《数据结构》
课程设计报告

课程名称：	《数据结构》课程设计
课程设计题目：	哈夫曼树编码
姓　　名：	金宏睿
院　　系：	计算机学院
专　　业：	计算机科学与技术
班　　级：	19052318
学　　号：	19051814
指导教师：	肖周芳

2020 年 11 月 27 日

一、需求分析

功能需求：

设计并实现一个哈夫曼码的编/译码系统，系统 功能包括：
（1）I：初始化（Initialization）。从终端读入字符集大 小 n，以及 n 个字符和 n 个权值，建立哈夫曼树，并 将它存于文件 hfmTree 中；
（2）E：编码（Encoding）。利用以建好的哈夫曼树（如 不在内存，则从文件 hfmTree 中读入），对文件 ToBeTran 中的正文进行编码，然后将结果存入文件 CodeFile 中；
（3）D：译码（Decoding）。利用已建好的哈夫曼树将 文件 CodeFile 中的代码进行译码，结果存入文件 TextFile 中；
（4）P：印代码文件（Print）。将文件 CodeFile 以紧 凑格式显示在终端上，每行 50 个代码。同时将此字符 形式的编码文件写入文件 CodePrin 中；
（5）T：印哈夫曼树（Tree printing）。将已在内存中 的哈夫曼树以直观的方式（树或凹入表形式）显示在 终端上，同时将此字符形式的哈夫曼树写入文件 TreePrint 中。

界面需求：

程序操作主页，介绍每个各功能模块的作用，并提示用户进行选择，部分功能模块会提示用户对功能进行特定选择。

对于用户的每一个操作有具体提示和反馈

二、概要设计

模块层次结构设计

void HuffmanHomepag () //所有操作的主入口，函数内可触发所有功能

接口设计

```
//删除哈夫曼树结点
void HuffmanDel(HFTree n);

//对哈夫曼树先序遍历，并对结点进行可选的函数操作
void HuffmanTra(HFTree t, func fun);

//读取用户输入数据并保存，用于创建哈夫曼树
int datainput();

//读取 ToBeTran.txt 文件，并返回保存了文件内容的字符串
char *ToBeTranLoad();

//读取 CodeFile.txt 文件并返回保存了文件内容的字符串
char *CodeFileLoad();

//返回结点数组中权重最小的结点的位置，参数 con 为选择中排除的的位置
int HuffmanMinVal();

//根据输入的数据创建哈夫曼树
void HuffmanCreate();

//根据哈夫曼树建立哈夫曼编码，并存入 str 字符串数组
//参数分别为哈夫曼树和用于创建当前字符哈夫曼编码的字符串
void HuffmanCodeBuild(HFTree t, char *s);

//输出哈夫曼编码的关键信息到 hfmTree.txt 中
void HuffmanFileOut();

//输出哈夫曼编码的关键信息到 hfmTree.txt 中
void HuffmanFileOut();

//导入 hfmTree.txt 中的信息
bool HuffmanLoad();

//导入 hfmTree.txt 中的信息
bool HuffmanLoad();
```

```
//对传入的字符串进行编码
void HuffmanEncode(char *s);

//调用函数，使用导入的信息构造哈夫曼树
void HuffmanBuild();

//递归构造哈夫曼树
void HuffmanBdTra(HFTree r, char *BDstr, char dt);

//对传入的哈夫曼码进行解码
void HuffmanDecode(char *s);

//对传入的哈夫曼码进行解码
void HuffmanDecode(char *s);

//递归先序遍历,将末端结点的 data 输入到输出流
int HuffmanDcTra(HFTree t, char *s, int num, ostream &os);

//打印哈夫曼树的凹入吧并存入文件
void HuffmanDisplay();

//先序遍历对于每个字符进行凹入表构造
void HuffmanTable(HFTree t, int num, int flag, ostream &os);
```

数据结构设计

```
string str;
int wei[1000]; //存储各结点权重
char c[1000]; //保存内存中用于编码的字符
int vis[1000];//在相关操作中用于标记
char *code[1000];//用于保存每个字符对应的编码
int HuffmanLen;//当前字符的个数

//哈夫曼树的结点
typedef struct HFnode
{
    HFnode()
    {
        parent = NULL;
        lchild = NULL;
        rchild = NULL;
        data = -1;
    }
    char data;
```

```
    int we;
    HFnode *parent;
    struct HFnode *lchild;
    struct HFnode *rchild;
} HFnode, *HFTree;
```

三、详细设计

```
#include <bits/stdc++.h>
using namespace std;

using ElementType = char;

string str;
int wei[1000];       //存储各结点权重
char c[1000];        //保存内存中用于编码的字符
int vis[1000];       //在相关操作中用于标记
char *code[1000]; //用于保存每个字符对应的编码
int HuffmanLen;      //当前字符的个数

//哈夫曼树的结点
typedef struct HFnode
{
    HFnode()
    {
        parent = NULL;
        lchild = NULL;
        rchild = NULL;
        data = -1;
    }
    ElementType data;
    int we;
    HFnode *parent;
    struct HFnode *lchild;
    struct HFnode *rchild;
} HFnode, *HFTree;
typedef void (*func)(HFTree);

HFTree HuffmanNode[1000]; //保存哈夫曼树的各结点
HFTree HuffmanTree;             //内存总保存的总哈夫曼树

//删除哈夫曼树结点
void HuffmanDel(HFTree n);
```

```
//对哈夫曼树先序遍历，并对结点进行可选的函数操作
void HuffmanTra(HFTree t, func fun);

//读取用户输入数据并保存，用于创建哈夫曼树
int datainput();

//读取 ToBeTran.txt 文件，并返回保存了文件内容的字符串
char *ToBeTranLoad();

//读取 CodeFile.txt 文件并返回保存了文件内容的字符串
char *CodeFileLoad();

//返回结点数组中权重最小的结点的位置，参数 con 为选择中排除的的位置
int HuffmanMinVal();

//根据输入的数据创建哈夫曼树
void HuffmanCreate();

//根据哈夫曼树建立哈夫曼编码，并存入 str 字符串数组
//参数分别为哈夫曼树和用于创建当前字符哈夫曼编码的字符串
void HuffmanCodeBuild(HFTree t, char *s);

//输出哈夫曼编码的关键信息到 hfmTree.txt 中
void HuffmanFileOut();

//输出哈夫曼编码的关键信息到 hfmTree.txt 中
void HuffmanFileOut();

//导入 hfmTree.txt 中的信息
bool HuffmanLoad();

//导入 hfmTree.txt 中的信息
bool HuffmanLoad();

//对传入的字符串进行编码
void HuffmanEncode(char *s);

//调用函数，使用导入的信息构造哈夫曼树
void HuffmanBuild();

//递归构造哈夫曼树
void HuffmanBdTra(HFTree r, char *BDstr, char dt);
```

```
//对传入的哈夫曼码进行解码
void HuffmanDecode(char *s);

//对传入的哈夫曼码进行解码
void HuffmanDecode(char *s);

//递归先序遍历,将末端结点的 data 输入到输出流
int HuffmanDcTra(HFTree t, char *s, int num, ostream &os);

//打印哈夫曼树的凹入吧并存入文件
void HuffmanDisplay();

//先序遍历对于每个字符进行凹入表构造
void HuffmanTable(HFTree t, int num, int flag, ostream &os);

//打印字符极其对应编码到终端
void huaffmanShow();

//获取进行编码的字符串
char *GetEncodeStr();

//获取进行解码的字符串
char* GetDecodeStr();

//程序操作主页，用于提示用户输入操作方式解释各选项的作用
void HuffmanHomepage()
{
    int ck = 1;              //用于在某些步骤检测是否成功加载文件中的哈夫曼树
    char et[100] = {0}; //用于读取用户的选项及进行选项判断
    char *strenc;

    //无限循环，使程序能持续工作
    while (1)
    {
        memset(et, 0, sizeof(et)); //每次判断前清空用于判断的字符数组
        cout << "------------------------------------------------------------" << endl;
        cout << "-please enter the num to choose the faction :                    -" << endl;
        cout << "-       I - Initializing Huffman tree.                           -" << endl;
        cout << "-       E - Encoding the ToBeTran.txt.                           -" << endl;
        cout << "-       D - Decoding the CodeFile.txt.                           -" << endl;
        cout << "-       P - Printing the CodeFile.txt in the terminal.           -" << endl;
        cout << "-       T - Printing and saving the HuffmanTree table.           -" << endl;
        cout << "------------------------------------------------------------" << endl;
        cout << endl;
```

```
        cout<<"enter your choice : ";
        cin>>et;

        cout << endl;

        if (et[0] == 'I') //选择了对哈夫曼树进行初始化和数据输入
        {
            HuffmanTra(HuffmanTree, HuffmanDel); //删除内存中原有哈夫曼树
            datainput();                          //对新哈夫曼树输入数据
            HuffmanCreate();                      //创建哈夫曼树
            HuffmanCodeBuild(HuffmanTree,  "");     //对于创建出来的树进行哈夫曼编码构
造
            HuffmanFileOut();                     //将哈夫曼编码及对应字符存入文件
            huaffmanShow();                       //打印字符极其对应编码到终端
        }
        else if (et[0] == 'E') //对文件中的正文进行编码
        {
            if (HuffmanLen == 0) //如果树不存在于内存就从文件中加载
            {
                ck = HuffmanLoad(); //加载文件中的哈夫曼树
                if (!ck)                    //文件加载失败就中止本次操作
                    continue;
                cout << "a new huffmanTree will be created" << endl;
                HuffmanBuild();     //根据文件中读取的哈夫曼树的信息构造哈夫曼树
                HuffmanFileOut(); //将哈夫曼树及对应字符存入文件
            }
            strenc = GetEncodeStr();
            HuffmanEncode(strenc); //加载 ToBeTranc 中的正文并进行编码
            cout<<endl;
        }
        else if (et[0] == 'D') //对文件中的哈夫曼码进行解码
        {
            if (HuffmanLen == 0) //如果树不存在于内存就从文件中加载
            {
                ck = HuffmanLoad(); //加载文件中的哈夫曼树
                if (!ck)                    //文件加载失败就中止本次操作
                    continue;
                cout << "a new huffmanTree will be created" << endl;
                HuffmanBuild();     //根据文件中读取的哈夫曼树的信息构造哈夫曼树
                HuffmanFileOut(); //将哈夫曼树及对应字符存入文件
            }
            strenc = GetDecodeStr();
            HuffmanDecode(strenc); //加载 CodeFile 中的哈夫曼码进行解码
            cout << " CodeFile.txt decode completed" << endl;
```

```
            }
            else if (et[0] == 'P') //将 CodeFile 中的哈夫曼码输入到终端和存入文件
            {
                cout << CodeFileLoad() << endl;
                fstream f("CodePrin.txt", ios::out | ios_base::trunc);
                if (!f)
                {
                    cout << "CodePrin.txt open error" << endl;
                    break;
                }
                f << CodeFileLoad() << endl;
                f.close();
            }
            else if (et[0] == 'T') //将哈夫曼树的凹入表打印到终端并存入文件
            {
                HuffmanDisplay();
            }
            else //输入其他则再次进入程序
            {
                HuffmanHomepage();
            }
        }
}

int main()
{
    HuffmanLen = 0;
    HuffmanHomepage();
    return 0;
}

// 5
// a(1) b(2) c(3) d(4) e(5)

//删除哈夫曼树结点
void HuffmanDel(HFTree n)
{
    if (n)
    {
        if (n->parent->lchild == n)
            n->parent->lchild == NULL;
        if (n->parent->rchild == n)
            n->parent->rchild == NULL;
        delete n;
```

```
    }
}

//对哈夫曼树先序遍历，并对结点进行可选的函数操作
void HuffmanTra(HFTree t, func fun)
{
    if (t)
    {
        fun(t);
        if (t->lchild)
            HuffmanTra(t->lchild, fun);
        if (t->rchild)
            HuffmanTra(t->rchild, fun);
    }
}

//读取用户输入数据并保存，用于创建哈夫曼树
int datainput()
{
    cout << "Please enter the number of characters : ";
    cin >> HuffmanLen;
    if (HuffmanLen < 1)
        return HuffmanLen;
    for (int i = 0; i < HuffmanLen; i++)
    {
        cout << "Please enter the value and weight of character " << i + 1 << " : ";
        cin >> c[i] >> wei[i];        //将字符和对应权重存入数组
        HFnode *node = new HFnode; //创建结点空间
        node->data = c[i];              //将对应字符存入结点的 data
        node->we = wei[i];               //将字符对应权重存结点的 wei
        HuffmanNode[i] = node;         //将结点保存到哈夫曼结点数组
    }
    c[HuffmanLen] = 0; //将字符数组的末尾设置为 0
    return HuffmanLen; //返回哈夫曼树的长度
}
//获取进行解码的字符串
char* GetDecodeStr()
{
   while (1)
    {
        cout << "1. using CodeFile file" << endl;
        cout << "2. using your own string " << endl;
        cout << "3. back to previous" << endl;
        cout << endl;
```

```
            cout << "enter yout choice : ";
            int ck;
            cin >> ck;
            if (ck == 1)
            {
                char * str = CodeFileLoad();
                cout<< " string to be decode is : "<<str<<endl;
                cout<<endl;
                cout<<endl;
                return str;
            }
            else if (ck == 2)
            {
                cout << "please input your string :";
                char *str = new char[1000];
                cin >> str;
                return str;
            }
            else if (ck == 3)
            {
                HuffmanHomepage();
            }
        }
}

//获取进行编码的字符串
char *GetEncodeStr()
{
    while (1)
    {
        cout << "1. using ToBeTran file" << endl;
        cout << "2. using your own string " << endl;
        cout << "3. back to previous" << endl;
        cout << endl;
        cout << "enter yout choice : ";
        int ck;
        cin >> ck;
        if (ck == 1)
        {
            char * str = ToBeTranLoad();
            cout<< " string to be encode is : "<<str<<endl;
            cout<<endl;
            cout<<endl;
            return str;
```

```
            }
            else if (ck == 2)
            {
                cout << "please input your string :";
                char *str = new char[1000];
                cin >> str;
                return str;
            }
            else if (ck == 3)
            {
                HuffmanHomepage();
            }
        }
}

//读取 ToBeTran.txt 文件，并返回保存了文件内容的字符串
char *ToBeTranLoad()
{
    FILE *f;
    f = fopen("ToBeTran.txt", "r");
    if (f == NULL)
    {
        cout << "open file failed" << endl;
        return NULL;
    }
    char *TranStr = new char[1000]; //开辟字符数组空间保存文件内容
    memset(TranStr, 0, sizeof(TranStr));
    fscanf(f, "%s", TranStr);
    fclose(f);
    return TranStr;
}

//读取 CodeFile.txt 文件并返回保存了文件内容的字符串
char *CodeFileLoad()
{
    FILE *f;
    f = fopen("CodeFile.txt", "r");
    if (f == NULL)
    {
        cout << "open file failed" << endl;
        return NULL;
    }
    char *CodeStr = new char[1000]; //开辟字符数组空间保存文件内容
    memset(CodeStr, 0, sizeof(CodeStr));
```

```
    fscanf(f, "%s", CodeStr);
    fclose(f);
    return CodeStr;
}

//返回结点数组中权重最小的结点的位置，参数 con 为选择中排除的的位置
int HuffmanMinVal()
{
    if (HuffmanLen)
    {
        int minwe;
        int pos = -1;
        int flag = 0;
        for (int i = 0; i < HuffmanLen; i++)
        {
            if (vis[i]) //标记过的位置不参与选择
                continue;
            //第一次进入循环时，保存到最小值设置为第一个
            if (!flag)
            {
                minwe = HuffmanNode[i]->we;
                pos = i;
                flag = 1;
            }
            //比较权重，保存最小的权重和相应位置
            if (HuffmanNode[i]->we < minwe)
            {
                minwe = HuffmanNode[i]->we;
                pos = i;
            }
        }
        return pos; //返回最小的点坐标
    }
    return -1;
}

//根据输入的数据创建哈夫曼树
void HuffmanCreate()
{
    memset(vis, 0, sizeof(vis));
    HFnode *tp;
    int count = 1;
    int last, seclast = -1; //分别用于保存当前权重最小和第二小的结点位置
    while (count < HuffmanLen)
```

```
    {
        last = HuffmanMinVal();                                    //保存权重最小的
结点
        vis[last] = 1;                                             //标记此位置不参
与之后的选择
        seclast = HuffmanMinVal();                                 //保存权重第二小
的结点
        HFTree ht = new HFnode;                                    //开辟空间作为
新的树结点
        ht->lchild = HuffmanNode[seclast];                         //左子树为第二小的
结点
        ht->rchild = HuffmanNode[last];                            //右子树为最下的
结点
        ht->we = HuffmanNode[last]->we + HuffmanNode[seclast]->we; //新结点的权重为左
右子树权重之和
        HuffmanNode[seclast] = ht;                                 //保存新的结点在
第二小的结点的位置
        count++;
    }
    //如果没有参加选择，哈夫曼树就只有结点数组的首结点
    if (seclast == -1)
    {
        HuffmanTree = HuffmanNode[0];
    }
    //否则就返回当前保存的最新的树
    else
    {
        HuffmanTree = HuffmanNode[seclast];
    }
}

//根据哈夫曼树建立哈夫曼编码，并存入 str 字符串数组
//参数分别为哈夫曼树和用于创建当前字符哈夫曼编码的字符串
void HuffmanCodeBuild(HFTree t, char *s)
{
    if (t)
    {
        if (t->lchild)
        {
            //如果左子树存在，在当前字符串之前加一个 0,并参与下一次递归
            char *str = new char[strlen(s) + 2];
            strcpy(str, s);
            str[strlen(s)] = '0';
            str[strlen(s) + 1] = 0;
```

```
            HuffmanCodeBuild(t->lchild, str);
        }
        if (t->rchild)
        {
            //如果左子树存在，在当前字符串之前加一个 0,并参与下一次递归
            char *str = new char[strlen(s) + 2];
            strcpy(str, s);
            str[strlen(s)] = '1';
            str[strlen(s) + 1] = 0;
            HuffmanCodeBuild(t->rchild, str);
        }
        //循环到末尾结点则为保存了字符的无孩子结点
        if (t->lchild == NULL && t->rchild == NULL)
        {
            for (int i = 0; i < HuffmanLen; i++)
            {
                if (t->data == c[i])
                {
                    code[i] = s;
                    break;
                }
            }
        }
    }
}

//输出哈夫曼编码的关键信息到 hfmTree.txt 中
void HuffmanFileOut()
{
    cout << endl;
    FILE *f;
    f = fopen("hfmTree.txt", "wb");
    if (f == NULL)
    {
        cout << "open file failed" << endl;
        return;
    }
    fprintf(f, "%d\n", HuffmanLen);
    //通过循环将信息输入文件
    for (int i = 0; i < HuffmanLen; i++)
    {
        fprintf(f, "%c %s\n", c[i], code[i]);
    }
    fclose(f);
```

```
	cout << "Huffman Save Completed " << endl;
	cout << endl;
	cout << endl;
}

//打印字母以及对应编码
void huaffmanShow()
{
	for (int i = 0; i < HuffmanLen; i++)
	{
		printf("%c ---- %s\n", c[i], code[i]);
	}
	cout << endl;
	cout << endl;
}

//导入 hfmTree.txt 中的信息
bool HuffmanLoad()
{
	memset(c, 0, sizeof(c));
	memset(code, 0, sizeof(code));
	FILE *f;
	f = fopen("hfmTree.txt", "r");
	if (f == NULL)
	{
		cout << "open file failed" << endl;
		return 0;
	}
	char ck;
	ck = fgetc(f);
	//如果文件开头为空则终止操作
	if (ck <= 0 || ck == '\n')
	{
		cout << "File Empty" << endl;
		return 0;
	}
	HuffmanLen = ck - '0';
	for (int i = 0; i < HuffmanLen; i++)
	{
		fgetc(f);
		c[i] = fgetc(f);
		fgetc(f);
		char *strL = new char[1000];
		fscanf(f, "%s", strL);
```

```
            code[i] = strL;
        }
        c[HuffmanLen] = 0;
        return 1;
}

//对传入的字符串进行编码
void HuffmanEncode(char *s)
{
        fstream f("CodeFile.txt", ios::out);
        if (!f)
        {
            cout << "open file failed" << endl;
            return;
        }
        int len = strlen(s);
        cout<<"the result is : ";
        //不断查找待编码字符与在内存中的对应编码
        for (int i = 0; i < len; i++)
        {
            for (int j = 0; j < HuffmanLen; j++)
            {
                if (s[i] == c[j])
                {
                    f << code[j];
                    cout<<code[j];
                    break;
                }
            }
        }
        f.close();
        cout<<endl;
        cout << "ToBeTran.txt encode completed " << endl;
}

//递归构造哈夫曼树
void HuffmanBdTra(HFTree r, char *BDstr, char dt)
{
        //字符串为空时结束递归并将信息存入结点
        if (*BDstr == '\0')
        {
            r->data = dt;
        }
        else
```

```
    {
        //当前首字符为 1 就访问左子树。为 0 就访问右子树，字符串前进一个字符
        //如果子树不存在就构造一个新的结点

        if (*BDstr == '1')
        {
            if (!r->rchild)
            {
                HFnode *tp = new HFnode;
                tp->parent = r;
                r->rchild = tp;
            }
            HuffmanBdTra(r->rchild, BDstr + 1, dt);
        }
        else if (*BDstr == '0')
        {
            if (!r->lchild)
            {
                HFnode *tp = new HFnode;
                tp->parent = r;
                r->lchild = tp;
            }
            HuffmanBdTra(r->lchild, BDstr + 1, dt);
        }
    }
}

//调用函数，使用导入的信息构造哈夫曼树
void HuffmanBuild()
{
    HuffmanTree = new HFnode;
    //对于每一个字符和对应字符都构造一次
    for (int i = 0; c[i] != 0; i++)
    {
        HuffmanBdTra(HuffmanTree, code[i], c[i]);
    }
}

//递归先序遍历,将末端结点的 data 输入到输出流，返回当前编码的坐标
int HuffmanDcTra(HFTree t, char *s, int num, ostream &os)
{
    if (t)
    {
        if (t->lchild == NULL && t->rchild == NULL)
```

```
            {
                os << t->data;
                cout<< t->data;
                return num;
            }
            if (s[num] == '0')
            {
                return HuffmanDcTra(t->lchild, s, num + 1, os);
            }
            if (s[num] == '1')
            {
                return HuffmanDcTra(t->rchild, s, num + 1, os);
            }
        }
        return num;
}

//对传入的哈夫曼码进行解码
void HuffmanDecode(char *s)
{
        //字符串为空就中止操作
        if (s == NULL)
        {
            cout << "HuffmanCode Load Error" << endl;
            return;
        }
        fstream f("TextFile.txt", ios::out | ios_base::trunc);
        int len = strlen(s);
        cout<<"the result is : ";
        for (int i = 0; i < len - 1;)
        {
            i = HuffmanDcTra(HuffmanTree, s, i, f);
        }
        f.close();
        cout<<endl;
        cout << endl;
}

//先序遍历对于每个字符进行凹入表构造
void HuffmanTable(HFTree t, int num, int flag, ostream &os)
{
        //flag 标记判断左右子树
        //num 控制每个结点输出时前的空格
        //os 为输出的输出流
```

```
    if (t)
    {
        for (int i = 0; i < num * 4; i++)
        {
            cout << " ";
            os << " ";
        }
        if (t->lchild == NULL && t->rchild == NULL)
        {
            cout << t->data << " (" << flag << ")"
                 << "--------------------" << endl;
            os << t->data << " (" << flag << ")"
                << "--------------------" << endl;
        }
        else
        {
            cout << "NULL"
                 << " (" << flag << ")"
                 << "--------------------" << endl;
            os << "NULL"
                << " (" << flag << ")"
                << "--------------------" << endl;
        }

        if (t->lchild)
        {
            HuffmanTable(t->lchild, num + 1, 0, os);
        }
        if (t->rchild)
        {
            HuffmanTable(t->rchild, num + 1, 1, os);
        }
    }
}

//打印哈夫曼树的凹入吧并存入文件
void HuffmanDisplay()
{
    if (HuffmanLen == 0) //内存中不存在哈夫曼树就存内存导入
    {
        cout << "Find HuffmamTree in File" << endl;
        HuffmanLoad();
        HuffmanBuild();
    }
```

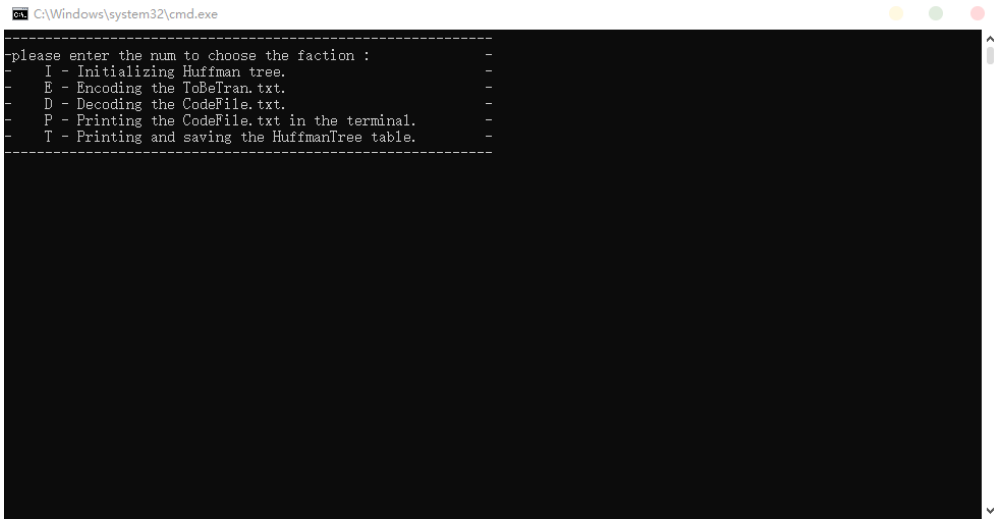
```
    fstream f("TreePrint.txt", fstream::out | ios_base::trunc);
    if (!f)
    {
        cout << "TreePrint.txt open error" << endl;
        return;
    }
    HuffmanTable(HuffmanTree, 0, -1, f);
    f.close();
    cout << "HuffmanTree Table saved completed" << endl;
}
```

四、调试分析

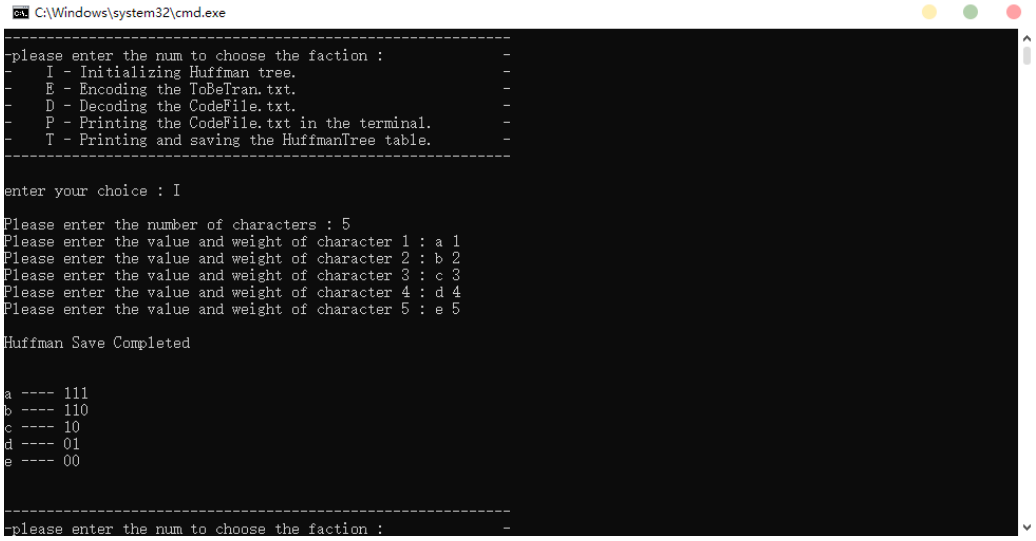
1、本次作业要求设计一个哈夫曼树编译系统，对相应文件内的暗码和明文进行相关操作。
2、对于哈夫曼编码的建立采用从根开始自上而下的遍历，采用递归的方式对树进行遍历遍历至叶结点结束，并返回对应字符的编码。
3、译码过程也是自上而下的匹配，同样采用递归，遇到匹配的字符结束递归。

五、用户手册

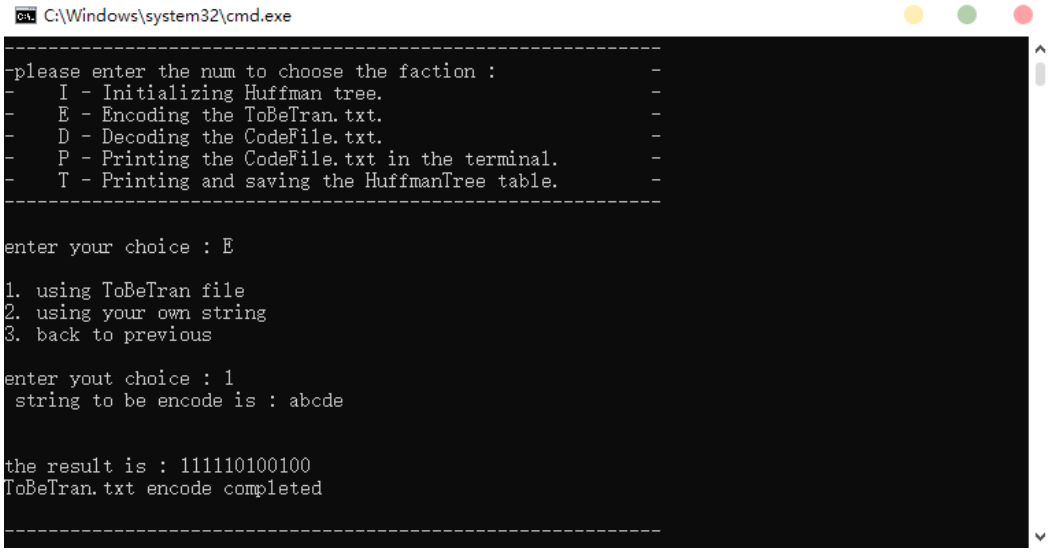
1、本程序的执行文件为：Huffman.exe
2、进入演示程序后，将显示如下的界面



3、输入 I 进行哈夫曼表构造，构造完效果如下：

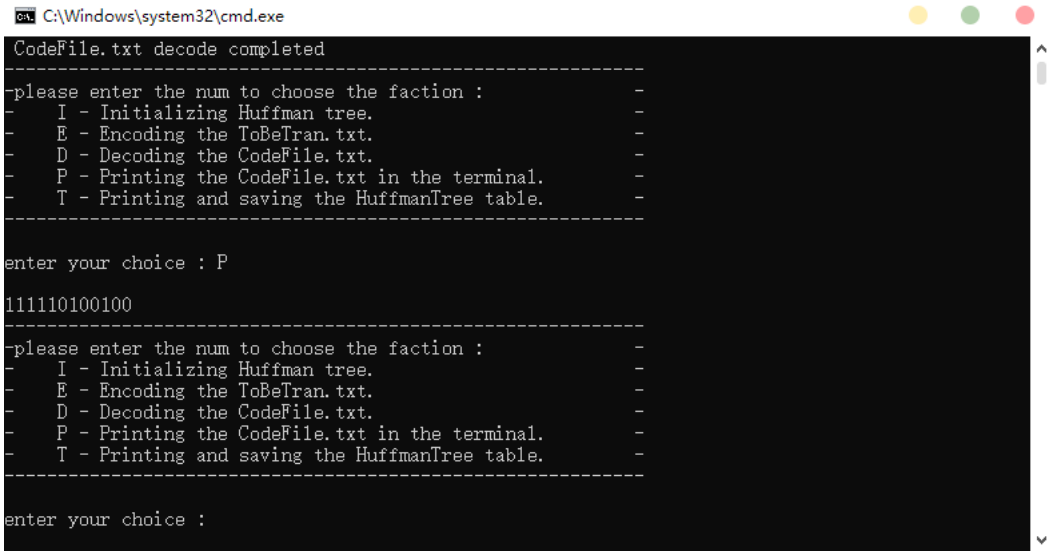


4、输入 E 进行编码.效果如下：

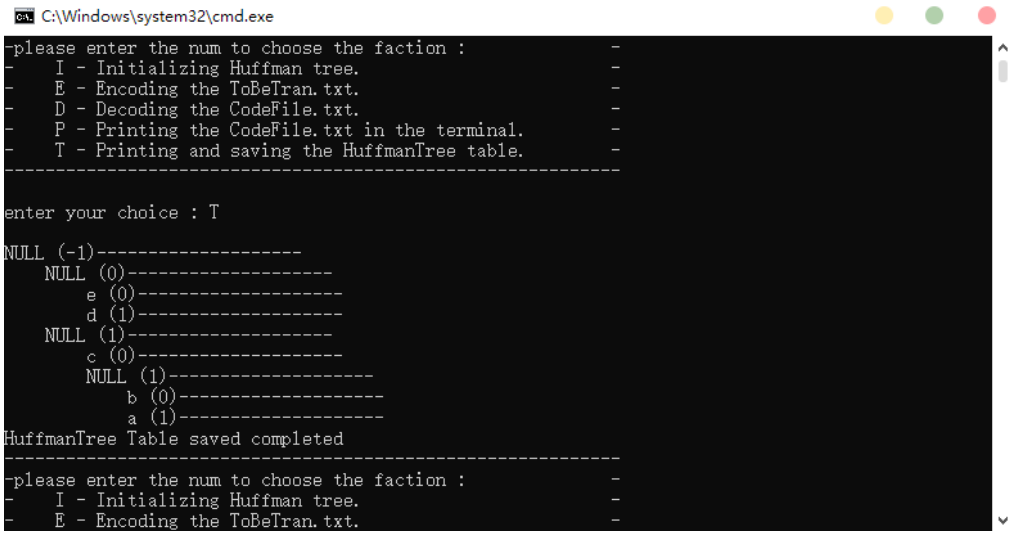


5、输入 D 进行译码，效果如下：

6、输入 P 打印 CodeFile 内容并存入 CodePrin，效果如下：

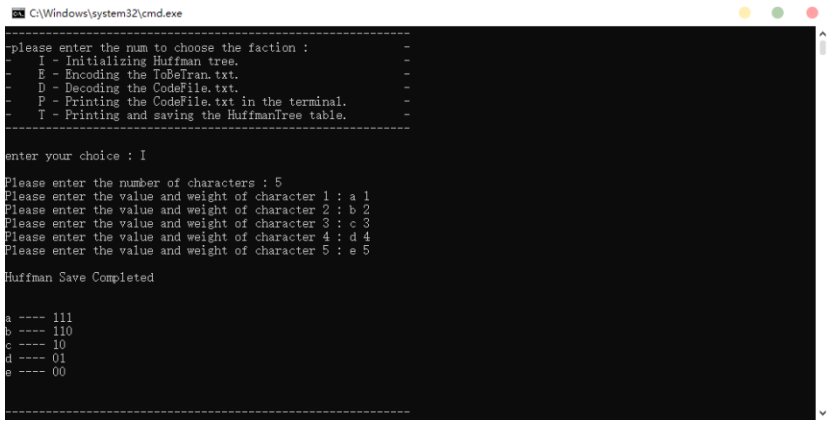


7、输入 T 打印哈夫曼树凹入表并存入文件：



六、测试结果

哈夫曼树初始化，根据提示进行操作可构造出哈夫曼树并生成对应哈夫曼编码



输入：
I
5
a 1
b 2
c 3
d 4
e 5

使用程序进行编码：

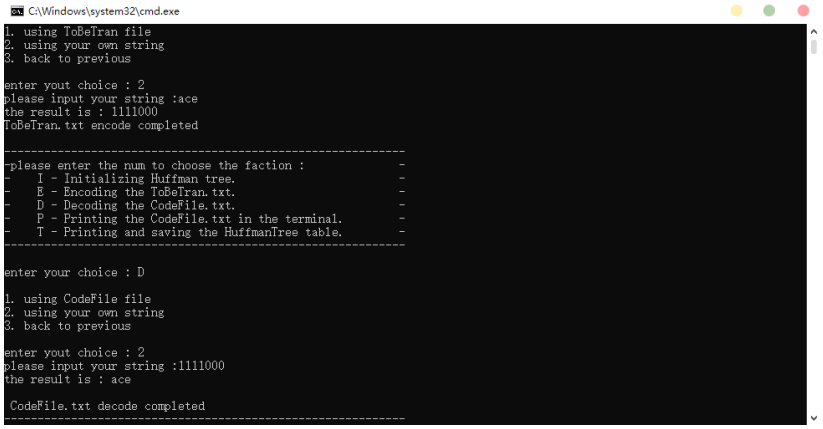
输入：

E

2

ace

输入 D 进行解码



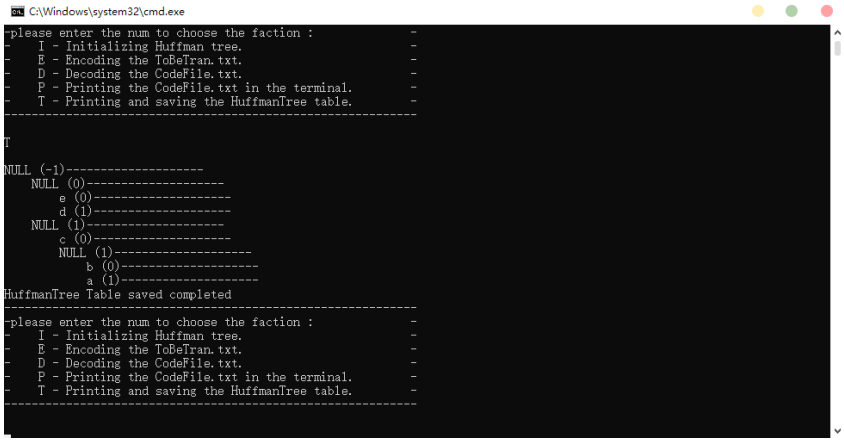
输入：

D

2

1111000

打印当前哈夫曼树的凹入表形式



输入：

T

七、附录

源程序文件名清单：CodeFile.txt，CodePrin.txt，hfmTree.txt，Huffman.cpp，Huffman.exe，TextFile.txt，ToBeTran.txt，TreePrint.txt