗 Huffman 树。

首先，建立一个字符数组 s 储存输入的字符串，建立一个储存字符权值的数组 w[128]，由于 128 个字符不一定每个都会出现，所以还需要一个 info 数组按出现的顺序储存出现的字符。根据键盘输入或者 txt 文件的文本内容遍历得到 0-127ASCII 码在电文中出现的频率存入权值数组 w[]中。在遍历的同时也可以求出出现的不同字符数 n，作为后面使用的工具。

其次，遍历权值数组，得到权值最小的两个元素准备构建节点。设置两个全局变量 MostMin 和 SecondMin，动态指向权值最小的两个元素下标。

在建立 HuffmanTree 过程中，要注意到有 n 个叶子节点的 Huffmantree 有 2n-1 个节点，所以根据之前计算的 n 为数组分配空间。然后对数组进行初始化，使 1—2n-1 的每个节点 lchild，rchild，parent 都为 0，使 1—n 的初始代表每个字符的节点权值为权值数组中对应的权值。然后开启循环，先找到当前权值最小的两个节点作为左右子树，从第 n+1 个元素开始，每次两节点的 parent 指向该元素，该元素权值为二者之和，左右子树分为二者下标。

重复这个过程 n-1 次即可得到哈夫曼树。

此时每个字符的编码都被储存在树种，现在需要遍历树进行解码，我们采用从叶子节点到根节点逆向求编码的方式，对从 1—n 代表字符的叶子节点，每个节点回溯寻找其 parent，判断自己是左子树还是右子树，如果是左子树那么编码最后一位为 0，否则为 1，依次类推直到回溯到根节点。动态建立二维数组 Huffmancode，每一维储存一个字符的哈夫曼编码，将每一次回溯到的结果存入二维数组中，由于存储的顺序也是文本中字符出现的顺序，所以想要输出一段文本的哈夫曼编码，只需要对每一个字符寻找其哈夫曼编码并输出即可。

2.4 代码实现

```
typedef int Status;
typedef struct HTNode {
    char data;
    int weight;
    int lchild, rchild, parent;
}HTNode, * HuffTree;//节点类型定义
HuffTree HT;
char s[100000] = {};//输入的文本
int w[128] = {};//权值数组
int n = 0;
char info[128] = {};//出现的字符
typedef char** HuffmanCode;//二维数组储存每个字符的哈夫曼编码
HuffmanCode HC;
int MostMin = -1;
int SecondMin = -1;
int input(char s[])
{
    int i = 0;
    while (1)
```

```
    {
        /*getchar();*/
        char ch;
        ch = getchar();
        if (ch == '\n')
            break;
        s[i] = ch;
        i++;
    }
    return i;//得到文本的长度
}
void getweight(int l)
{
    int flag = 0;
    for (int i = 0;i < l;i++)
    {
        flag = 0;
        for (int j = 0;j < i;j++)
        {
            if (s[i] == s[j])
            {
                flag = 1;
                w[j]++;//得到权值
                break;
            }
        }
        if (flag == 0)
        {
            w[n]++;
            info[n] = s[i];//如果没有出现过，加入 info
            n++;
        }


    }
}
void getmin(HuffTree T, int k)//得到两个权值最小的节点
{
    for (int i = 0; i <= k; i++) {
        if (T[i].weight != 0 && T[i].parent == 0)
        {
            if (MostMin == -1) {
                MostMin = i;
            }
```

```
            else if (SecondMin == -1) {
                SecondMin = i;
                if (T[MostMin].weight > T[SecondMin].weight) {
                    swap(SecondMin, MostMin);
                }
            }
            else {
                if (T[i].weight < T[MostMin].weight)
                {
                    SecondMin = MostMin;
                    MostMin = i;
                }
                else if (T[i].weight < T[SecondMin].weight) {
                    SecondMin = i;
                }
            }
        }
    }
}
void CreateHuffTree(HuffTree& HT, HuffmanCode& HC, int* w, int n, int
l)
{
    if (n <= 1)
        return;
    int m = 2 * n - 1;
    HT = (HuffTree)malloc((m+1) * sizeof(HTNode));
    int i;
    HuffTree p;
    for (p = HT, i = 0; i < n; ++i, ++p, ++w)
        //初始化 1—n 的元素（字符）
    {
        p->weight = *w; p->parent = 0; p->lchild = 0;p->rchild =
0;p->data = info[i];
    }
    for (; i <= m;++i, ++p)//初始化 n—m 的元素（空）
    {
        p->weight = 0; p->parent = 0; p->lchild = 0; p->rchild = 0;
    }
    for (i = n; i < m;++i)
    {
        getmin(HT, i - 1);
        HT[MostMin].parent = i; HT[SecondMin].parent = i;
        HT[i].lchild = MostMin; HT[i].rchild = SecondMin;
        HT[i].weight = HT[MostMin].weight + HT[SecondMin].weight;//将
```

新节点依次放入 n—m 个位置中

```
        MostMin = -1;
        SecondMin = -1;
    }
    HC = (HuffmanCode)malloc((n+1) * sizeof(char *));
    char* cd = (char*)malloc(n * sizeof(char));
    cd[n - 1] = '\0';
    cout << "各字符的编码为：" << endl;
    for (i = 0;i < n;++i)
    {
        int start = n - 1;
        for (int c = i, f = HT[c].parent; f != 0; c = f, f = HT[f].parent)//
从叶子向根逆向求编码
            if (HT[f].lchild == c) cd[--start] = '0';
            else cd[--start] = '1';
        HC[i] = (char*)malloc((n - start) * sizeof(char));
        strcpy_s(HC[i], sizeof(&cd[start]), &cd[start]);
        cout << info[i] << ":" << HC[i] << endl;
    }
    cout << endl;
    cout << "转换得到的 Huffman 编码为：" << endl;
    for (int i = 0;i < l;i++)
    {
        int j = 0;
        while (1)
        {
            if (s[i] == info[j])
            {
                cout << HC[j];
                break;
            }
            j++;
        }
    }
    free(cd);
}
void decode_Huff(string s, HuffTree &HT, int n) {//解码
    int i = 0, j = 0, p = 2 * n - 2;
    char a[1000] = {};
    while (s[i] != '\0') {
        if (s[i] == '0') {
            p = HT[p].lchild;
        }
        else if (s[i] == '1') {
```

```
            p = HT[p].rchild;
        }                   //判断 p 是否为叶子结点，是则将对应数据储存进
a
        if (HT[p].lchild == 0 && HT[p].rchild == 0) {
            a[j] = HT[p].data; j++;
            p = 2 * n - 2;
        }
        else if (s[i + 1] == '\0' && HT[p].lchild != 0 && HT[p].rchild !=
0) {
            cout << "译码失败！" << endl;
            return;
        }
        i++;
    }
    int k = 0;
    while (a[k] != 0) cout << a[k++];
    cout << endl;
}

int main()
{
    cout << "********************" << endl;
    cout << "1.编码" << endl;
    cout << "2.译码" << endl;
    cout << "0.退出" << endl;
    cout << "********************" << endl;
    while (1)
    {
        cout << "请输入执行的功能编号：" << endl;
        int m;
        cin >> m;
        getchar();
        switch (m)
        {
        case 1: {
              cout << "请输入需要转化的文本:" << endl;
              int l = input(s);
              getweight(l);
              cout << endl;
              CreateHuffTree(HT, HC, w, n, l);
              cout << endl;

        }
              break;
```

```
        case 2: {string s;
            cout << "请输入需要解压的 Huffman 编码" << endl;
            cin >> s;
            decode_Huff(s, HT, n);
            cout << endl;
            break;}
        case 0:
            return 0;
            break;
        }
    }
    return 0;
}
```

2.5 功能展示

初始化菜单界面，键盘输出操作数来进行操作

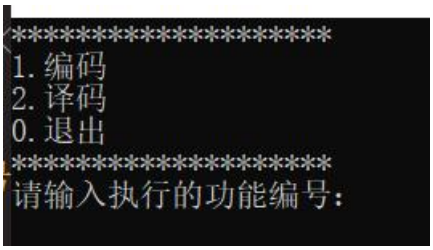

图 2 简单的菜单界面

选择 1，对某一文本进行编码，此处输入的是著名演讲“I have a dream”中的一段话

编码后首先按字符出现顺序输出每个字符的不等长编码，然后一起输出转换后的 Huffman 编码文本

图 3

再将转换得到的编码输入，可以得到原来的文本

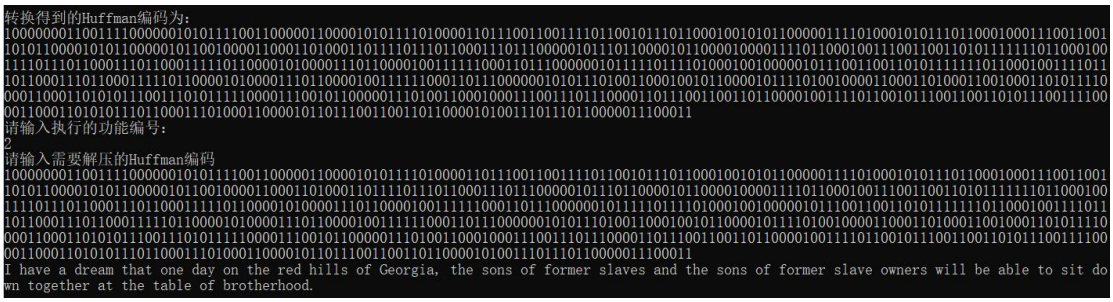

图 4

再输入 0 关闭窗口，结束进程。

图 5

3.实验总结

3.1 程序分析

3.1.1 效率和健壮性分析

本程序通过键盘输入文本来生成 huffman 数，再通过 huffman 树来确定每个字母编码并输出代表文本的编码，对于文本的输入默认输入的全部是常用的 128 个 ascii 码的内容，不会出错，然后将这些字符全部存入数组并计算权值，接着根据权值分配来构造 huffman 最优树，再遍历叶子节点得到路径和对应编码。解压操作则从头遍历判断是否是叶子节点，如果是则存进结果数组，否则继续遍历。其中建立 huffman 树获得 huffman 编码和 decodehuffman 编码的时间复杂度均为 O(n)，较为高效。程序的健壮性主要体现在通过菜单页面进行功能选择，不会造成输入错误，在各处分配空间时都严格匹配需要的内存，并且在译码错误时会弹出提示并结束程序，防止溢出。总体来说，程序效率较高，也有一定的健壮性，对于用户比较友好。

3.1.2 调试与改进

本程序的调试问题主要出在通过 huffman 树读取 huffman 编码的过程中，内存分配的问题，要事先计算好一一共需要多少内存，刚开始的时候由于少分配了一个字符的内存导致运行报错，同时，在输入的过程中我们只是知道了字符的输入个数和顺序，还需要一个 info 数组来根据顺序存放已经出现的字符。而在对 huffman 树前 n 个元素进行初始化时，其 data 域应该等于对应 info 数组的值，而不是输入内容 s 数组的值，否则会在后面译码的时候造成重复译码或者漏译码的问题。

本程序主要的改进点在于能否用文件方式进行读写，因为在实际应用中，不可能手动向控制台输入很长的文本，通常是通过文件形式进行传输。所以考虑在菜单栏中加一项“3.选择编码译码模式(默认为键盘输入)”可以更改读写模式，如果选择文件读写模式，则需要在文件夹中首先自己创建一个 input.txt，存放需要编码的文本，然后在程序中定义:

```
    FILE* stream1,*stream2;
stream1 = fopen("input.txt", "r");//只读方式
stream2 = fopen("output.txt", "w+");//创建一个新 txt，既可以读又可以写
        while (1)
        {
            /*getchar();*/
            char ch;
            ch = getc(stream1);
            if (ch == EOF)
                break;
            s[i] = ch;
            i++;
        }//读入
            fclose(stream1);
    fprintf(stream2, "%c:%s", info[i], HC[i]);//输出每个字符的编码
    fprintf(stream2, "%s" ,HC[i]);
    fclose(stream2);
```

这种功能同样也可以通过输入输出重定向和管道运算符来实现，也同样比较方便可以将输入输出都以文件形式呈现。

3.2 总结和收获

本次作业是实现 huffman 编码和解码的操作，这些操作和算法课本和 ppt 上都有，但是只有我们通过自己去亲身实现它才能更加深刻的理解。对于树这一部分的内容，我们经过几周的学习其实只是触及表层，树还有更多更加广泛的应用空间和算法领域，我们在后期的学习中应该不断主动去探索和实现。在这次作业中，我重新复习巩固了文件读写的有关内容，并且逐渐形成一直工程的思维，即要把一个程序做的精益求精，既要有对于用户的友好和方便性，也要有健壮性和高效率，一直在改进程序和改正错误中提升自己。其实很感谢这种报告作业，让我们在自我实现程序的过程中也能每次都得到新的收获并记录下来。