

**数据结构课程实践**

**学号 姓名**

实习报告-实习二

2023

目录

[需求分析 2](#_Toc87012154)

[程序所能达到的功能 2](#_Toc87012157)

[测试数据 2](#_Toc87012158)

[概要设计 2](#_Toc87012159)

[ADT定义 2](#_Toc87012160)

[主程序流程图 2](#_Toc87012161)

[模块层次与调用关系 2](#_Toc87012162)

[详细设计 2](#_Toc87012163)

[主程序代码 3](#_Toc87012165)

[函数调用关系图 3](#_Toc87012166)

[调试分析 3](#_Toc87012167)

[遇到的问题与解决 3](#_Toc87012168)

[经验与体会 3](#_Toc87012170)

[用户使用说明 3](#_Toc87012171)

[测试结果 3](#_Toc87012172)

[附录 3](#_Toc87012173)

**一、需求分析**

**程序所能达到的功能：**

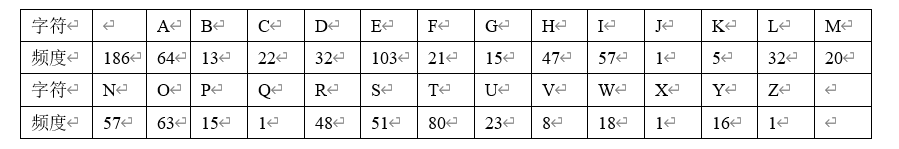
1.初始化： 从终端输入字符集大小n，以及n个字符和n个权值，建立哈夫曼树，并将它存入文件hfmtree文件中。

2.编码: 利用已建立的哈夫曼树，对文件tobetran中的正文进行编码，然后将结果存入codefile中。

3.译码: 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.印代码文件: 将文件codefile以紧凑格式显示在终端上，每行50个代码。

测试数据：利用下表给出的字符集和频度的实际统计数据建立哈夫曼树，并实现以下报文的编码和译码：“THIS PROGRAM IS MY FAVORITE”



输出结果：

1011111010010111111100100101111001010100011111001101010101100010111111101010100100111101010110110101000001011111000101011000

**二、概要设计**

ADT定义

ADT HuffmanTree{

数据对象D：D是具有相同特性的数据元素的集合

数据关系R：

若D = Φ,则R = Φ，称HuffmanTree为空二叉树

若D ≠ Φ,则R = {H}，H有如下二元关系：

（1）在D中存在唯一的称为根的数据元素root,它在关系H下无前驱；

（2）若D-{root} ≠ Φ,则存在D - {root} = {Dl,Dr},且Dl ∩ Dr = Φ；

（3）若Dl ≠ Φ,则Dl中存在唯一的元素xl，<root,xl>∈H,且存在Dl上的关系Hl⊂H；

若Dr ≠ Φ,则Dr中存在唯一的元素xr，<root,xr>∈H,且存在Dr上的关系Hr⊂H；

（4）(Dl,{Hl})是一棵符合本定义的二叉树，称为根的左子树.

基本操作P:

void CreateHuffmanTree(HuffmanTree tree[]);

操作结果:建立哈夫曼树tree。

void WriteHuffmanCode(char\* str,HuffmanCode code[], HuffmanTree tree[])；

初始条件:哈夫曼树tree存在。

操作结构:产生哈夫曼树编码。

void TransHuffmanCode(HuffmanCode code[], HuffmanTree tree[], char s[])；

初始条件:哈夫曼树存在。

操作结果:产生哈夫曼译码。

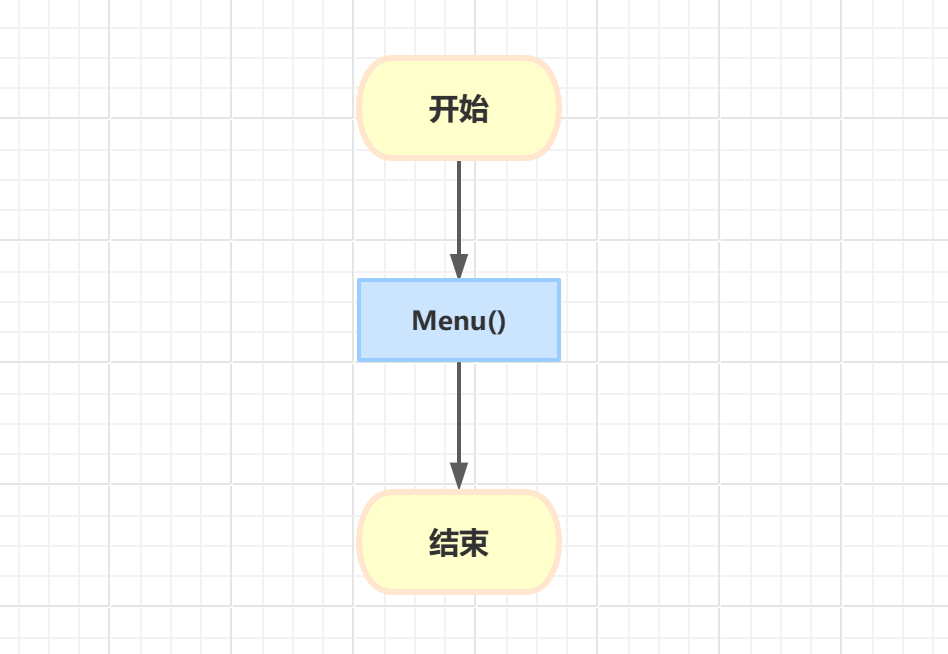
void PrintHuffmanTree(HuffmanTree tree[])；

初始条件:哈夫曼树存在。

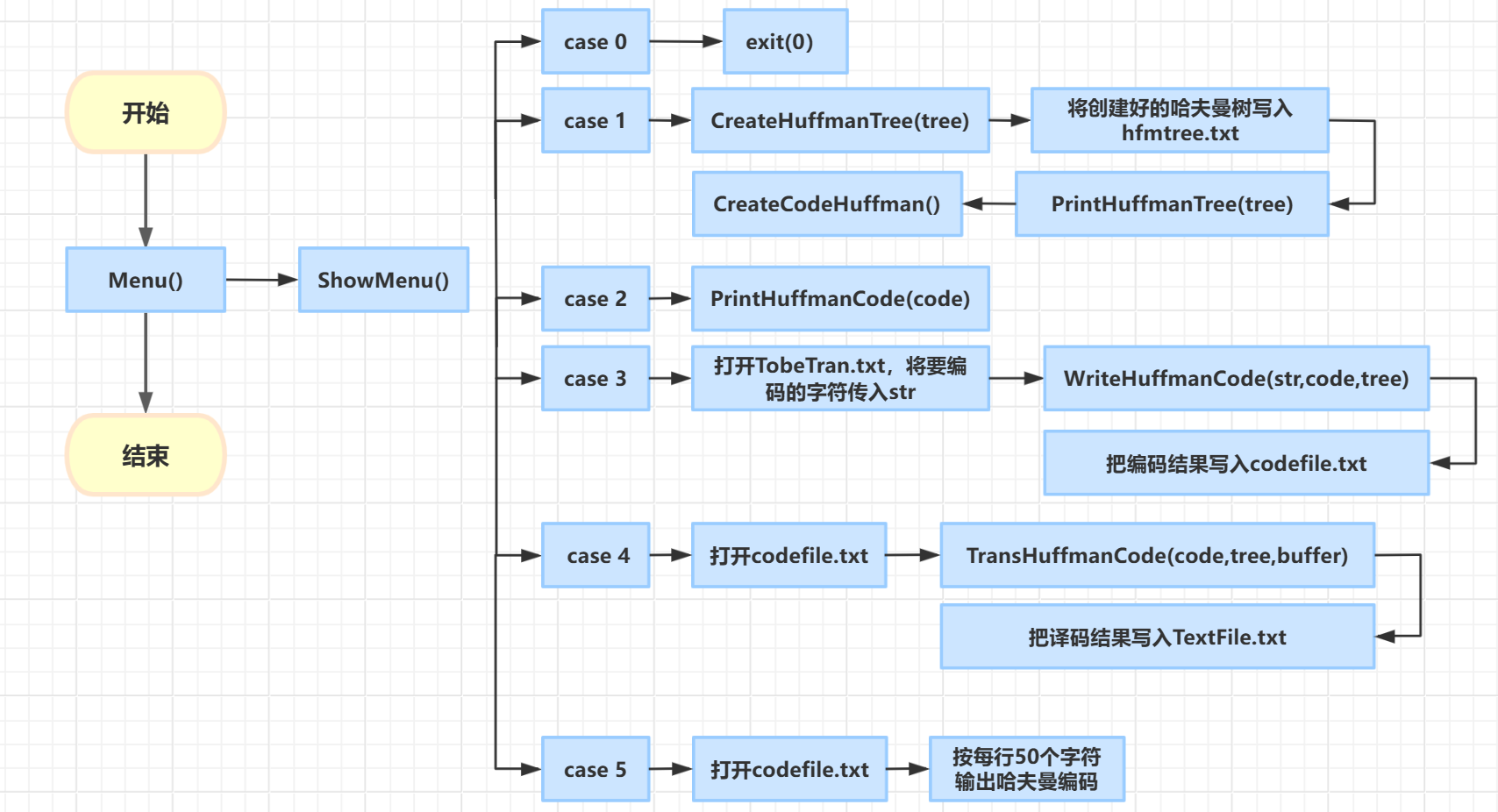
操作结果:用表格形式打印哈夫曼树。

}ADT HuffmanTree

**主程序流程图**



**模块层次与调用关系**



**三、详细设计**

**主程序代码**

#include<iostream>

#include<string>

#define MAXSIZE 1024

#define leafnum 28

#define hufftreenum 2\*leafnum//

#define maxdouble 9999//

using namespace std;

typedef struct Treenode {

char data;

double weight;

int lchild, rchild, parent;

}HuffmanTree;//哈夫曼树结点结构体

typedef struct Codenode {

char bits[leafnum + 1];

int start;

char ch;

}HuffmanCode;//哈夫曼编码表结构体

HuffmanCode code[leafnum + 1];

HuffmanTree tree[hufftreenum];

char hufcode[1000];//记录输入字符串的哈夫曼编码

char transcode[1000];

//定义全局数组存放字符名称及其对应频度

char ch[] = { '\0',' ','\n','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'};

float w[] = { 0,186,150,64,13,22,32,103,21,15,47,57,1,5,32,20,57,63,15,1,48,51,80,23,8,18,1,16,1 };

//建立哈夫曼树

void CreateHuffmanTree(HuffmanTree tree[]) {

int i, j, min1index, min2index;

double min1, min2;

//初始化哈夫曼树结点

for (i = 1; i <= hufftreenum; i++) {

tree[i].data = '\0';

tree[i].parent = 0;

tree[i].lchild = 0;

tree[i].rchild = 0;

tree[i].weight = 0;

}

for (i = 1; i <= leafnum; i++) {//为每个叶结点赋权值和字符

tree[i].data = ch[i];

tree[i].weight = w[i];

}

for (i = leafnum + 1; i <= hufftreenum; i++) {//构造哈夫曼树

min1index = 0;

min2index = 0;//两个最小值

min1 = min2 = maxdouble;//最小值索引

for (j = 1; j < i; j++) {//从已经建立的结点里找

if (tree[j].parent == 0) {//如果是未使用过的根节点

if (tree[j].weight < min1) {

min2 = min1;

min1 = tree[j].weight;

min2index = min1index;

min1index = j;

}

else {

if (tree[j].weight < min2) {

min2 = tree[j].weight;

min2index = j;

}

}

}

}

//建立新结点

tree[min1index].parent = i;

tree[min2index].parent = i;

tree[i].lchild = min1index;

tree[i].rchild = min2index;

tree[i].weight = tree[min1index].weight + tree[min2index].weight;

}

tree[hufftreenum - 1].parent = 0;

}

//建立哈夫曼编码表

void CreateCodeHuffman() {

int i, cur, p;

HuffmanCode buf;

for (i = 1; i <= leafnum; i++) {

buf.ch = ch[i];

buf.start = leafnum;

cur = i;

p = tree[i].parent;

while (p) {

buf.start--;

if (tree[p].lchild == cur) buf.bits[buf.start] = '0';

else buf.bits[buf.start] = '1';

cur = p;

p = tree[p].parent;

}

code[i] = buf;

}

}

//哈夫曼编码

void WriteHuffmanCode(char\* str,HuffmanCode code[], HuffmanTree tree[]) {

int i, j, k, n = 0;

cout << "得到的哈夫曼编码为：" << endl;

for (i = 0; i < strlen(str); i++) {

for (j = 1; j <= leafnum; j++) {

if (str[i] == tree[j].data) {

for (k = code[j].start; k < leafnum; k++) {

cout << code[j].bits[k];

hufcode[n] = code[j].bits[k];

n++;

}

}

}

}

}

//哈夫曼译码

void TransHuffmanCode(HuffmanCode code[], HuffmanTree tree[], char s[]) {

int i;

int n = 0;

char\* q = NULL;

i = hufftreenum - 1;

q = s;

while (\*q != '\0') {

if (\*q == '0') i = tree[i].lchild;

if (\*q == '1') i = tree[i].rchild;

if (tree[i].lchild == 0 && tree[i].rchild == 0) {

transcode[n++] = code[i].ch;

i = hufftreenum - 1;

}

q++;

}

cout << endl;

}

//打印哈夫曼树

void PrintHuffmanTree(HuffmanTree tree[]) {

int i;

cout << "根据字符的使用概率所建立的哈夫曼树为:" << endl;

cout << "字符序号 字符名称 字符频率 双亲位置 左孩子 右孩子" << endl;

for (i = 1; i < hufftreenum; i++) {

if (tree[i].data == '\n') {

cout << " " << i << "\t " << "\\n" << " " << '\t' << " ";

cout << tree[i].weight << '\t' << '\t' << tree[i].parent << '\t' << " " << tree[i].lchild << '\t' << " " << tree[i].rchild << endl;

}

else

{

cout << " " << i << "\t " << tree[i].data << '\t' << '\t' << " ";

cout << tree[i].weight << '\t' << '\t' << tree[i].parent << '\t' << " " << tree[i].lchild << '\t' << " " << tree[i].rchild << endl;

}

}

}

//输出每个字符的哈夫曼编码

void PrintHuffmanCode(HuffmanCode code[]) {

int i, j;

cout << "根据哈夫曼树对字符所建立的哈夫曼编码为:" << endl << "字符序号 字符名称 字符编码" << endl;

for (i = 1; i <= leafnum; i++) {

if(code[i].ch=='\n')

cout << " " << i << '\t' << " " << "\\n" << " " << '\t';

else

cout << " " << i << '\t' << " " << code[i].ch << '\t' << '\t';

for (j = code[i].start; j < leafnum; j++) {

cout << code[i].bits[j];

}

cout << endl;

}

}

void ShowMenu() {

cout << " =============================================================================\n";

cout << "|| ★★★★★★★哈夫曼编码与译码★★★★★★★ ||\n";

cout << "||============================================================================||\n";

cout << "||============================================================================||\n";

cout << "|| 【0】--- 退出 ||\n";

cout << "|| 【1】--- 创建哈夫曼树 ||\n";

cout << "|| 【2】--- 打印哈夫曼编码表 ||\n";

cout << "|| 【3】--- 进行哈夫曼编码 ||\n";

cout << "|| 【4】--- 进行哈夫曼译码 ||\n";

cout << "|| 【5】--- 打印哈夫曼编码文件 ||\n";

cout << " ==============================================================================\n";

cout << "请输入数字来选择对应的功能：";

}

void Menu() {

while (1)

{

system("cls");

ShowMenu();

int num;

if (!(cin >> num))

{

cout << "输入格式错误！请重新输入：" << endl;

}

else {

switch (num) {

case 0:

exit(0);

case 1://创建哈夫曼树和哈夫曼编码表

{

system("cls");

CreateHuffmanTree(tree);

FILE\* fp;

if ((fp = fopen("hfmTree.txt", "w")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

for (int i = 1; i < hufftreenum; i++) {

fprintf(fp, "%d %c %.0f %d %d %d\n", i, tree[i].data, tree[i].weight, tree[i].parent, tree[i].lchild, tree[i].rchild);

}

cout << "哈夫曼树成功写入hfmTree.txt文件" << endl;

fclose(fp);

PrintHuffmanTree(tree);

CreateCodeHuffman();

break;

}

case 2://打印哈夫曼编码表

{

system("cls");

PrintHuffmanCode(code);

break;

}

case 3://哈夫曼编码

{

system("cls");

FILE\* fp;

if ((fp = fopen("TobeTran.txt", "r")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

char\* str = new char[MAXSIZE];

for (int i = 0; i < MAXSIZE; i++)

str[i] = '\0';

int j = 0;

while (!feof(fp)) {

str[j++] = fgetc(fp);

}

str[j-1] = '\0';

fclose(fp);

printf("将要编码的字符串为：%s\n",str);

WriteHuffmanCode(str, code, tree);

cout << endl<<"编码成功" << endl;

FILE\* fp2;

if ((fp2 = fopen("codefile.txt", "w")) == NULL)

{

printf("文件打开失败！\n");

exit(0);

}

fprintf(fp2, "%s", hufcode);

fclose(fp2);

break;

}

case 4://哈夫曼译码

{

system("cls");

FILE\* fp;

if ((fp = fopen("codefile.txt", "r")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

char\* buffer = new char[MAXSIZE];

for (int i = 0; i < MAXSIZE; i++)

buffer[i] = '\0';

int j = 0;

while (!feof(fp))

buffer[j++] = fgetc(fp);

buffer[j - 1] = '\0';

fclose(fp);

TransHuffmanCode(code, tree, buffer);

cout << "codefile文件中代码译码为:" << transcode << endl;

FILE\* fp2;

if ((fp2 = fopen("TextFile.txt", "w")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

fprintf(fp2, "%s", transcode);

fclose(fp2);

cout << "译码成功写入TextFile文件" << endl;

break;

}

case 5: //打印codefile代码文件

{

system("cls");

FILE\* fp;

if ((fp = fopen("codefile.txt", "r")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

char\* buffer = new char[MAXSIZE];

for (int i = 0; i < MAXSIZE; i++)

buffer[i] = '\0';

int j = 0;

while (!feof(fp))

buffer[j++] = fgetc(fp);

buffer[j - 1] = '\0';

cout << "哈夫曼编码为：" << endl;

for (int k = 0; k < strlen(buffer); k++) {

cout << buffer[k];

if ((k + 1) % 50 == 0) cout << endl;

}

fclose(fp);

FILE\* fp2;

if ((fp2 = fopen("TextFile.txt", "r")) == NULL) {

cout << "文件打开失败！" << endl;

exit(0);

}

char\* buf = new char[MAXSIZE];

for (int i = 0; i < MAXSIZE; i++)

buf[i] = '\0';

j = 0;

while (!feof(fp2))

buf[j++] = fgetc(fp);

buf[j - 1] = '\0';

fclose(fp2);

break;

}

}

}

system("pause");

}

}

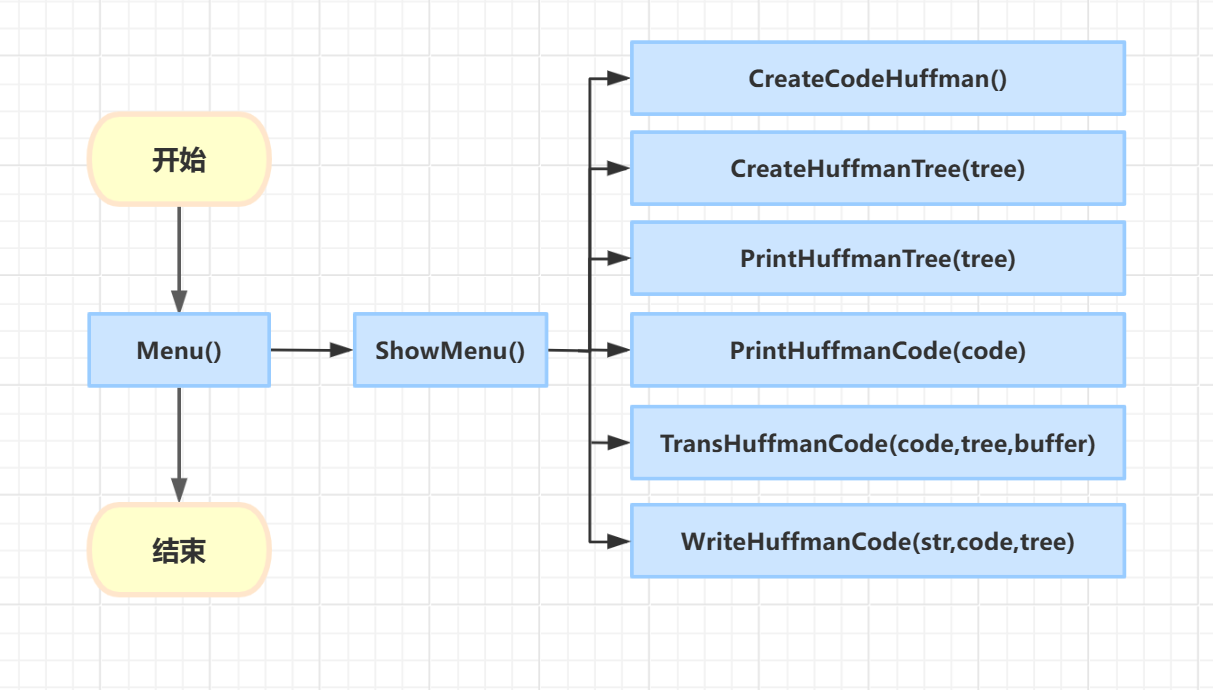
int main() {

Menu();

return 0;

}

**函数调用关系图**



**四、调试分析**

**遇到的问题与解决**

1. 用fread函数读取文件中待编码的字符会有空格，应该如何处理？

答：在哈夫曼树中添加新结点‘\n’，赋予权重

1. 在建立哈夫曼树中，如何取到最小值和次小值？

答：遍历结点，找到遍历序列中的最小值min1，然后接着找比当前最小值min1更小的最小值curmin，如果curmin<min1,那么把min1赋值给次小值min2,把curmin赋值给min1,否则把curmin赋值给min2

**经验与体会**

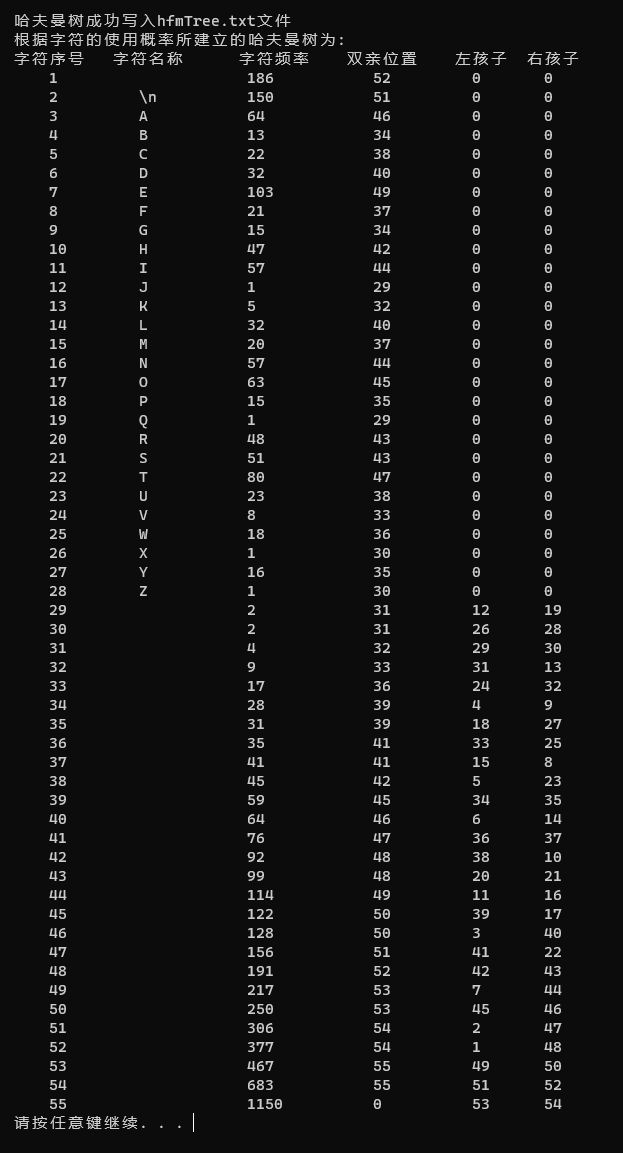
通过实现哈夫曼树编码器和译码器，我深刻体会到了哈夫曼编码的优势。它能够根据不同字符出现频率的不同，为每个字符分配最优的编码长度，实现了对数据的高效压缩和解压缩。同时，哈夫曼编码也具有无损性，即通过解码可以完全还原原始数据，不会产生任何信息损失。

**五、用户使用说明**

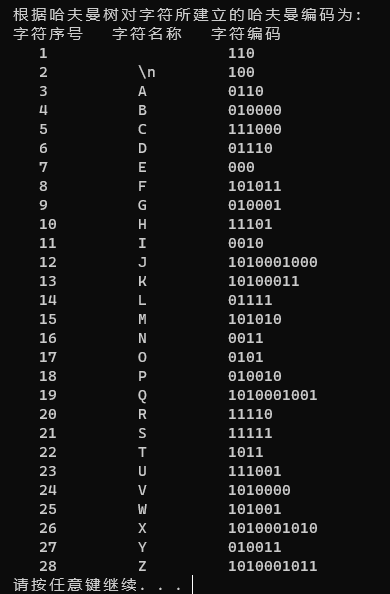
用户首先要输入“1”，创建哈夫曼编码树，然后根据菜单界面显示的选项数字，输入对应的编号，进行各项功能的使用。

1. **测试结果**

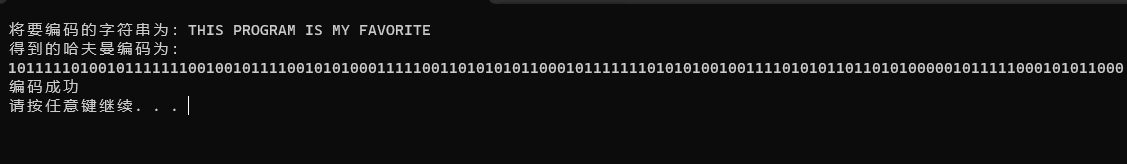
**1.创建哈夫曼树**



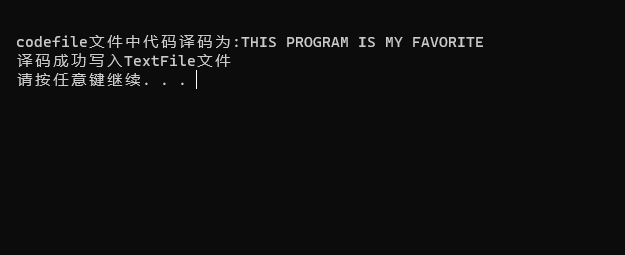
**2.打印哈夫曼编码表**



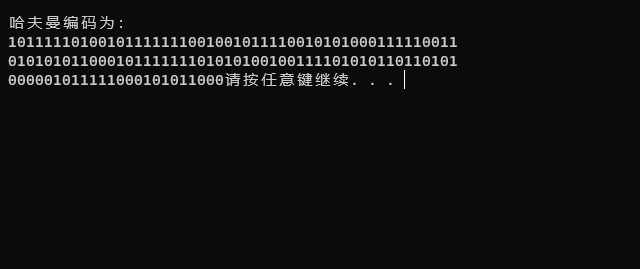
**3.进行哈夫曼编码**



**4.进行哈夫曼译码**



**5.打印哈夫曼编码文件**



**七、附录**

程序文件名清单：

algorithm book.sln

hfmTree.txt

codefile.txt

TextFile.txt

TobeTran.txt