

数据结构课程设计

哈夫曼编码译码器

13 网络工程 1

李啸侠

1310322132

2014 年 12 月 14 日

目录

一、 问题描述.....	1
二、 需求分析.....	2
三、 概要设计.....	3
四、 调试分析.....	4
五、 使用说明.....	5
六、 测试结果.....	6
七、 附录（程序清单）	11

一、 问题描述

1. 题目内容

利用哈夫曼编码进行通信可以大大提高信道利用率，缩短信息传输时间，降低传输成本。

但是，这要求在发送端通过一个编码系统对待传数据预先编码，在接收端将传来的数据进行

译码（复原）。对于双工信道（即可以双向传输信息的信道），每端都需要一个完整的编/译

码系统。试为这样的信息收发站写一个哈夫曼码的编/译码系统。

2. 基本要求

一个完整的系统应具有以下功能：

- 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）。利用已建好的哈夫曼树（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: 打印代码文件（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: 打印哈夫曼树（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。

3. 测试数据

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 频度	空格	A	B	C	D	E	F	G	H	I	J	K	L	M
	186	64	13	22	32	103	21	15	47	57	1	5	32	20
字符 频度	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	
	57	63	15	1	48	51	80	23	8	18	1	16	1	

二、 需求分析

1. 程序所能达到的基本功能

读入字符集，建立哈夫曼树，利用哈夫曼树编码、译码，以直观的方式输出数据。

2. 输入的形式和输入值的范围

按照题目要求，输入的字符集和要编码的报文已给定。在程序中，若能打开对应的存放了字符集和报文的文件，则直接从文件中读入数据；若不能，则要求用户从控制台输入。

3. 输出的形式

按照题目要求，建立好的哈夫曼树、编码后的报文、译码后的结果、紧凑格式的编码以及直观方式表达的哈夫曼树都需要存入文件。

4. 测试数据要求

测试数据题目中已给出。

三、概要设计

1. 所需的数据结构，它们的作用

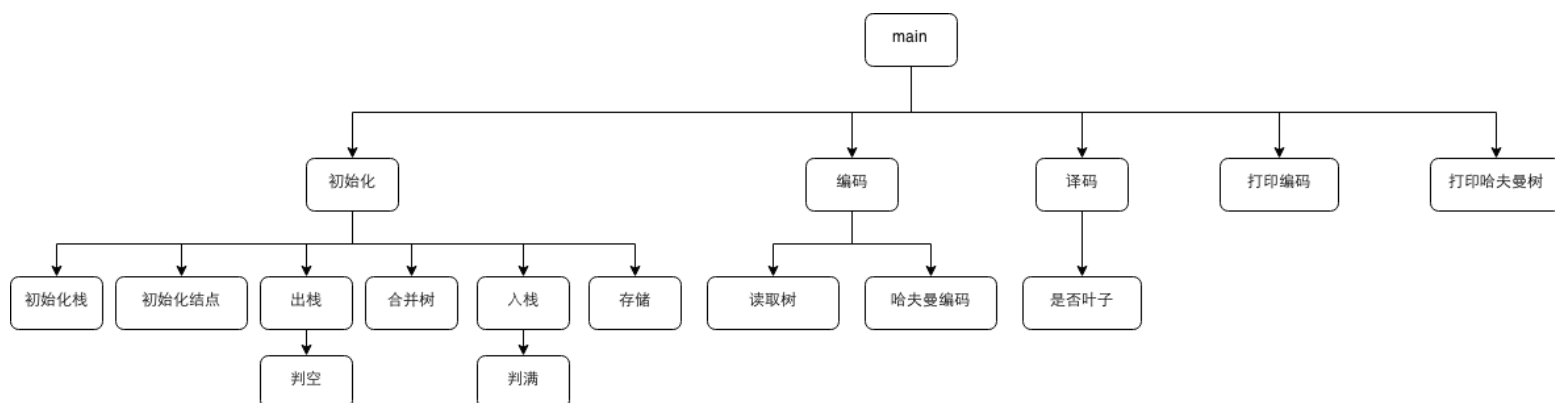
- 1) 树: 整个程序的重要数据结构, 建立哈夫曼树和编码译码的过程中所有字符以及对应的权值全都储存在树节点中。

```
typedef struct TNode
{
    int weight;
    char ch;
    char code[20];
    struct TNode *lchild, *rchild;
}TNode, *Tree;
```

- 2) 栈: 建立哈夫曼树是用栈来保证总是取最小的两个权值为根节点的树来组成一颗新的树。

```
typedef struct
{
    Tree data[MAX];
    int top;
}Stack;
```

2. 主程序流程及模块调用关系



四、 调试分析

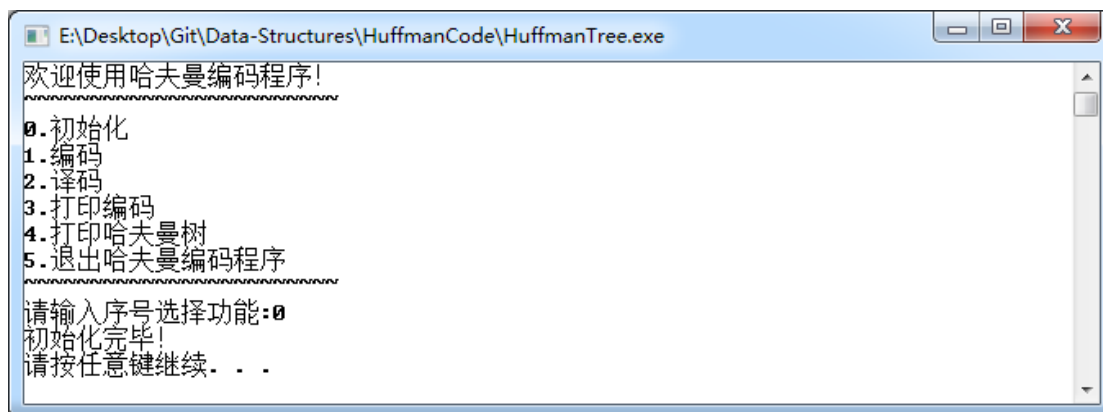
1. 调试过程中遇到的问题及分析、体会
2. 主要算法的时间复杂度分析

五、 使用说明

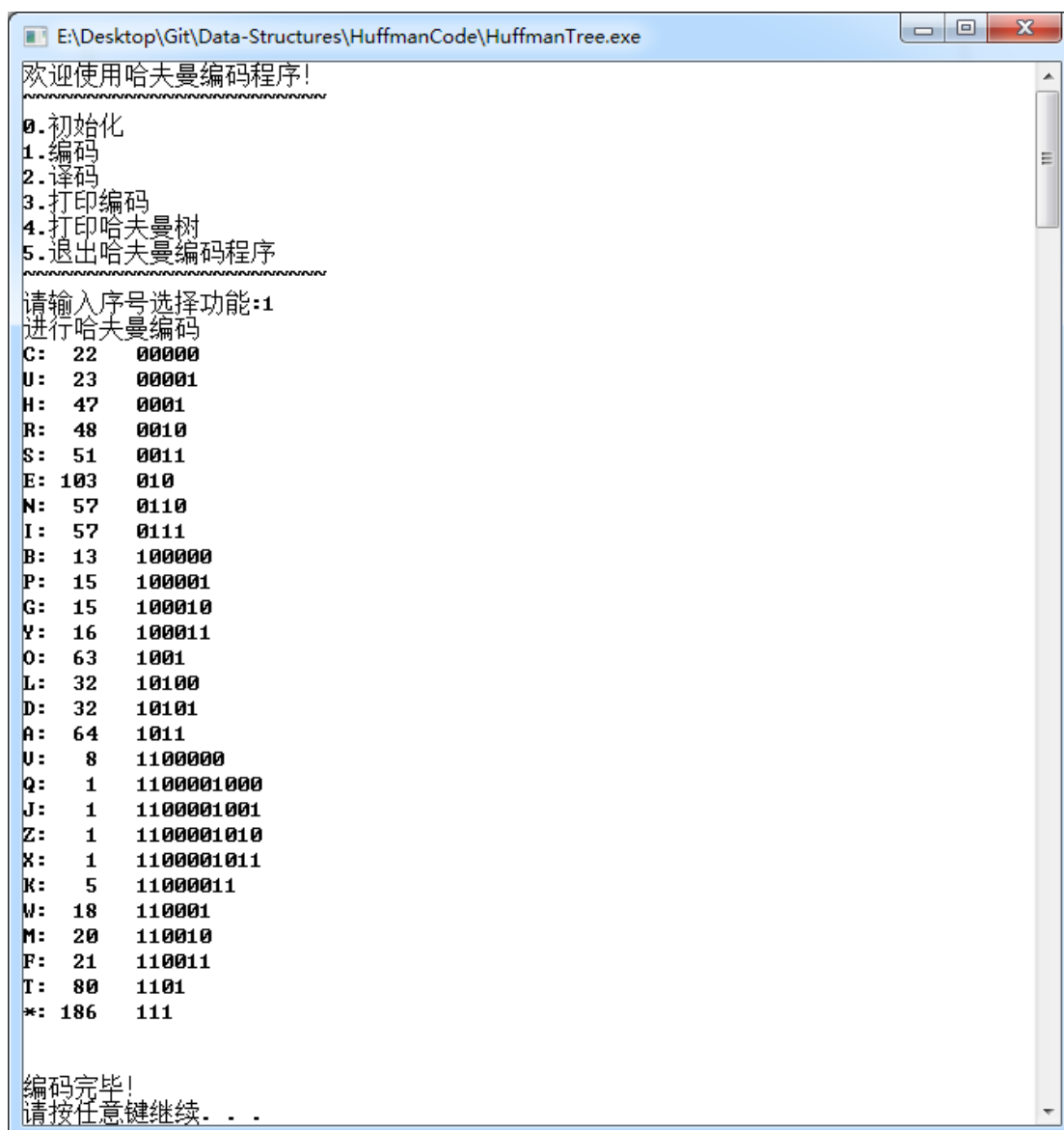
第一次运行，按照 0、1、2、3、4、5 的顺序输入，一次运行各个功能；第二次运行，由于目录下已存在储存了上次运行过程中建好的哈夫曼树的 `hfmTree.txt` 文件，直接输入 1，通过从文件中读取出的树进行哈夫曼编码；第三次运行，由于目录下已存在上次运行过程中保存的储存有编码的 `CodeFile.txt` 文件，直接输入 2，通过从文件中读取出的编码进行译码。

六、 测试结果

1. 第一次运行（按顺序实现功能）



```
E:\Desktop\Git\Data-Structures\HuffmanCode\HuffmanTree.exe
欢迎使用哈夫曼编码程序!
~~~~~
0. 初始化
1. 编码
2. 译码
3. 打印编码
4. 打印哈夫曼树
5. 退出哈夫曼编码程序
~~~~~
请输入序号选择功能:0
初始化完毕!
请按任意键继续. . .
```



```
E:\Desktop\Git\Data-Structures\HuffmanCode\HuffmanTree.exe
欢迎使用哈夫曼编码程序!
~~~~~
0. 初始化
1. 编码
2. 译码
3. 打印编码
4. 打印哈夫曼树
5. 退出哈夫曼编码程序
~~~~~
请输入序号选择功能:1
进行哈夫曼编码
C: 22 00000
U: 23 00001
H: 47 0001
R: 48 0010
S: 51 0011
E: 103 010
N: 57 0110
I: 57 0111
B: 13 100000
P: 15 100001
G: 15 100010
Y: 16 100011
O: 63 1001
L: 32 10100
D: 32 10101
A: 64 1011
V: 8 1100000
Q: 1 1100001000
J: 1 1100001001
Z: 1 1100001010
X: 1 1100001011
K: 5 11000011
W: 18 110001
M: 20 110010
F: 21 110011
T: 80 1101
*: 186 111

编码完毕!
请按任意键继续. . .
```



```

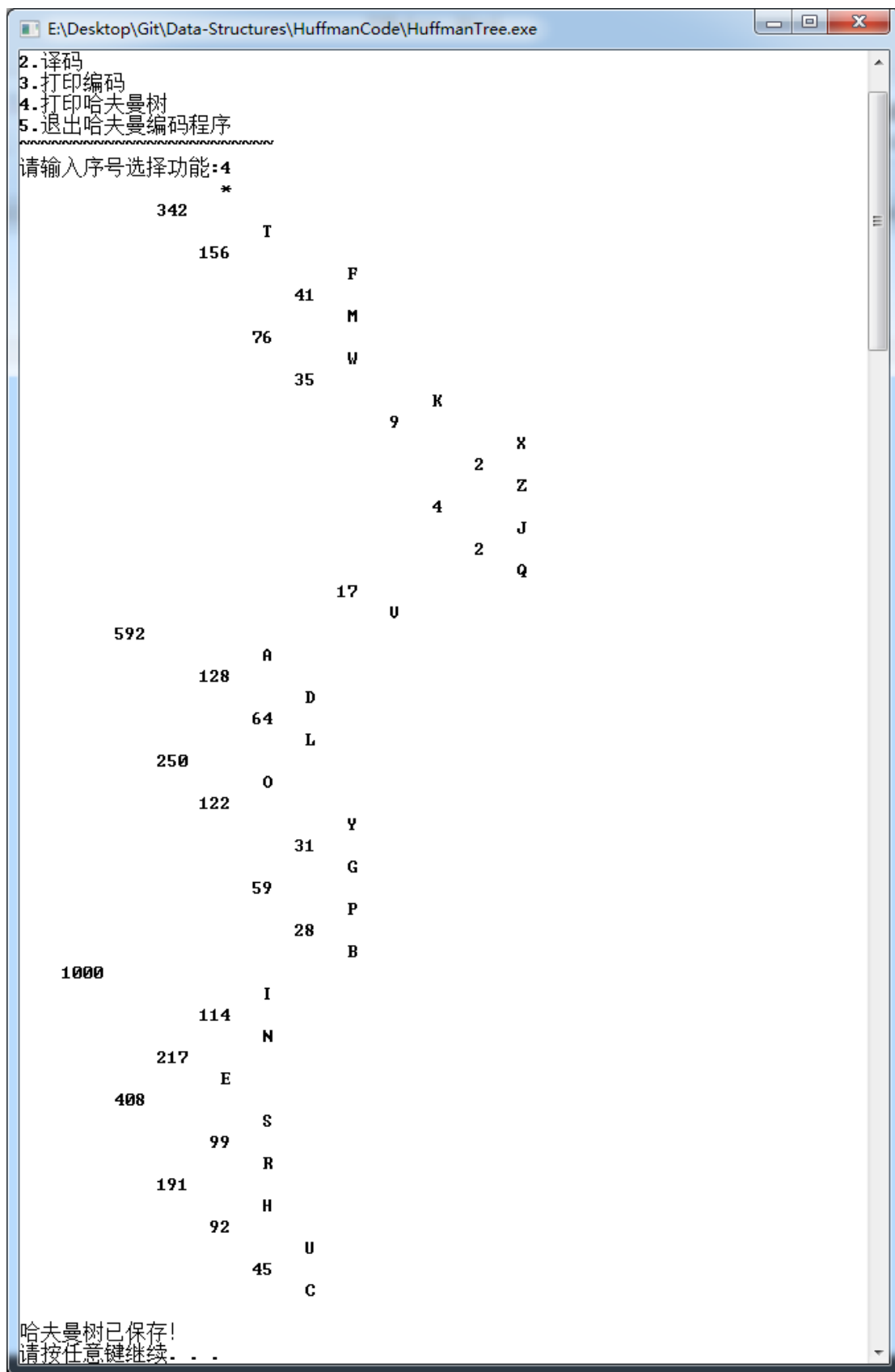
E:\Desktop\Git\Data-Structures\HuffmanCode\HuffmanTree.exe
欢迎使用哈夫曼编码程序!
~~~~~
0. 初始化
1. 编码
2. 译码
3. 打印编码
4. 打印哈夫曼树
5. 退出哈夫曼编码程序
~~~~~
请输入序号选择功能:2
THIS PROGRAM IS MY FAVORITE
译码结果已保存
请按任意键继续. . .

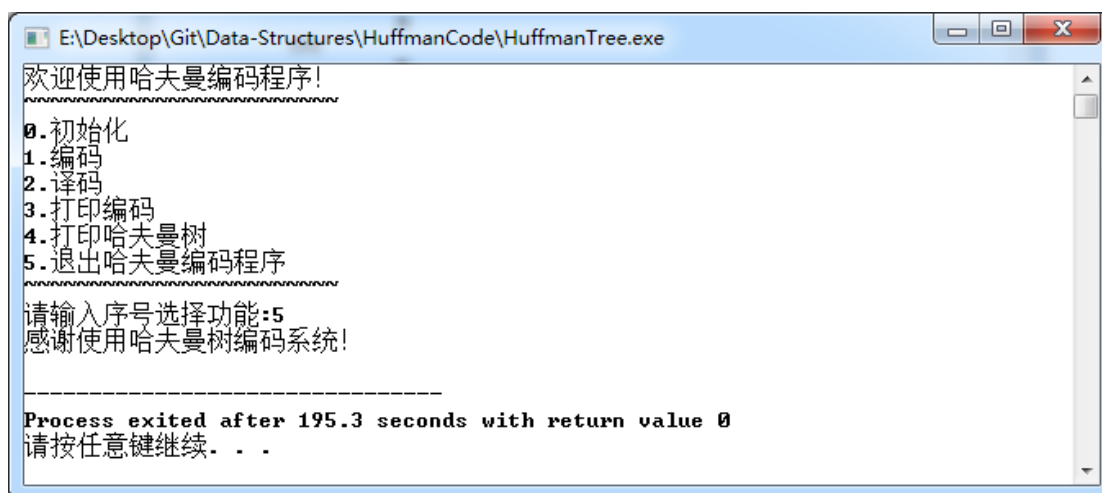
```

```

E:\Desktop\Git\Data-Structures\HuffmanCode\HuffmanTree.exe
欢迎使用哈夫曼编码程序!
~~~~~
0. 初始化
1. 编码
2. 译码
3. 打印编码
4. 打印哈夫曼树
5. 退出哈夫曼编码程序
~~~~~
请输入序号选择功能:3
1101000101110011111000010010100110001000101011110
010111011100111111001010001111110011101111000001
001001001111101010
打印编码完毕!
请按任意键继续. . .

```





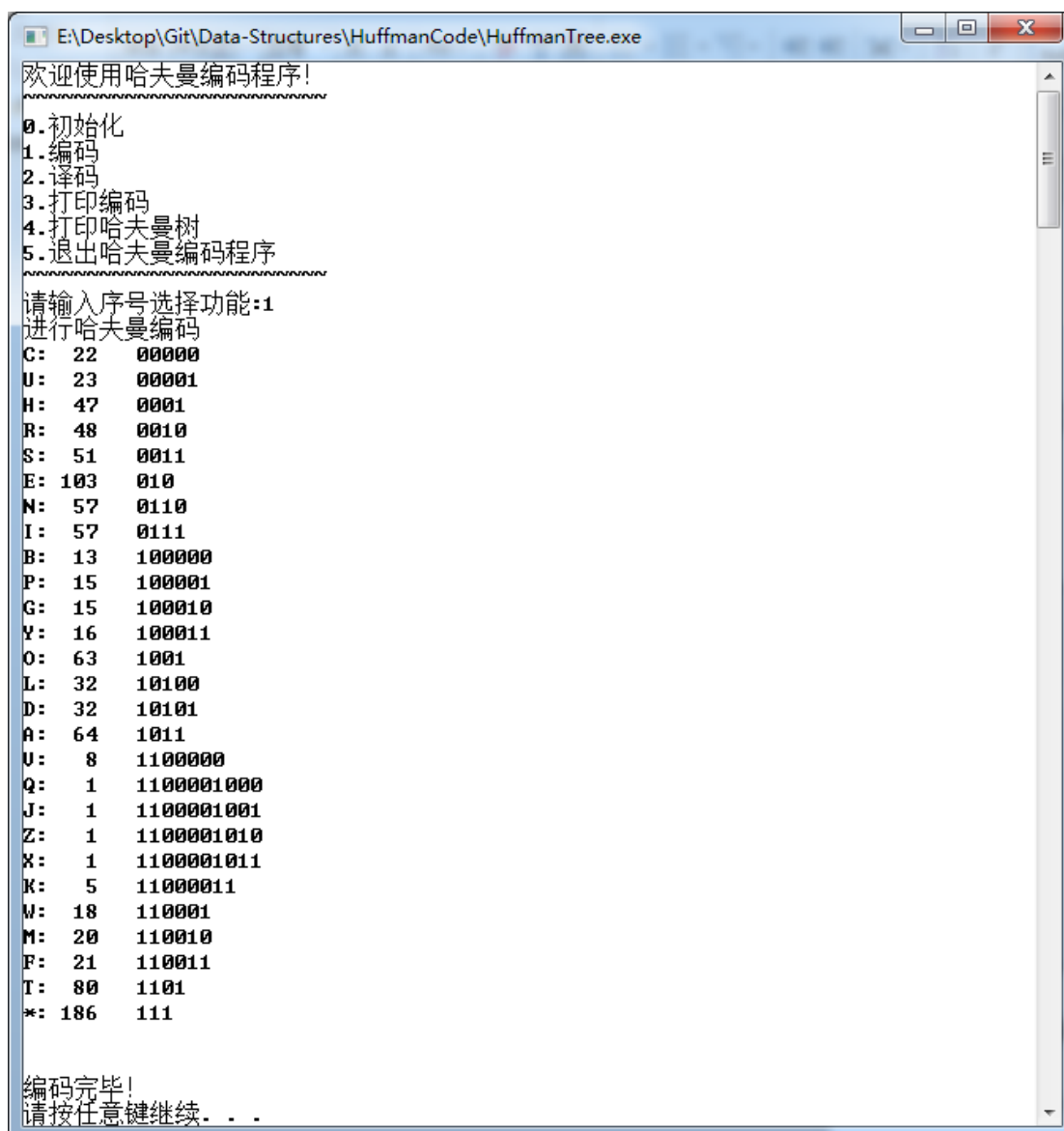
```

E:\Desktop\Git\Data-Structures\HuffmanCode\HuffmanTree.exe
欢迎使用哈夫曼编码程序!
~~~~~
0. 初始化
1. 编码
2. 译码
3. 打印编码
4. 打印哈夫曼树
5. 退出哈夫曼编码程序
~~~~~
请输入序号选择功能:5
感谢使用哈夫曼树编码系统!

-----
Process exited after 195.3 seconds with return value 0
请按任意键继续. . .

```

2. 第二次运行（从文件中读取哈夫曼树进行编码）



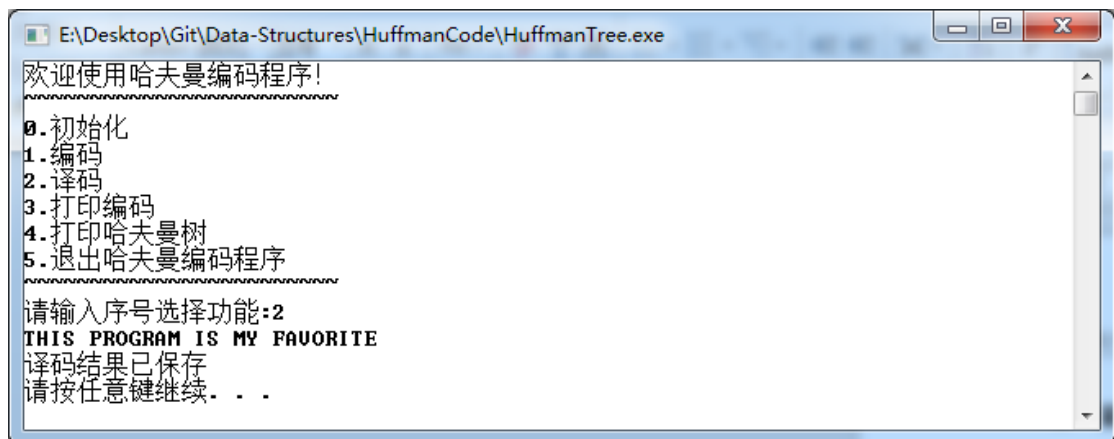
```

E:\Desktop\Git\Data-Structures\HuffmanCode\HuffmanTree.exe
欢迎使用哈夫曼编码程序!
~~~~~
0. 初始化
1. 编码
2. 译码
3. 打印编码
4. 打印哈夫曼树
5. 退出哈夫曼编码程序
~~~~~
请输入序号选择功能:1
进行哈夫曼编码
C: 22 00000
U: 23 00001
H: 47 0001
R: 48 0010
S: 51 0011
E: 103 010
N: 57 0110
I: 57 0111
B: 13 100000
P: 15 100001
G: 15 100010
Y: 16 100011
O: 63 1001
L: 32 10100
D: 32 10101
A: 64 1011
V: 8 1100000
Q: 1 1100001000
J: 1 1100001001
Z: 1 1100001010
X: 1 1100001011
K: 5 11000011
W: 18 110001
M: 20 110010
F: 21 110011
T: 80 1101
*: 186 111

编码完毕!
请按任意键继续. . .

```

3. 第三次运行（从文件中读取编码进行译码）



七、 附录（程序清单）

Stack.h

```
#define MAX 50

typedef struct TNode
{
    int weight;
    char ch;
    char code[20]; //每个结点自带一个字符数组存储当前结点的哈夫曼编码
    struct TNode *lchild, *rchild;
}TNode, *Tree;

typedef struct
{
    Tree data[MAX];
    int top;
}Stack;

int InitStack(Stack *S) //初始化栈
{
    (*S).top = -1;
    return 0;
}

int IsEmpty(Stack S) //判空
{
    return -1 == S.top;
}

int IsFull(Stack S) //判满
{
    return MAX - 1 == S.top;
}

int Push(Stack *S, Tree e) //入栈
{
    if(IsFull(*S))
        return -1;
    (*S).top += 1;
    (*S).data[(*S).top] = e;
    return 0;
}
```

```
int Pop(Stack *S, Tree *e)      //出栈
{
    if(IsEmpty(*S))
        return -1;
    *e = (*S).data[(*S).top];
    (*S).top -= 1;
    return 0;
}
```

HuffmanCode.c

```
//Author: Double X Li
```

```
//Date: 2014-12-9
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include "Stack.h"
```

```
FILE *fp;
```

```
char save[27][15]; //储存27个字符的哈夫曼编码
```

```
char codestr[500]; //储存编码后的报文
```

```
//判断是否为叶子
```

```
int IsLeaf(Tree T)
{
    if(T->lchild == NULL && T->rchild == NULL)
        return 1;
    else
        return 0;
}
```

```
//初始化树结点
```

```
void InitNode(Tree *t, int x, char y)
{
    *t = (TNode *)malloc(sizeof(TNode));
    (*t)->weight = x;
    (*t)->ch = y;
    (*t)->lchild = NULL;
    (*t)->rchild = NULL;
}
```

```
//合并树
void Make(Tree one, Tree two, Tree *templ)
{
    (*templ) = (TNode *)malloc(sizeof(TNode));
    (*templ)->weight = one->weight + two->weight;
    (*templ)->ch = '#';
    (*templ)->lchild = one;
    (*templ)->rchild = two;
}

//从文件中读取树
void ReadTree(Tree *T)
{
    int flag;
    *T = (TNode *)malloc(sizeof(TNode));
    fscanf(fp, "%c %d %d\n", &((*T)->ch), &((*T)->weight), &flag);

    if(!flag)
    {
        ReadTree(&((*T)->lchild));
        ReadTree(&((*T)->rchild));
    }
}

//将树保存到文件中
void SaveTree(Tree T)
{
    fprintf(fp, "%c %d %d\n", T->ch, T->weight, IsLeaf(T));
    if(!IsLeaf(T))
    {
        SaveTree(T->lchild);
        SaveTree(T->rchild);
    }
}

//初始化
Tree Init()
{
    int i, j, temp;
    char c;
    int n = 27;
    int flag = 1;
    int weight[27]; //储存读入的权重
    char ch[27]; //储存读入的字符
```

```
Stack S, K;
Tree x;
Tree one, two;
Tree temp1, temp2;

//准备好两个栈
InitStack(&S);
InitStack(&K);

if((fp = fopen("character.txt","r")) != NULL) //有字符集则读入
{
    for(i = 0; i < n; i++)
    {
        fscanf(fp,"%c %d\n",&ch[i],&weight[i]);
    }
    fclose(fp);
}
else //无则输入并存储
{
    for(i = 0; i < n; i++)
    {
        printf("请输入第%d个字母:\n",i + 1);
        if(( ch[i] = getchar() ) == ' ' )
            ch[i] = '*';
        getchar();
        printf("及它的权值:\n");
        scanf("%d",&weight[i]);
        getchar();
    }
    fp = fopen("character.txt","w");
    for(i = 0; i < n; i++)
    {
        fprintf(fp,"%c %d\n",ch[i],weight[i]);
    }
    fclose(fp);
}

//对字符按权值排序
for(i = 0;i < n; i++)
{
    for(j = 1; j < n - i; j++)
    {
        if(weight[j-1] < weight[j])
        {
```



```

        temp = weight[j];
        c = ch[j];
        weight[j] = weight[j-1];
        ch[j] = ch[j-1];
        weight[j-1] = temp;
        ch[j-1] = c;
    }
}

//将每个字符及其权值构造成一棵树，并入栈
for(i = 0; i < 27; i++)
{
    InitNode(&x,weight[i],ch[i]);
    Push(&S,x);
}

//构造哈夫曼树
while(flag)
{
    //去最小的两个权值
    Pop(&S, &one);
    Pop(&S,&two);
    //构造一颗新树
    Make(one, two, &temp1);
    if(IsEmpty(S))
        break;
    temp2 = (TNode *)malloc(sizeof(TNode));
    Pop(&S,&temp2);
    while(temp1->weight > temp2->weight && !IsEmpty(S))
    {
        Push(&K,temp2);
        Pop(&S,&temp2);
    }
    if(temp1->weight > temp2->weight)
    {
        Push(&S,temp1);
        Push(&S,temp2);
    }
    else
    {
        Push(&S,temp2);
        Push(&S,temp1);
    }
}

```

```

        while(!IsEmpty(K))
        {
            Pop(&K,&temp2);
            Push(&S,temp2);
        }
        if(temp1->weight == 1000)
            flag = 0;
    }
    fp = fopen("hfmTree.txt","w");
    SaveTree(temp1);
    fclose(fp);
    printf("初始化完毕! \n");
    return temp1;
}

Tree HuffmanCoding(Tree T)
{
    //遍历哈夫曼树的过程中进行编码
    if(NULL != T)
    {
        //当前为叶子—需编码的字符所在结点
        if(IsLeaf(T))
        {
            printf("%c:%4d  %s\n",T->ch,T->weight,T->code);
            if(isalpha(T->ch))
            {
                strcpy(save[(int)(T->ch - 64)],T->code);
            }
            else if('*' == T->ch)
            {
                strcpy(save[0],T->code);
            }
        }
        //当前为非叶子
        if(T->lchild) //左孩子编码添加0
        {
            strcpy(T->lchild->code,T->code);
            strcat(T->lchild->code,"0");
        }
        if(T->rchild) //右孩子编码添加1
        {
            strcpy(T->rchild->code,T->code);
            strcat(T->rchild->code,"1");
        }
    }
}

```

```

    }
    HuffmanCoding(T->lchild);
    HuffmanCoding(T->rchild);
}
return T;
}

//编码
Tree Encoding(Tree T)
{
    int i = 0;
    char str[MAX];

    if(T == NULL)
    {
        fp = fopen("hfmTree.txt", "r");
        ReadTree(&T);
        fclose(fp);
    }
    printf("进行哈夫曼编码\n");
    T = HuffmanCoding(T);
    //读入报文
    printf("\n");
    if((fp = fopen("ToBeTran.txt", "r")) != NULL)
    {
        fgets(str, MAX, fp);
        fclose(fp);
    }
    else
    {
        printf("请输入要编码的文件: \n");
        gets(str);
        fp = fopen("ToBeTran.txt", "w");
        fprintf(fp, "%s", str);
        fclose(fp);
    }
    //逐个字符查找对应编码并存入codestr
    while(str[i] != '\0')
    {
        if(isalpha(str[i]))
        {
            if(i == 0)
                strcpy(codestr, save[(int)(str[i] - 64)]);
            else

```

```
        strcat(codestr, save[(int)(str[i] - 64)]);
    }
    else if(' ' == str[i])
    {
        if(i == 0)
            strcpy(codestr, save[0]);
        else
            strcat(codestr, save[0]);
    }
    i++;
}
if((fp = fopen("CodeFile.txt", "w")) == NULL)
{
    printf("编码不能写入文件! ");
}
else
{
    fprintf(fp, "%s", codestr);
    fclose(fp);
}
printf("\n编码完毕! \n");
return T;
}

//译码
void Decoding(Tree T)
{
    char ch;
    Tree temp = T;
    FILE *fq;

    fq = fopen("TextFile.txt", "w");
    //读入编码, 通过哈夫曼树译码
    if((fp = fopen("CodeFile.txt", "r")) == NULL)
    {
        printf("无法译码! \n");
        fclose(fp);
        return;
    }
    else
    {
        while(1)
        {
            while(!IsLeaf(temp))
```

```

    {
        if((ch = fgetc(fp)) == EOF)
        {
            printf("\n");
            fclose(fp);
            fclose(fq);
            printf("译码结果已保存\n");
            return;
        }
        if(ch == '0')
            temp = temp->lchild;
        if(ch == '1')
            temp = temp->rchild;
    }
    if(temp->ch == '*')
        temp->ch = ' ';
    printf("%c",temp->ch);
    fprintf(fq,"%c",temp->ch);
    temp = T;
}
}
}

```

//按格式打印编码并保存

```

void PrintCode()
{
    int i;

    fp = fopen("CodePrint.txt","w");

    for(i = 0; codestr[i] != '\0'; i++)
    {
        if( i != 0 && i % 50 == 0)
        {
            printf("\n");
            fprintf(fp,"%c",'\\n');
        }
        printf("%c",codestr[i]);
        fprintf(fp,"%c",codestr[i]);
    }
    fclose(fp);
    printf("\n\n打印编码完毕! \n");
}

```

//按树形打印哈夫曼树并保存

```
void PrintTree(Tree T, int n)
{
    int i;
    if(T == NULL)
        return;
    PrintTree(T->rchild,n+1);
    for(i = 0; i < n; i++)
    {
        printf("    ");
        fprintf(fp,"%s", "    ");
    }
    if(T->ch == '#')
    {
        printf("%4d\n",T->weight);
        fprintf(fp,"%4d\n",T->weight);
    }
    else if(T->ch == ' ')
    {
        printf("%4c\n",'*');
        fprintf(fp,"%4c\n",'*');
    }
    else
    {
        printf("%4c\n",T->ch);
        fprintf(fp,"%4c\n",T->ch);
    }
    PrintTree(T->lchild,n+1);
}

int main(void)
{
    int x;
    Tree T = NULL;

    system("color F0");
    Begin:
    printf("欢迎使用哈夫曼编码程序! \n");
    printf("~~~~~\n");
    printf("0.初始化\n");
    printf("1.编码\n");
    printf("2.译码\n");
    printf("3.打印编码\n");
    printf("4.打印哈夫曼树\n");
```

```

printf("5.退出哈夫曼编码程序\n");
printf("~~~~~\n");
printf("请输入序号选择功能:");
scanf("%d",&x);
getchar();
switch(x)
{
    case 0: T = Init();
            system("pause");
            system("cls");
            goto Begin;//输入字符及权值，建立哈夫曼树并储存
    case 1: T = Encoding(T);
            system("pause");
            system("cls");
            goto Begin;//进行哈夫曼编码
    case 2: Decoding(T);
            system("pause");
            system("cls");
            goto Begin;//进行哈夫曼译码
    case 3: PrintCode();
            system("pause");
            system("cls");//按格式打印编码并保存
            goto Begin;
    case 4: fp = fopen("TreePrint.txt","w");
            PrintTree(T,1);
            fclose(fp);
            printf("\n哈夫曼树已保存! \n");
            system("pause");
            system("cls");
            goto Begin;//按树形打印哈夫曼树并保存
    case 5: printf("感谢使用哈夫曼树编码系统! \n");
            return 0;//退出
    default: printf("There must be something wrong, please
check!\n");
            system("pause");
            system("cls");
            goto Begin;//输入有误，重新输入
}
}

```