**计算机软件基础实习报告**

**题目** 哈夫曼编/译码器

**班级**  软工四班

**姓名** 罗佳淦

**学号**  423141030117

**指导老师** 张敬敏

**完成日期**  2025年1月10日

## 一、需求分析

1. 本程序中，需要完成对n个字符的字符集中所有字符及其权值的线性储存(每种元素只存一遍，若再次存入一个已有的元素则给出提示)并排序，将存入的字符根据其权重建立哈夫曼树，并完成以下功能：

(1)编码：可完成从内存读入和文件读入两种方式获取字符及其权值构造哈夫曼树，，再读入字符串并利用哈夫曼树进行对各个字符的编码，将其转化为哈夫曼编码串（若代码中有元素未被编码过，则不进行译码，提示错误）。

(2)译码：可完成从内存读入和文件读入两种方式获取编码串，将文件中的编码串进行译码。最后在终端上打印译码结果

2. 演示程序以用户与计算机交互方式执行，即在计算机终端上显示“提示信息”之后，由用户在键盘上输入演示程序中规定的运算命令；相应的输入数据显示在其后。

3. 程序执行的命令包括：

(0)退出(1)输入字符和权值；(2)查看所有字符情况(3)删除字符(4)构造哈夫曼树(5)查看哈夫曼树(6)编码(7)译码(8)读取文件构造哈夫曼树(9)读取文件进行编码(10)读取文件进行译码

4. 测试数据

(1)利用教科书例 6-2 中的数据调试程序。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HT初始状态 | | | | |
| Node | Weight | Parent | Lchild | Rchild |
| 1 | 5 | 0 | 0 | 0 |
| 2 | 29 | 0 | 0 | 0 |
| 3 | 7 | 0 | 0 | 0 |
| 4 | 8 | 0 | 0 | 0 |
| 5 | 14 | 0 | 0 | 0 |
| 6 | 23 | 0 | 0 | 0 |
| 7 | 3 | 0 | 0 | 0 |
| 8 | \* | 0 | 0 | 0 |
| 9 | \* | 0 | 0 | 0 |
| 10 | \* | 0 | 0 | 0 |
| 11 | \* | 0 | 0 | 0 |
| 12 | \* | 0 | 0 | 0 |
| 13 | \* | 0 | 0 | 0 |
| 14 | \* | 0 | 0 | 0 |
| 15 | \* | 0 | 0 | 0 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HT终结状态 | | | | |
| Node | Weight | Parent | Lchild | Rchild |
| 1 | 5 | 9 | 0 | 0 |
| 2 | 29 | 14 | 0 | 0 |
| 3 | 7 | 10 | 0 | 0 |
| 4 | 8 | 10 | 0 | 0 |
| 5 | 14 | 12 | 0 | 0 |
| 6 | 23 | 13 | 0 | 0 |
| 7 | 3 | 9 | 0 | 0 |
| 8 | 11 | 11 | 0 | 0 |
| 9 | 8 | 11 | 7 | 1 |
| 10 | 15 | 12 | 3 | 4 |
| 11 | 19 | 13 | 9 | 8 |
| 12 | 19 | 14 | 5 | 10 |
| 13 | 42 | 15 | 11 | 6 |
| 14 | 58 | 15 | 2 | 12 |
| 15 | 100 | 0 | 13 | 14 |

(2)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"。

字符 \_ A B C D E F G H I J K L M

频度 186 64 13 22 32 103 21 15 47 57 1 5 32 20

字符 N O P Q R S T U V W X Y Z

频度 57 63 15 1 48 51 80 23 8 18 1 16 1

操作结果：

哈夫曼编码：A1010\B100000\C00000\D10110\E010\F110011\G100001\H0001\I0110\J1100001000\K11000011\L10111\M110010\N0111\O1001\P100010\Q1100001001\R0010\S0011\T1101\U00001\V1100000\W110001\X1100001010\Y100011\Z1100001011

报文编码：1101000101100011111100010001010011000010010101011001011101100011111110010100011111110011101011000001001001001101101010

译码：THIS\_PROGRAM\_IS\_MY\_FAVORITE

(3)根据(2)中哈夫曼树，对英文文件中的内容进行编码和译码，文件内

容自定，但要求字符总数>100。

编码内容：HGU\_IS\_THE\_BEST\_UNIVERSITY

编码结果：000110000100001111011000111111101000101011110000001000111101111000010111011011000000100010001101101101100011

## 二、概要设计

### 1. 程序使用的抽象数据类型

为实现上述程序功能，需要两个抽象数据类型：线性表和二叉树。

1）线性表的抽象数据类型定义为：

ADT LinearList

{

**数据对象：**D = {ai|ai∈ElemSet,i=1,2,…,n,n≥0}

**数据关系：**R = {<ai-1,ai>| ai-1,ai∈D,i=2,…,n}

**基本操作：**

ListInit(&L)

操作结果：构造一个空的线性表L。

ListAdd (&L,e)

初始条件：线性表L已存在。

操作结果：在有线性表L的未尾插入元素e。

ListVisit(&L)

初始条件：线性表L已存在。

操作结果：遍历线性表并输出各个字母及其权值。

ListDel(&L,Elem D)

初始条件：线性表L存在.

操作结果：在表内查找是否有字符D;若存在，则将其删除,返回OK；

若不存在，则返回ERROR

}ADT LinearList

2）. 二叉树的抽象数据类型定义为：

ADT BiTree{

**数据对象D**：D是具有相同特征数据元素的集合；

**数据关系R**：

若D=Φ，则R=Φ，称BinaryTree为空二叉树；

若D≠Φ，则R={H}，H是如下二元关系；

（1）在D中存在惟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}={D1,Dr}，且D1∩Dr =Φ；

（3）若D1≠Φ，则D1中存在惟一的元素x1，<root,x1>∈H，且存在D1上的关系H1 ⊆H；若Dr≠Φ，则Dr中存在惟一的元素xr，<root,xr>∈H，且存在上的关系Hr ⊆H；H={<root,x1>,<root,xr>,H1,Hr}；

（4）(D1,{H1})是一棵符合本定义的二叉树，称为根的左子树；(Dr,{Hr})是一棵符合本定义的二叉树，称为根的右子树。

**基本操作：**

TreeInit()

操作结果：初始化二叉树。

TreeDataIn(&T,&L)

初始条件：L为线性表、二叉树T已存在。

操作结果：将线性表L中的所有字符及其权重输入到二叉树中。

TreeCreat(&T)

初始条件：树T存在。

操作结果：创建哈夫曼树。

TreeNodeSort(&T)

初始条件：树T存在。

操作结果：将树中的每个节点按其权重排序。

TreeListOut(&T)

初始条件：二叉树T存在；

操作结果：将树中的数据输出，包括每个节点的字符、权重、父节点、左右节点、哈夫曼编码。

GetHafmCode(&T)

初始条件：二叉树T存在且哈夫曼树构建完成

操作结果：给每个哈夫曼树的叶子节点中的字符数值填充上哈夫曼编码。

HafmEncode( &T,char E[], int len)

初始条件：二叉树T存在，E为编码字符的数组，len为数组长度。

操作结果：读取字符数组；输出其哈夫曼编码数组。

HafmDecode(TNode &T,char n[],int len)

初始条件：二叉树T存在，E为译码字符的数组，len为数组长度。

操作结果：读取译码字符数组；输出其对应的编码数组。

}ADT BiTree

### 2. 本程序包含四个模块

1) 主程序模块

void main()

{

while(!退出)

{

输入字符和权值；

查看所有字符情况；

删除字符

构造哈夫曼树；

查看哈夫曼树

编码

译码

读取文件构造哈夫曼树

读取文件进行编码

读取文件进行译码

}

｝

2)二叉树创建单元模块 实现有序表的抽象数据类型；

3)线性表单元块，实现线性表的抽象数据类型

4）文件控制模块，实现从文件获取数据的功能

各模块之间的调用关系如下:

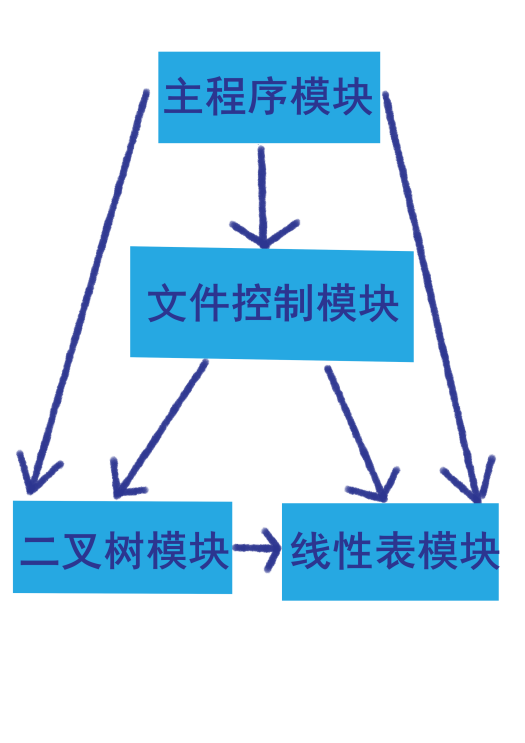


图2-1 模块调用关系图

## 详细设计

### 1. 线性表ADT的 C语言形式

1）线性表的C语言定义

集合中元素存放在以单链表存储的有序表中，集合元素类型、单链表结点类型的C语言定义如下：

typedef struct SQList {

Elem E[MAXSIZE];//字符元素

weight W[MAXSIZE];//权值

int top;

}Node,\*LNode;//节点类型

2）. 线性表的基本操作的C语言描述

int ListInit(LNode &L)

//构造一个的空的顺序表L,并返回OK

int ListAdd(LNode& L, Elem D, weight W)；

//在顺序表L中添加字符和权重

//输入成功返回OK；输入失败返回ERROR；

int ListDel(LNode& L, Elem D)

//在顺序表中查找字符D；

若找到，将其删除并返回OK；若未找到，返回ERROR;

int ListVisit(LNode& L)

3）线性表的基本操作C语言实现的伪代码

int ListInit(LNode &L){

L = new SQList;//分配空间

L->top=0;//头

return OK;

}

int ListAdd(LNode& L, Elem D, weight W) {

L->E[L->top] = D;

L->W[L->top] = W;

L->top++;

return OK;

}

int ListDel(LNode& L, Elem D) {

for (int i = 0; i < L->top; i++) {

if (L->E[i] == D) {

while (i + 1 < L->top)

{L->E[i] = L->E[i + 1];

L->W[i] = L->W[i + 1];

i++;

} L->top--;//长度减1

return OK;

}}}

int ListVisit(LNode& L) {

for (int i = 0; i < L->top; i++) {

cout << "----------------------" << endl;

cout << "第" << i + 1 << "项" << endl;

cout << "元素:'" << L->E[i] << "'" << endl;

cout << "权重：" << L->W[i] << endl;

}

cout << "----------------------" << endl;

return OK;}

### 2. 二叉树ADT的C语言描述

1）二叉树的C语言定义

为操作方便，二叉树节点中除了保存有指向父、左孩子、右孩子的节点指针外，还有一个指向邻节点的值针，将树的节点像链表一样链接起来

typedef struct TreeNode {

Elem E;//字符元素

weight W=0;//权值

char hafmcode[MAXSIZE];//哈夫曼编码

int codeleng=0;//哈夫曼编码长度

struct TreeNode\* Next = NULL;//邻节点

struct TreeNode\* Lchild = NULL;//左孩子

struct TreeNode\* Rchild = NULL;//右孩子

struct TreeNode\* Parent = NULL;//父节点

}TreeNode, \* TNode;2）. 集合基本操作的C语言描述

int TreeInit(TNode &T)

//二叉树的初始化

int TreeDataIn(TNode& T, LNode& L)

//将顺序表内的字符元素及其权重输入到二叉树中

int TreeNodeSort(TNode& T)

//对二叉树节点链表进行插入排序

int TreeListOut(TNode& T)

//输出二叉树的所有节点及各个节点的详细信息

int TreeCreat(TNode &T)

//创建哈夫曼树给，所有树节点连接起来

void GetHafmCode(TNode &T,int top)

//递归获取哈夫曼编码，从树根开始，读到左孩子给0，读到右孩子给1

int HafmEncode(TNode &T,char E[], int len)

//输入一串字符数组及其长度，将其转化为哈夫曼编码，将编码直接输出

int HafmDecode(TNode &T,char n[],int len)

//输入一串哈夫曼编码数组及其长度，将其转化为字符数组并直接输出

3）二叉树的基本操作C语言实现的伪代码

int TreeInit(TNode &T) {//初始化

T = new TreeNode;

return OK;}

int TreeDataIn(TNode& T, LNode& L) {

//将顺序表内的字符元素及其权重输入到二叉树中

if (L->top != 0) {

TNode P = T;

for (int i = 0; i < L->top; i++) {

TNode N = new TreeNode;

N->E = L->E[i];

N->W = L->W[i];

P->Next = N;

P = P->Next;

}return OK;

}

}

TreeNodeSort(TNode& T) {//对二叉树节点链表进行插入排序

TNode P = T;

TNode Pnext = P->Next;

while (true)

{ if (Pnext->W<P->W) {

TNode Q = T;

TNode Qnext = Q->Next;

while (Q!=P) {

if (Qnext->W > Pnext->W) {

P->Next = Pnext->Next;Pnext->Next = Qnext;Q->Next = Pnext;

break;}

Q = Q->Next;Qnext = Q->Next;}}

else {P = P->Next;}

if (P == NULL || P->Next == NULL) {return OK;}

Pnext = P->Next;

}}

int TreeListOut(TNode& T) {//输出二叉树的所有节点及各个节点的详细信息

TNode P = T->Next;

while (P!= NULL)

{

if (P->Lchild == NULL && P->Rchild == NULL) {

cout << P->E << "\t" << P->W << "\t无\t无\t";

}else if(P->Lchild != NULL && P->Rchild == NULL) {

cout << P->E << "\t" << P->W << "\t"<<P->Lchild->E<<"\t无\t - "<<endl;}

else if(P->Lchild == NULL && P->Rchild != NULL) {

cout << P->E << "\t" << P->W << "\t无\t"<< P->Rchild->E << "\t - "<<endl;}

else {

cout << P->E << "\t" << P->W << "\t" << P->Lchild->E << "\t" << P->Rchild->E << "\t - "<<endl;}

P = P->Next;

}return OK;

}

int TreeCreat(TNode &T) {

//创建哈夫曼树，给所有树节点连接起来，非叶子节点字符用‘\*’代替

TNode Fir = T->Next;

TNode Sec = Fir->Next;

while (Sec!=NULL)

{

TNode N = new TreeNode;

TNode P = T;

while (P->Next != NULL) {

P = P->Next;

}

P->Next = N;

N->W = Fir->W + Sec->W;

N->E = '\*';

N->Lchild = Fir; N->Rchild = Sec;

Fir->Parent = N;

Sec->Parent = N;

TreeNodeSort(T);

Fir = T->Next;

while (Fir->Parent != NULL) {

Fir = Fir->Next;

}

Sec = Fir->Next;

while (Sec!=NULL&&Sec->Parent!=NULL) {

Sec = Sec->Next;

}

}

T->Lchild = Fir;

Fir->Parent = T;

return OK;

}

void GetHafmCode(TNode &T,int top) {

//递归获取哈夫曼编码，从树根开始读，读到左孩子给0，读到右孩子给1。

top++;

if (T->Lchild != NULL) {

temp[top-1] = '0';

GetHafmCode(T->Lchild,top);

}

if (T->Rchild != NULL) {

temp[top-1] = '1';

GetHafmCode(T->Rchild,top);

}

if (T->Lchild == NULL && T->Rchild == NULL) {

T->codeleng = top-1;

for (int i = 0; i < top; i++) {

T->hafmcode[i] = temp[i];

}

}

}

int HafmEncode(TNode &T,char E[], int len) {

TNode P = T->Next;

for (int i = 0; i < len; i++) {

while (P->E != E[i])

{

P = P->Next;

if(P->E == E[i]) {

for (int j = 0; j < P->codeleng; j++) {

cout << P->hafmcode[j];}}

P = T->Next;

}return OK;}}

int HafmDecode(TNode &T,char n[],int len) {

TNode P = T->Lchild;

for (int i = 0; i < len; i++) {

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

P = P->Lchild;

}

else {

P = P->Rchild;

}

if (P->Lchild == NULL && P->Rchild == NULL) {

cout << P->E;

P = T->Lchild;

}

}

return OK;

}

**3.. 文件操作的代码实现**

1）语言定义

int EncodeFromfile(TNode &T)

//从文件中读取字符，根据哈夫曼树进行编码

int DecodeFromfile(TNode& T)

//从文件中读取哈夫曼编码，根据哈夫曼树进行译码

int GetElemFromfile(LNode &L)

//从文件中读取字符和权值，输入到线性表中

1. 伪代实现

int EncodeFromfile(TNode &T) {

fstream fin;

fin.open("Encode.txt", ios::in);//打开文件

char Inline[MAXSIZE];

int top = 0;

while (fin>>Inline[top])//从文件中读取字符和权重

{ top++;}

HafmEncode(T, Inline, top);//进行编码

fin.close();

return OK;

}

int DecodeFromfile(TNode& T) {

fstream fin;

fin.open("Decode.txt", ios::in );//打开文件

char Inline[MAXSIZE];

int top = 0;

while (fin >> Inline[top])//从文件中读取哈夫曼码

{top++;}

HafmDecode(T, Inline, top);//译码

fin.close();//关闭文件

return OK;

}

int GetElemFromfile(LNode &L) {//从文件中读取字符和权值

fstream fin;

fin.open("Tree.txt", ios::in);//打开文件

int n; char m;

while (fin >> m&& fin >> n){//依次读取字符、权值

ListAdd(L, m, n);//入表

}

fin.close();

cout << "读取成功" << endl;

return OK;}

4. 主函数伪码算法

int main()

{// 主函数

while(loop)

{

switch (loop) {

case 0://退出

break;

case 1:

cin >> E;

cin >> W;//输入权值和字符到线性表中

ListAdd(L,E,W);

break;

case 2:

ListVisit(L);//遍历线性表

break;

case 3:

cin >> E;//输入字符，在线性表中查找并删除

ListDel(L, E);

break;

case 4:

TreeDataIn(T, L);

TreeNodeSort(T);构建哈夫曼树

TreeCreat(T);

break;

case 5:

HafmEncodeInit(T);

TreeListOut(T);输出哈夫曼树

break;

case 6:

linelen = 0;

while (cin >> Inline[linelen])

{

linelen++;

if (Inline[linelen - 1] == '\*')break;//输入字符数组进行编码

}

HafmEncode(T, Inline, linelen);

break;

case 7:

linelen = 0;

while (cin >> Inline[linelen])//请输入字符进行译码(输入\*停止)

{

linelen++;

if (Inline[linelen - 1] == '\*')break;

}

HafmDecode(T, Inline, linelen);

break;

case 8:

GetElemFromfile(L);//读取文件中的字符和权值

cin >> n;//输入是否进行遍历的命令

if (n) {

ListVisit(L);

}

cin >> n;//输入是否构建哈夫曼树的命令

if (n) {

TreeDataIn(T, L);

TreeNodeSort(T);

TreeCreat(T);

}

break;

case 9:

EncodeFromfile(T);//读取文件进行编码

break;

case 10:

DecodeFromfile(T);//读取文件进行译码

break;

default:

cout << "无效输入" << endl;

break;//输入其他不符操作时的结果

}

}

}//main

5. 函数的调用关系图反映了演示程序的层次结构 :

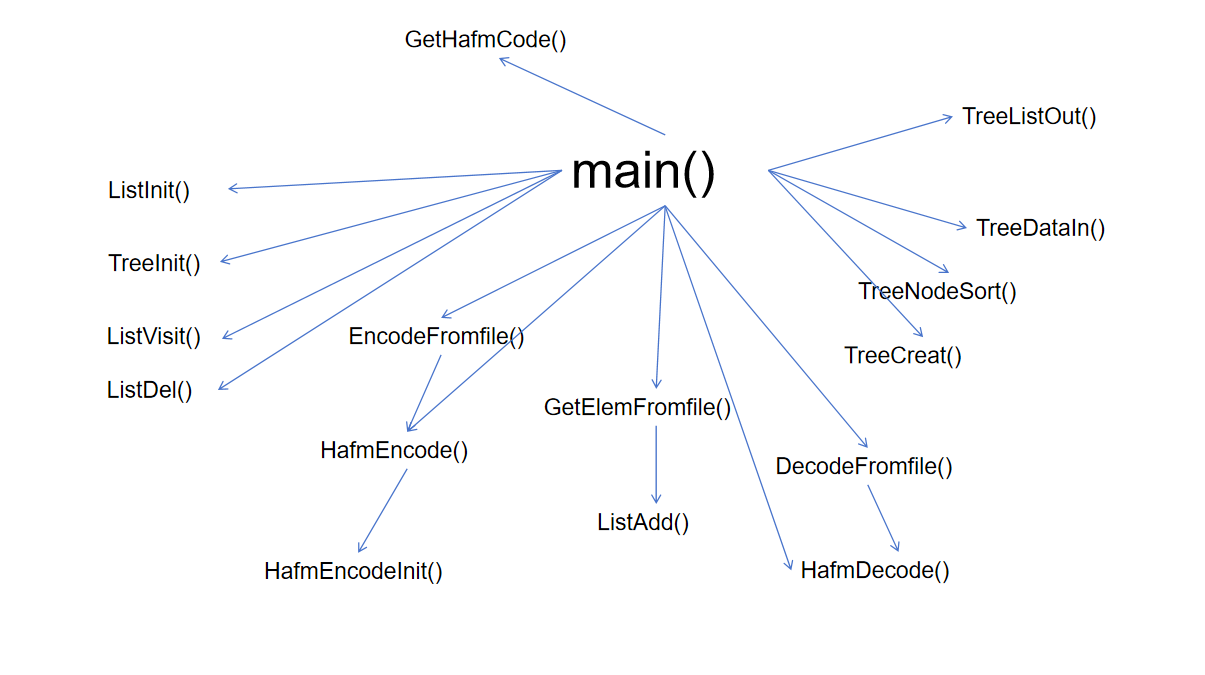


图3-1 函数调用图

## 四、调试分析

1. 问题回顾与设计分析

1)由于在TreeNodeSort()函数排序时没有区分特殊情况和多次混淆头节点和头节点的下一个节点，导致排序及在后面的哈夫曼树的构建上出现了很多问题，反复调试后最终得以解决。

2)由于对C++的读取文件的操作不了解，花了很多时间学习文件输入流，最终

在函数EncodeFromfile()、DecodeFromfile()、GetElemFromfile()中完成了文件的读取操作。

1. 在哈夫曼树表输出TreeListOut()编写时未分类讨论不同的结果及区分叶子节点和分叶子节点，导致输出错误。最后在添加多重限制条件后完成了对哈夫曼树的输出。此外在该函数中读到叶子节点后未将节点指针重新回到根节点，导致出现空指针，加一条回归根节点的代码后以解决。
2. 算法的时空分析
3. 由于本次使用操作对输入的需求较多而使用顺序表，更加节省时间复杂度顺序表元素添加函数ListAdd()，时间复杂度为O(1)，空间复杂度为O(1)。

删除操作的函数 ListDel()由于需要完成查找和元素移动，时空复杂度较高，时间复杂度为O(n),空间复杂度为O(1)。

遍历线性表的函数 ListVisit()时间复杂度为O(n),空间复杂度为O(n)。

1. 由于哈夫曼编码需使用到二叉树，且需要多次进行排序和插入操作所以使用二叉链表的形式构建二叉树。函数TreeDataIn()将顺序表中的数据输入到二叉树中，时间复杂度为O(n)空间复杂度为O(n)。

排序函数TreeNodeSort()使用的是插入排序，时间复杂度为O(n2)，空间复杂度O(1)。

输出函数TreeListOut()用于将二叉树中所有节点的详细信息输出，时间复杂度O(n),未使用其他中间量，空间复杂度为O(1)。

创建哈夫曼树函数TreeCreat()用于创建哈夫曼树，时间复杂度为O(n2),空间复杂度为O(n2)。

哈夫曼编码和译码函数HafmDecode()、HafmEncode()需要在哈夫曼树中对每个输入的数据进行查找、译制时间复杂度较高，为O(n2)，空间复杂度为O(n)。

1. 函数EncodeFromfile()、DecodeFromfile()是从文件中读取数据进行编译码的操作，

时间复杂度为O(n3)，空间复杂度为O(n2)

函数GetElemFromfile()作用为从文件中读取字符和权值至顺序表中，时间复杂度为O(n)，空间复杂度为O(n)，

1. 经验和体会

本次实习作业采用数据抽象的程序设计方法，将程序划分为四个模块；顺序表、二叉树、文件操作模块和主控制模块。先将输入和读取的字符及其权值存入线性表，然后可以对其进行遍历，查找，删除的操作，以此来保证字符及其权值输入的准确性。确认无误后，可将各个字符及其权值导入二叉链表，构造哈夫曼树，以此获取哈夫曼编码并将编码保存到各个对应的叶子节点中。在编码和译码时根据叶子节点的哈夫曼值完成对应操作。通过这次学习我学习并掌握了哈夫曼树的构建以及文件的读取方法，同时还了解了不同的输入方法cin、cin.get()的区别。另外，还巩固了对于顺序表和链表的使用，包括插入、删除、遍历等基本操作，还有排序、查找等较为复杂的操作。

在此次实习中感觉学习到了很多，受益匪浅。深感得到了一次良好的程序设计训练。

## 五、测试

执行命令1

输入字符和权值

执行命令2

查看所有字符情况

执行命令3

删除字符

执行命令4

构造哈夫曼树

执行命令5

查看哈夫曼树

执行命令6

编码

执行命令7

译码

执行命令8

读取文件构造哈夫曼树

执行命令9

读取文件进行编码

执行命令10

读取文件进行译码

执行命令0

退出

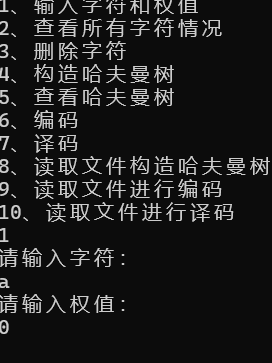


图5-1 输入字符及权值

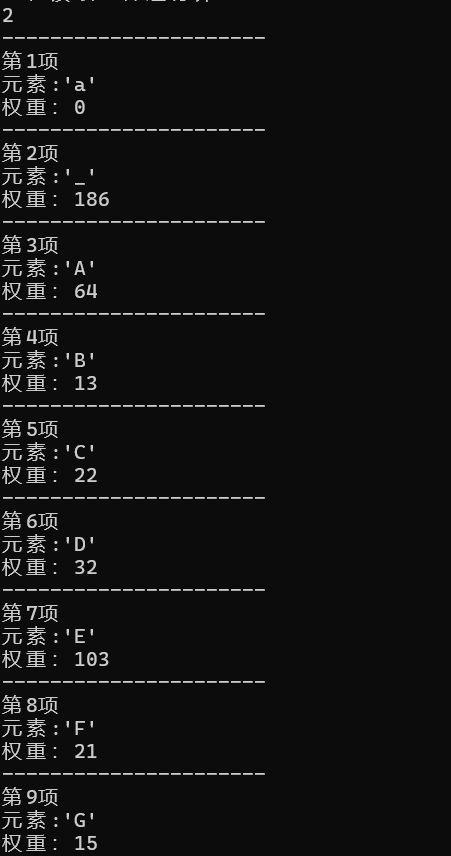


图5-2 输入查看输入的字符

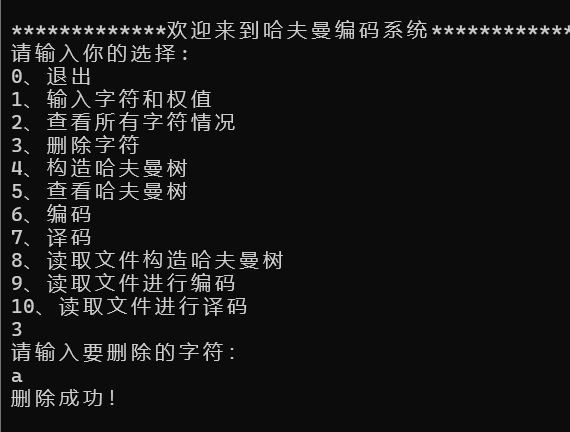


图5-3 删除字符

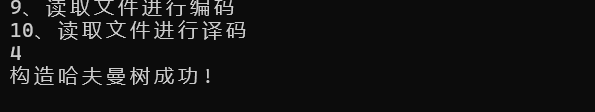


图5-4 构建哈夫曼树

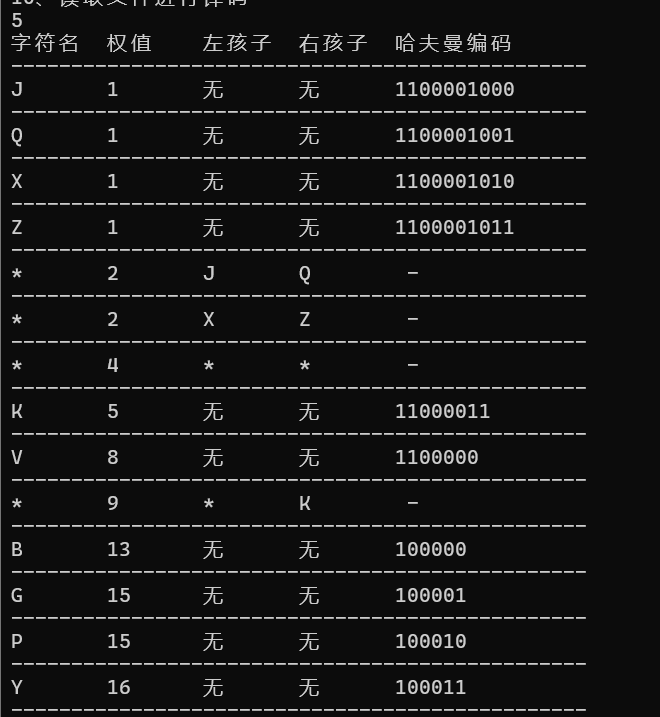


图5-5 查看哈夫曼树

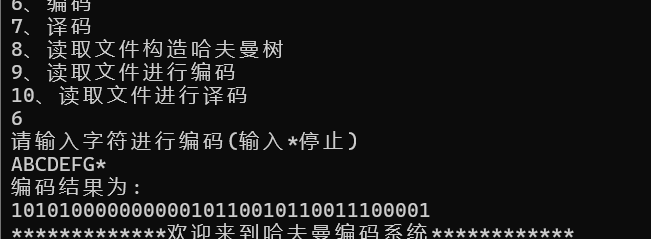


图5-6 编码

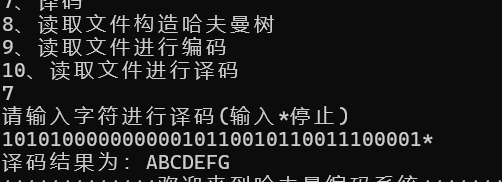


图5-7 译码

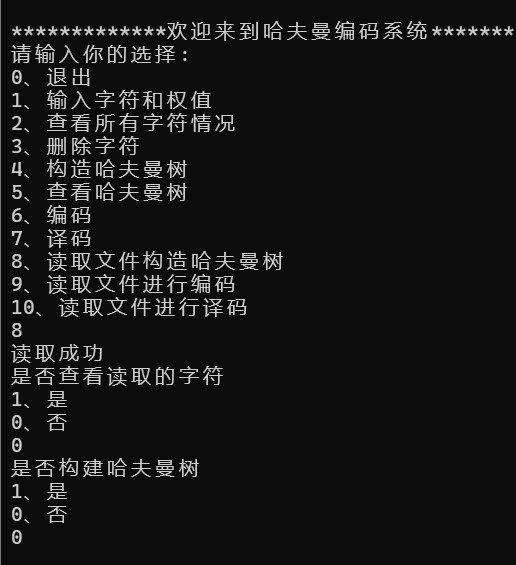


图5-8 读取本地字符

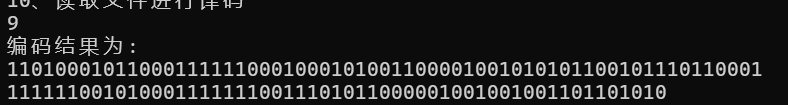


图5-9 输入字符及权值

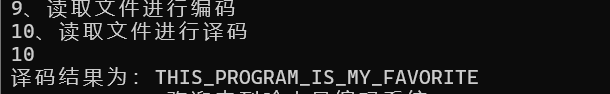


图5-10 输入字符及权值

## 六、附录 源程序文件名清单

BiTree.h //二叉树单元。

List.h //顺序表单元

file.h //文件控制单元

Decode.txt//译码文件

Encode.txt//编码文件

Tree.txt//字符文件

main.cpp //主程序