# 实验报告

**题目**：试写一个哈夫曼码的编/译码系统

**班级**：电信1808 **姓名**：喻越 **学号**：U201813467 **完成日期**：2019.05.07

1. **需求分析**

**1.输入的形式和输入值的范围：**为了程序调试方便，无输入

**2.输出的形式：**程序直接显示对应字符的霍夫曼码与相应的码所解信息

**3.程序所能达到的功能**

* 1. 创建及初始化霍夫曼树
  2. 得到 A~Z与‘ ’的霍夫曼码表
  3. 进行霍夫曼编码
  4. 进行霍夫曼解码

**4.测试数据**

报文“THE PROJECT IS MY FAVORITE”的编码为：

1101000101011110001000101001110000100001000000110111101100011111110010100011111110011101011000001001001001101101010

**二、概要设计**

1. **所有抽象数据类型的定义**：

为实现上述功能，需要一个抽象数据类型：树

ADT Tree{

Data

树是由一个根结点和若干棵子树构成。树中结点具有相同数据类型及层次关系。

Operation

InitTree(\*T):构造空树T

DestroyTree(\*T):销毁树T

CreateTree(\*T,definition):按definition中给出树的定义来构造树

ClearTree(\*T):若树T存在，则将树T情空为空树

TreeEmpty(T):若T为空树，返回true，否则返回false

TreeDepth(T):返回T的深度

Root(T):返回T的根结点

Value(T,cur\_e):cur\_e是树T中的一个结点，返回此结点的值

Assign(T,cur\_e,value):给树T的结点cur\_e赋值为value

Parent(T,cur\_e):若cur\_e是树T的非根结点，则返回它的双亲，否则返回空

LeftChild(T,cur\_e):若cur\_e是树T的非叶结点，则返回它的最左孩子，否则返回空

RightSibling(T,cur\_e):若cur\_e有右兄弟，则返回它的右兄弟，否则返回空

InsertChild(\*T,\*p,i,c):其中p指向树T的某个结点，i为所指结点p的度上加1，非空树c与T不相交，操作结果为插入c为树T中p指结点的第i棵子树

DeleteChild(\*T,\*p,i):其中p指向树T的某个结点，i为所指结点p的度，操作结果为删除T中p所指结点的第i棵子树

endADT

**2.本程序包含四个模块**

1.主程序模块

**int main()**

**{**

建立并初始化霍夫曼树；

运行程序**hfmcoding;**

输出对应的编码结果

运行程序**hfmdecoding;**

输出对应的解码结果

**return 0;**

**}**

* 1. 霍夫曼树建立模块——实现霍夫曼树的建立与初始化
  2. 霍夫曼编码模板——对对应报文进行霍夫曼编码
  3. 霍夫曼解码模板——对对应霍夫曼码进行解码

**各模块之间的调用关系：**

主程序模块

霍夫曼树建立与初始化模块 霍夫曼编码模板 霍夫曼解码模板

1. **详细设计**

**1.主要数据结构**

**typedef struct \_huffmantree**//霍夫曼数结点数据类型

**{**

**char ch;**

**int weight;**

**int parent,lchild,rchild;**

**} Huffmantree;**

**typedef struct \_hfmcode**//一个字母对应的码的数据类型

**{**

**char code[M];**

**} Hfmcode;**

**char \*codestr;**//字符串类型

**#define M 100**//全局变量

1. **主程序算法**

**int main()**

**{**

**Huffmantree\*** **hfmtree=(Huffmantree\*)malloc((2\*N-1)\*sizeof(Huffmantree));**

**char \*codestr = "THE PTOJECT IS MY FAVORITE";**

**int pindu[30]=**

**{186,64,13,22,32,103,21,15,47,57,1,5,32,20,57,63,15,1,48,51,80,23,8,18,1,16,1};**

**int i;**

**creat(hfmtree, pindu, N);**//初始化霍夫曼数

/\*进行霍夫曼编码\*/

**Hfmcode codelist[N];**

**hfmcoding(hfmtree, codelist, N);**

**for(i=0; i<N; i++)**

**{**

**if(codestr[i] == ' ')**

**//printf("%c: %s\n", codestr[i], codelist[0].code);**

**printf("%s",codelist[0].code);**

**else**

**//printf("%c:%s\n",codestr[i], codelist[codestr[i]-'A'+1].code);**

**printf("%s",codelist[codestr[i]-'A'+1].code);**

**}**

**printf("\n");**

/\*进行霍夫曼解码\*/

**char \* decodestr=**

**"1101000101011110001000101001110000100001000000110111101100011111110010100011111110011101011000001001001001101101010";**

**char \* decodelist = (char\*)malloc(M\*sizeof(char));//设置最多只能解码出M个字母**

**int x=hfmdecoding(hfmtree, decodestr, decodelist, N);**

**if(x)**

**puts(decodelist);**

**else**

**printf("解码失败\n");**

**return 0;**

**}**

**3.霍夫曼树的建立和初始化算法**

**void creat(Huffmantree \*hfmtree, int pindu[30], int n)**

**{**

**int i,j;**

**/\*下面进行结点赋值操作\*/**

**for(i=0; i<n; i++)**

**{**

**hfmtree[0].ch=' ';**

**if(i>0)**

**{**

**hfmtree[i].ch='A'+i-1;**

**}**

**hfmtree[i].weight = pindu[i];**

**hfmtree[i].parent = hfmtree[i].lchild = hfmtree[i].rchild = -1;**

**}**

**/\*下面进行Huffmantree的搭建操作\*/**

**for(i=n; i<2\*n-1; i++)**

**{**

**int mw1 = 1000000;**

**int mi1 = -1;**

**int mw2 = 1000000;**

**int mi2 = -1;**

**/\*寻找权值最小的两个结点\*/**

**for(j=0; j<i; j++)**

**{**

**if(hfmtree[j].parent == -1)**

**{**

**if(hfmtree[j].weight < mw1)**

**{**

**mi1 = j;**

**mw1 = hfmtree[j].weight;**

**}**

**}**

**}**

**for(j=0; j<i; j++)**

**{**

**if( j!= mi1 && hfmtree[j].parent==-1)**

**{**

**if(hfmtree[j].weight < mw2)**

**{**

**mi2 = j;**

**mw2 = hfmtree[j].weight;**

**}**

**}**

**}**

**//printf("min\_1, %d, %d\n", mi1, hfmtree[mi1].weight);**

**// printf("min\_2, %d, %d\n", mi2, hfmtree[mi2].weight);**

**hfmtree[i].weight= mw1 + mw2;**

**hfmtree[i].parent= -1;**

**hfmtree[i].lchild = mi1;**

**hfmtree[i].rchild = mi2;//记录孩子结点次序**

**hfmtree[mi1].parent = i;//记录双亲结点次序**

**hfmtree[mi2].parent = i;**

**}**

**}**

1. **霍夫曼编码程序**

**/\*先做出霍夫曼对应表，再根据要求进行对应编码\*/**

**void hfmcoding(Huffmantree \*hfmtree, Hfmcode codelist[N], int n)**

**{**

**int i,c,p;**

**for(i=0; i<n; i++)**

**{**

**char \*s = (char \*)malloc(sizeof(char) \* M);**

**c=i;**

**p=hfmtree[c].parent;**

**int num = M;**

**s[num] = '\0';**

**while(p!=-1)**

**{**

**if(hfmtree[p].lchild==c)**

**{**

**s[--num]='0';**

**}**

**else**

**s[--num]='1';**

**c=p;**

**p=hfmtree[c].parent;**

**}**

**for (int j=0; num<=M; j++, num++)**

**codelist[i].code[j] = s[num];//**

**//for(int x=0;x<=M; ++x)**

**//printf("每个字母对应的编码：**

**%c ，%s\n",'A'+i-1,codelist[i].code[x]);**

**free(s);**

**}**

**}**

**函数的调用关系反映了模拟程序的层次关系：**

**main**

**creat hfmcoding hfmdecoding**

1. **调试分析**

**1.** **问题解决及经验与体会**

① .最开始由于大量参考室友程序且本身思路不清晰，浪费了许多时间，后来经过自己仔细琢磨，最终成功，说明“自食其力，丰衣足食”。

1. .将各字符频度和待编码的文字与待解码的霍夫曼均写入程序中，给后面调试带来了极大的便利
2. . 在if(decodestr[i] == '0')

root = hfmtree[root].lchild;

else if(decodestr[i] == '1')

root = hfmtree[root].rchild;

else

return 0; 这个if算法里对于‘0’与‘1’两个字符没有加单引号的低级错误应该少犯。

**2.算法的时空分析**

程序create和hfmcoding的时间复杂度均为O（n^2）,程序hfmdecoding的时间复杂度为O（n）,整个程序的时间复杂度O（n^2）。

1. **用户使用说明**

1..程序设定要编码和解码的字符数不得超过100

1. **测试结果**

编码：

1101000101011110001000101001110000100001000000110111101100011111110010100011111110011101011000001001001001101101010

解码：

THE PTOJECT IS MY FAVORITE

**附录**

**源程序文件名清单：**

**stdio.h**

**stdlib.h**  //运行malloc和free函数的源程序

**string,h**