|  |  |  |  |
| --- | --- | --- | --- |
| **学生学号** | 1750 | **实验课成绩** |  |

*河北工业大学*

**学 生 实 验 报 告 书**

|  |  |
| --- | --- |
| **实验课程名称** | 数据结构 |
| **开 课 学 院** | 国际教育学院 |
| **指导教师姓名** | 宛艳萍 |
| **学 生 姓 名** | 张心宇 |
| **学生专业班级** | 中法计172 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2018 | -- | 2019 | 学年 | 第 | 二 | 学期 |

# 实验内容

**基于哈夫曼（Huffmen）编码的通信系统的设计与实现**

【问题描述】

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输数据预先编码，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的编/译码系统。

【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）。利用已经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写入文件CodePrint中。
* （5） T：打印哈夫曼树（Tree printing）。将已经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。

【测试数据】

（1） 利用教科书例6-2中的数据调试程序。

（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 |

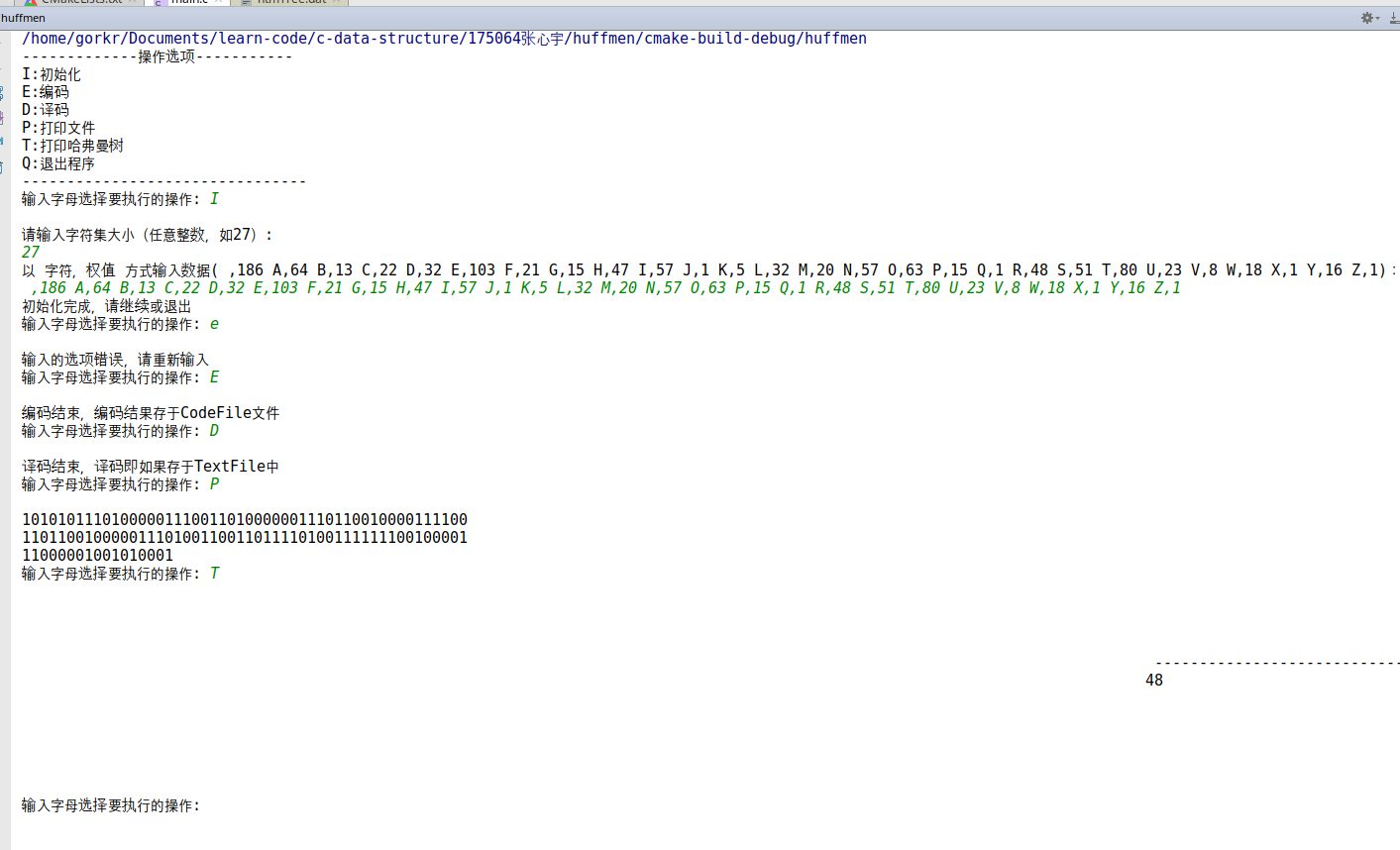
# 实验目的

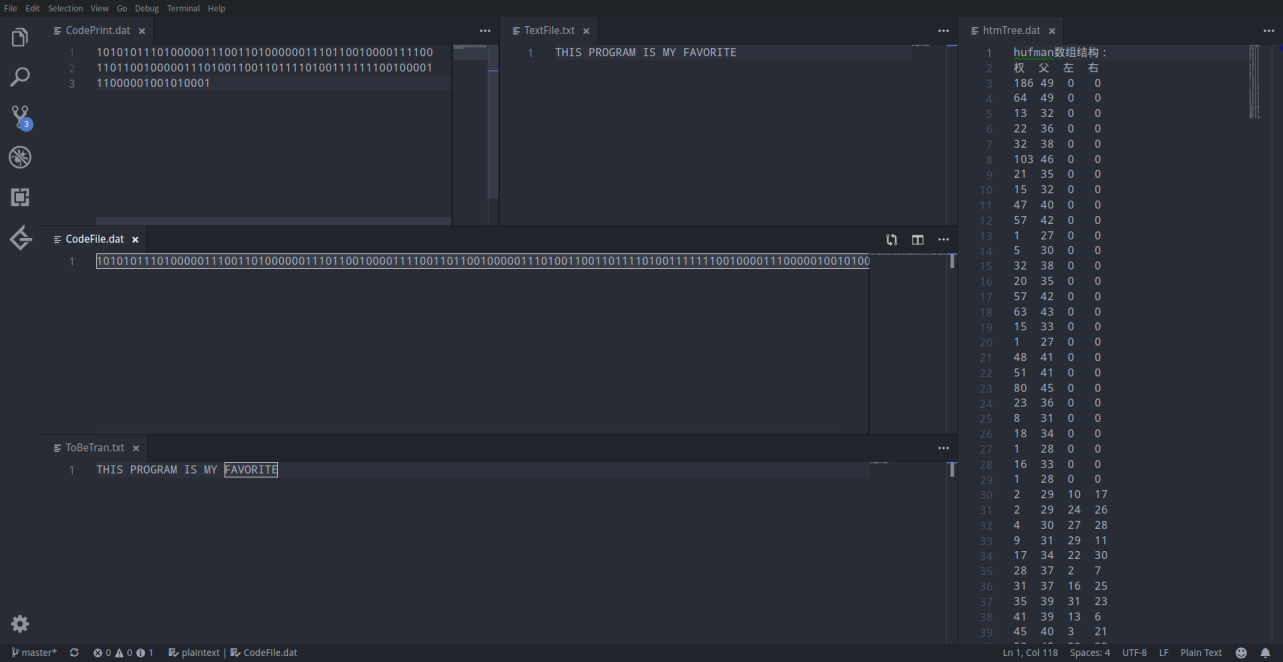
* （1） 掌握二叉树的存储结构及其相关操作。
* （2） 掌握构造哈夫曼树的基本思想，及其编码/译码过程。

# 程序清单

#include<stdio.h>  
#include<stdlib.h>  
#include<string.h>  
  
// 字符及其权值表  
typedef struct{  
 char c;  
 unsigned int weight;  
}CharNode,\* CharMap;  
  
// huffman 数组  
//- 这里应该让HuffmanTree类型独立出来，  
//- 让HTNode数组成为 HuffmanTree类型的一个元素。  
//- 同时让 len 属性独立出来  
typedef struct{  
 int len; // 叶子数  
 char c;  
 unsigned int weight;  
 unsigned int parent, lchild, rchild;  
 char \* code; // 叶子对应的编码  
}HTNode, \* HuffmanTree;  
  
HuffmanTree Initalization(); //  
  
  
  
// 将内存中的hftree写入文件hfmTree中  
void WriteHfm(HuffmanTree hf){  
 FILE \* fp;  
 fp = fopen("htmTree.dat", "w");  
  
  
 fprintf(fp, "hufman数组结构：\n权\t父\t左\t右\n");  
 HuffmanTree p =hf;  
 while(p->parent!=0){  
 fprintf(fp,"%d\t%d\t%d\t%d\n",p->weight,p->parent,p->lchild,p->rchild);  
 p++;  
 }  
 fprintf(fp,"%d\t%d\t%d\t%d\n",p->weight,p->parent,p->lchild,p->rchild);  
  
 fclose(fp);  
 //if(fclose(fp)!=0)printf("Error with closing");  
}  
  
//- 应该边输入边初始化hf，不必使用map字符表  
HuffmanTree Initalization(){  
 // 构建字符表  
 unsigned int map\_len;  
 printf("请输入字符集大小（任意整数，如27）:\n");  
 scanf("%d", &map\_len);getchar(); // getchar()读取回车  
 CharMap map = (CharMap)malloc(map\_len\*sizeof(CharNode));  
 printf("以 字符，权值 方式输入数据( ,186 A,64 B,13 C,22 D,32 E,103 F,21 G,15 H,47 I,57 J,1 K,5 L,32 M,20 N,57 O,63 P,15 Q,1 R,48 S,51 T,80 U,23 V,8 W,18 X,1 Y,16 Z,1)：\n");  
 for(int i=0;i<map\_len;i++){  
 scanf("%c,%d",&map[i].c, &map[i].weight);getchar();  
 }  
  
 // 创建huffman树  
 int m = 2\*map\_len-1;  
 HuffmanTree hf = (HuffmanTree)malloc(m\*sizeof(HTNode));  
 hf->len = map\_len;  
 unsigned i=0;HTNode\*p=hf;  
 for(;i<map\_len;++i,++p){ // 赋值叶子节点  
 p->c = map[i].c;  
 p->weight = map[i].weight;  
 p->parent=p->rchild=p->rchild=0;  
 p->code = NULL;  
 }  
 for(;i<m-1;i++,++p){ // 赋值中间节点  
 p->c='\0';  
 p->weight=p->parent=p->rchild=p->rchild=0;  
 }  
  
 for(i=map\_len;i<m;++i){ // 建huffman树  
 // 取得最小的两个权值的下标  
 unsigned int min[2] = {0,0}; // 存储最小的两个权值  
 int flag0=1, flag1=1; // 最小值未被初始化  
 for(unsigned int j=0;j<i;j++){  
 if(hf[j].parent==0){  
 if(flag0){min[0]=j;flag0--;}  
 else if(flag1){min[1]=j;flag1--;  
 if(min[0]>min[1]){  
 unsigned int tmp = min[0];  
 min[0] = min[1];  
 min[1] = tmp;  
 }  
 }  
 else {  
 if(hf[j].weight < hf[min[0]].weight){min[1] = min[0]; min[0] = j;}  
 else if (hf[j].weight < hf[min[1]].weight)min[1] = j;  
 }  
 }  
 }  
  
 hf[min[0]].parent = hf[min[1]].parent = i;  
 hf[i].lchild = min[0];  
 hf[i].rchild = min[1];  
 hf[i].weight = hf[min[0]].weight + hf[min[1]].weight;  
 }  
 free(map);  
 //WriteHfm(hf);  
 //? 为什么这条语句会影响到encoding 函数的写  
 return hf;  
}  
  
void Encoding(HuffmanTree hf){  
  
 // 字符对应编码  
 char \* code = (char \*)malloc(hf->len\*sizeof(char));  
 if(!code)exit(-1);  
  
 for(int i=0;i<hf->len;++i){ // 从叶子节点，向上回溯  
 int start = hf->len-1;  
 //? ~~code字符串为什么无法更改字符~~, 不是无法更改,是前面是空白,认定为空字符串,无法输出,也无法执行其他操作  
 code[hf->len]='\0';  
 int j =i;  
 for(int p=hf[j].parent; p!=0;j=p,p=hf[p].parent){  
 if(hf[p].lchild==j){  
 code[--start] = '0';  
 }  
 else {code[--start] = '1';  
 }  
 }  
 hf[i].code = (char\*)malloc((hf->len-start)\*sizeof(char));  
  
 //e strcpy(hf[i].code, code); 这样只会把空串复制过去  
 for(int k=0;start<hf->len;start++,k++){  
 hf[i].code[k]=code[start];  
 }  
 }  
 free(code);  
  
 // 读取文件，并进行编码  
 FILE \* fp\_r, \* fp\_w;  
 fp\_r = fopen("ToBeTran.txt", "r");  
 fp\_w = fopen("CodeFile.dat", "w");  
 int ch;  
 ch = getc(fp\_r);  
 while(ch != EOF){  
 HuffmanTree p = hf;  
 for(int i = 0; i<hf->len; i++,p++){  
 if(p->c==ch)break;  
 }  
 ch = getc(fp\_r);  
 fputs(p->code, fp\_w);  
 }  
 fclose(fp\_r);fclose(fp\_w);  
}  
  
void Decoding(HuffmanTree hf){  
 FILE \* fp\_r, \* fp\_w;  
 fp\_r = fopen("CodeFile.dat", "r");  
 fp\_w = fopen("TextFile.txt", "w");  
 HTNode root = hf[2\*hf->len-2]; // 根节点  
 HTNode p =root;  
 int ch;  
 ch = getc(fp\_r);  
 while(ch!=EOF){  
 if(p.code==NULL){  
 // p节点不是叶子节点  
 if(ch=='0')p=hf[p.lchild];  
 else p=hf[p.rchild];  
 ch = getc(fp\_r);  
 }else{  
 // p节点是叶子 节点  
 fputc(p.c, fp\_w);  
 p = root;  
 }  
  
 }  
 fputc(p.c, fp\_w);  
 fclose(fp\_r);fclose(fp\_w);  
}  
  
/\*\*  
 \* @param root 树的根节点  
 \* @param hf huffman树  
 \* @return 树的深度  
 \*/  
int maxDepth(HTNode root, HuffmanTree hf){  
 if(root.rchild==0&&root.lchild==0)  
 return 1;  
 int ldepth = maxDepth(hf[root.lchild], hf)+1;  
 int rdepth = maxDepth(hf[root.rchild], hf)+1;  
 return ldepth>rdepth?ldepth:rdepth;  
}  
/\*\*  
 \* huffman树节点表  
 \* @param map 满二叉树对应的表， 空位代表不存在的节点  
 \*/  
void fillMap(HTNode \*\*map, HTNode \* n,int index,HuffmanTree hf){  
 int i;  
 map[index] = n;  
 if(!(n->lchild==0&&n->rchild==0)){  
 fillMap(map, &hf[n->lchild], index\*2+1, hf);  
 fillMap(map, &hf[n->rchild], index\*2+2, hf);  
 }  
}  
// 打印n个c  
int putchars(char c, int n){  
 int value =n;  
 while(n--)putchar(c);  
 return value;  
}  
// 打印节点  
int printNode(HTNode \*n, int w){  
 int num;  
 if(n->rchild==0&&n->lchild==0){  
 return printf("%\*d", w, n->weight);  
 }else{  
 num = putchars(' ', w/2-1)+putchars('-', w/2-2)+printf("%4d",n->weight)+putchars('-', w/2-2);  
 return num;  
 }  
}  
  
// 将huffman树以ascii形式打印在终端  
void TreePrint(HuffmanTree hf){  
 HTNode \*root = &hf[hf->len\*2-2];  
 int depth = maxDepth(\*root, hf), i, j, index;  
  
 HTNode \*\*map = (HTNode\*\*)calloc((1<<depth)-1,sizeof(HTNode\*)); // depth深度的满二叉树所有节点个数为（2的depth次方-1）  
 fillMap(map, root, 0, hf);  
 for(j=0, index=0; j<depth; j++){  
 int w =1 << (depth -j);  
 for(i=0;i<1<<j;i++,index++){  
 if(map[index]) putchars(' ', w\*2-printNode(map[index], w));  
 else putchars(' ', w\*2);  
 }  
 putchar('\n');  
 }  
  
 free(map);  
}  
  
void Print(){  
 FILE \* f = fopen("CodeFile.dat", "r");  
 FILE \* in = fopen("CodePrint.dat", "w");  
 int i = 1;  
 int ch;  
  
 // 将文件f内容输出  
 while((ch=getc(f))!= EOF){  
 putchar(ch);  
 putc(ch, in);  
 if(i%50==0){  
 putchar('\n');  
 putc('\n', in);  
 }  
 i++;  
 }  
 fclose(f);fclose(in);  
}  
  
  
  
void interface(HuffmanTree hf){  
 char c;  
 printf("-------------操作选项-----------\n");  
 printf("I:初始化\nE:编码\nD:译码\nP:打印文件\nT:打印哈弗曼树\n");  
 printf("Q:退出程序 \n");  
 printf("--------------------------------\n");  
 while (1)  
 {  
 printf("输入字母选择要执行的操作: ");  
 scanf(" %c",&c);  
 printf("\n");  
  
 //跳出循环，退出程序  
 if(c=='Q')  
 break;  
  
 switch(c)  
 {  
 case 'I': hf = Initalization();printf("初始化完成，请继续或退出\n");break;  
 case 'E': Encoding(hf);printf("编码结束，编码结果存于CodeFile文件\n");break;  
 case 'D': Decoding(hf);printf("译码结束，译码即如果存于TextFile中\n");break;  
 case 'T': TreePrint(hf);break;  
 case 'P': Print();printf("\n");break;  
 default:  
 printf("输入的选项错误，请重新输入\n");  
 break;  
 }  
 }  
}  
  
int main(){  
 HuffmanTree hf;// = Initalization();  
 interface(hf);  
 return 0;  
  
}

# 运行结果





# 分析与思考

1. 对数据结构的定义应该更慎重一点， 比如这个试验中，可以吧数组独立为一个属性， 而不是每个数组元素包含大量重复元素。
2. 在一个数组中找两个最小值，或许可以将叶子节点权重先排序在一个新的数组，再通过处理数组得值。这样虽然牺牲了空间，却减少了计算量。