

学生学号	0121510880209	实验课成绩	
------	---------------	-------	--

武汉理工大学

学 生 实 验 报 告 书

实验课程名称 数据结构

开 课 学 院 计算机科学与技术学院

指导老师姓名 钟欣

学 生 姓 名 彭玉全

学生专业班级 软件 1503

2016 — 2017 学年 第 1 学期

实验课程名称： 数据结构

实验项目名称	哈夫曼编码及译码算法的设计及实现			实验成绩	
实验者	彭玉全	专业班级	软件 1503	组别	
同组者				实验日期	2016-11-28

第一部分：实验分析与设计（可加页）

一、实验内容描述（问题域描述）

实验题 5

编制赫夫曼编码

【问题描述】对任意输入的一段英文，为每个字符编制其相应的赫夫曼编码；并利用该编码为任意输入的 0、1 序列进行解码。

基本要求：一个完整的系统应具有以下功能：

（1）初始化 从终端读入一段英文字符，统计每个字符出现的频率，建立赫夫曼树，并将该树存入某文件；

（2）编码 利用建好的赫夫曼树对各字符进行编码，用列表的形式显示在屏幕上，并将编码结果存入另一文件中；

（3）解码 利用保存的赫夫曼编码，对任意输入的 0，1 序列能正确解码；

二、实验基本原理与设计(算法与程序设计)

问题分析：

解决本问题可以通过利用二叉树来设计二进制的前缀编码。假设每种字符在出现的次数为 w ，其编码长度为 l ，英文字符串中只有 n 中字符，可得到英文字符的总长度。对应到二叉树上，可得叶子结点的权，恰为从根到叶子结点的路径长度。由此可见，设计编码的最短的二进制前缀编码即为以 n 中字符出现的频率作为权，设计一棵赫夫曼树的问题。

由于赫夫曼树中没有度为 1 的结点，则一棵有 n 个叶子结点的赫夫曼树共有 $2n-1$ 个结点，可以存储在一个大小为 $2n-1$ 的一维数组中。由于在构成赫夫曼树后，为求编码需要从叶子结点出发走一条从叶子结点到根的路径；而为译码需要从根出发走一条从根到叶子结点的路径。

算法分析：

```
typedef struct
{
    int weight;
```

```

    int parent, lchild, rchild;
}HTNode, *HuffmanCode;//动态分配数组存储赫夫曼树

typedef char **HuffmanCode;//动态分配数组存储赫夫曼编码表
void HuffmanCoding(HuffmanTree &HT, HuffmanCode &HC, int *w, int n)
{
    //w 存放 n 个字符的权值。构造哈夫曼树 HT 并求出 n 个字符的赫夫
    曼编码
    int s1, s2;
    int m;
    int i;
    HuffmanTree p;
    if (n <= 1) return;
    m = 2 * n - 1;
    HT = (HuffmanTree)malloc(m*sizeof(HTNode)); //0 号单元未用
    for (i = 1, p = HT; i<n; ++i, ++p)
    {
        p->weight = w[i];
        p->lchild = -1;
        p->parent = -1;
        p->rchild = -1;
    }
    for (; i<m; ++i, ++p)
    {
        p->weight = 0;
        p->lchild = -1;
        p->parent = -1;
        p->rchild = -1;
    }
    for (i = n; i<m; ++i)//构建赫夫曼树
    {
        //在 HT[] 选择 parent 为 0 且 weight 最小的两个结点,其序号 s1 s2
        Select(HT, i - 1, s1, s2);
        HT[s1].parent = i;
        HT[s2].parent = i;
        HT[i].lchild = s1;
        HT[i].rchild = s2;
        HT[i].weight = HT[s1].weight + HT[s2].weight;
    }
    //从叶子到根逆向求每个字符的赫夫曼编码

```

```

        HC = (HuffmanCode)malloc(n*sizeof(char *)); //分配 n 个字符编码
        的头指针向量
        char *cd;
        cd = (char*)malloc(n*sizeof(char)); //分配求编码的工作空间
        cd[n - 1] = '\0'; //编码结束符
        for (i = 1; i < n; ++i) //逐个字符求编码
        {
            int start = n - 1; //编码结束位置
            for (int c = i, f = HT[i].parent; f != -1; c = f, f =
HT[f].parent) //从叶子到根求编码
            {
                if (HT[f].lchild == c) cd[--start] = '0';
                else cd[--start] = '1';
            }
            HC[i] = (char *)malloc((n - start)*sizeof(char)); //为第 i 个
            字符编码分配空间
            strcpy_s(HC[i], 1, &cd[start]); //从 cd 复制字符编码中 HC
        }
        free(cd);
    }

```

源代码:

```

#include<iostream>
#include<stdio.h>
#define N 3 //叶子数目
#define M (2*N-1) //结点总数
#define MAXVAL 10000.0
#define MAXSIZE 100 //哈夫曼编码的最大位数
using namespace std;

typedef struct
{
    char ch;
    float weight;
    int lchild, rchild, parent;
}Hufmtree;

typedef struct
{
    char bits[N]; //位串

```

```

        int start;          //编码在位串中的起始位置
        char ch;            //字符
    }Codetype;

    void CHuffman(Hufmtree tree[])//建立哈夫曼树
    {
        int i, j, p1, p2; //p1, p2 分别记住每次合并时权值最小和次小的两个根
        结点的下标
        float small1, small2, f;
        char c;
        for(i=0; i<M; i++)    //初始化
        {
            tree[i].parent=0;
            tree[i].lchild=-1;
            tree[i].rchild=-1;
            tree[i].weight=0.0;
        }
        cout<<"依次读入前"<<N<<"个结点的字符及权值(空格隔开)\n";
        for(i=0; i<N; i++)    //读入前 n 个结点的字符及权值
        {
            cout<<"输入第"<<(i+1)<<"个字符 权值:";
            cin>>c>>f;
            getchar();
            tree[i].ch = c;
            tree[i].weight = f;
        }
        for(i = N; i < M; i++)    //进行 n-1 次合并，产生 n-1 个新结点
        {
            p1 = 0 ;p2 = 0;
            small1 = MAXVAL; small2 = MAXVAL;    //maxval 是 float 类型
            的最大值
            for(j=0; j<i; j++)    //选出两个权值最小的根结点
            if(tree[j].parent==0)
            if(tree[j].weight<small1)
            {
                small2=small1;    //改变最小权、次小权及对应的位置
                small1=tree[j].weight;
                p2=p1;
                p1=j;
            }
        }
    }

```

```

    }
    else if(tree[j].weight<small2)
    {
        small2=tree[j].weight; //改变次小权及位置
        p2=j;
    }
    tree[p1].parent=i;
    tree[p2].parent=i;
    tree[i].lchild=p1; //最小权根结点是新结点的左孩子
    tree[i].rchild=p2; //次小权根结点是新结点的右孩子
    tree[i].weight=tree[p1].weight+tree[p2].weight;
}
}

void HuffmanCode(Codetype code[],Hufmtree tree[])//根据哈夫曼树求出哈夫曼编码
//codetype code[]为求出的哈夫曼编码
//hufmtree tree[]为已知的哈夫曼树
{
    int i,c,p;
    Codetype cd; //缓冲变量
    for(i=0;i<N;i++)
    {
        cd.start=N;
        cd.ch=tree[i].ch;
        c=i; //从叶结点出发向上回溯
        p=tree[i].parent; //tree[p]是 tree[i]的双亲
        while(p!=0)
        {
            cd.start--;
            if(tree[p].lchild==c)
                cd.bits[cd.start]='0'; //tree[i]是左子树，生成代
码'0'
            else
                cd.bits[cd.start]='1'; //tree[i]是右子树，生成代
码'1'
            c=p;
            p=tree[p].parent;
        }
    }
}

```

```

        code[i]=cd;    //第 i+1 个字符的编码存入 code[i]
    }
}

void decode(Hufmtree tree[])//依次输入，根据哈夫曼树译码
{
    int i,j=0;
    char b[MAXSIZE];
    char endflag='2';    //结束标志取 2
    i=M-1;                //从根结点开始往下搜索
    cout<<"输入发送的编码(以'2'为结束标志)： ";
    gets(b);
    cout<<"译码后的字符为->";
    while(b[j]!='2')
    {
        if(b[j]=='0')
            i=tree[i].lchild;    //走向左孩子
        else
            i=tree[i].rchild;    //走向右孩子
        if(tree[i].lchild==-1)    //tree[i]是叶结点
        {
            printf("%c",tree[i].ch);
            i=M-1;    //回到根结点
        }
        j++;
    }
    cout<<endl;
    if(tree[i].lchild!=-1&&b[j]!='2')    //读完，但尚未到叶子结点
        cout<<"Error"<<endl;
}

int main()
{
    cout<<"----哈夫曼编码----\n";
    cout<<"当前可编码"<<N<<"个字符\n";
    Hufmtree tree[M];
    Codetype code[N];
    int i,j;//循环变量
    CHuffman(tree);//建立哈夫曼树

```



```

        HuffmanCode(code, tree); //根据哈夫曼树求出哈夫曼编码
        cout<<"请输出每个字符的哈夫曼编码\n";
        for(i=0; i<N; i++)
        {
            printf("%c: ", code[i].ch);
            for(j=code[i].start; j<N; j++)
                printf("%c ", code[i].bits[j]);
            printf("\n");
        }
        cout<<"请输入代码进行译码\n";
        decode(tree); //依次输入，根据哈夫曼树译码
        return 0;
    }

```

三、主要仪器设备及耗材

1. 实验设备

PC 机

2. 开发环境

VS 2015 Pro


第二部分：实验调试与结果分析（可加页）

一、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1. 调试方法描述

完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2. 实验输入/输出数据记录



```
E:\编程练习2016\shiyans5final.exe
---哈夫曼编码---
当前可编码3个字符
依次读入前3个结点的字符及权值(空格隔开)
输入第1个字符 权值:a 1
输入第2个字符 权值:b 2
输入第3个字符 权值:c 3
请输出每个字符的哈夫曼编码
a: 1 0
b: 1 1
c: 0
请输入代码进行译码
输入发送的编码(以'2'为结束标志): 10110011102
译码后的字符为->abccba

-----
Process exited after 21.45 seconds with return value 0
请按任意键继续. . .

E:\编程练习2016\shiyans5final.exe
---哈夫曼编码---
当前可编码6个字符
依次读入前6个结点的字符及权值(空格隔开)
输入第1个字符 权值:a 1
输入第2个字符 权值:b 2
输入第3个字符 权值:c 3
输入第4个字符 权值:d 4
输入第5个字符 权值:e 5
输入第6个字符 权值:f 6
请输出每个字符的哈夫曼编码
a: 1 1 1 1 0
b: 1 1 1 1 1
c: 1 1 0
d: 0 0
e: 0 1
f: 1 0
请输入代码进行译码
输入发送的编码(以'2'为结束标志): 100100110111111102
译码后的字符为->fedcba

-----
Process exited after 40.02 seconds with return value 0
请按任意键继续. . .
```

二、实验小结、建议及体会

本次实验中所遇到的主要问题就是赫夫曼编码算法，以及编写过程中对变量和指针的控制。编写过程中出现了较多的问题，比如开始对赫夫曼树的理解不是很清楚，导致在编写过程中某些代码错误而没能及时修改，在最后进行修改时遇到了较多的麻烦。通过本次实验，掌握了树和哈夫曼树的基本操作，以及程序的整个算法，同时了解到赫夫曼编码是一种编码方式，以赫夫曼树一即最优二叉树，带权路径长度最小的二叉树，经常应用于数据的无损耗压缩。总之受益匪浅。

总之，实验本身不顺利为此丢掉了一些功能，但编码译码的功能最终还是实现了的。

实验课程名称： 数据结构

实验项目名称	图的算法设计及应用			实验成绩	
实验者	彭玉全	专业班级	软件 1503	组别	
同组者				实验日期	2016-11-28

第一部分：实验分析与设计（可加页）

一、实验内容描述（问题域描述）

实验题 6 为新建医院选址

【问题描述】 n 个村庄之间的无向图，边上的权值 $w(i, j)$ 表示村庄 i 和 j 之间道路长度。现要从这 n 个村庄中选择一个村庄新建一所医院，使离医院最远的村庄到医院的路程最短。设计一程序求解此问题。

【基本要求】

用邻接矩阵表示无向网，应显示所选中的村庄到各村庄的最短距离。

二、实验基本原理与设计(算法与程序设计)

问题分析：

解决此问题即为解决有向图中心点问题。输入村庄的个数，名称，以及村庄之间路的个数以及每一条路的长度，程序根据权值以及路来求解医院的选址。对医院选址的要求是每个村庄到医院的路径最长的值要最小。

算法设计：

C 语言描述的迪杰斯特拉算法

```
void ShortestPath_DIJ(MGraph G, int v0, PathMatrix &P,
ShortPathTable &D) {
    //用 Dijkstra 算法求有向网 G 的  $v_0$  顶点  $v_0$  到其余顶点  $v$  的最短路径
    P[v] 及其带权长度 D[v]
    //若 P[v][w] 为 true 则 w 是从  $v_0$  到  $v$  当前求得最短路径上的顶点
    //final[v] 为 true 当且仅当  $v$  属于 S 即已经求得  $v_0$  到  $v$  的最短路径
    for(v=0;v<G.vexnum; ++v) {
        final[v] = FALSE; D[v] = G.arcs[v0][v];
        for(w=0;w<G.vexnum; ++w)
            P[v][w] = FALSE; //设空路径
        if(D[v]<INFINITY) {
            P[v][v0] = TRUE;
            P[v][v] = TRUE;
        }
    }
}
```

```

D[v0] = 0; final[v0]=TRUE;
//开始主循环，每次求得 v0 到某个 v 顶点的最短路径，并加 v 到 s 集
for(i=1;i<G.vexnum;i++) { //其余 G.vexnum-1 个顶点
    min = INFINITY; //当前所知离 v0 顶点的最短距离
    for(w =0;w<G.vexnum;w++)
        if(!final[w])
            if(D[w] < min)
                {v = w;min = D[w];}
    final[v] = TRUE;
    for(w=0;w<G.vexnum;w++)
        if(!final[w]&&(min + G.arcs[v][w] < D[w])) {
            D[w] = min + G.arcs[v][w];
            P[w] = P[v];
            P[v][w] = TRUE;
        }
}
}
}

```

```

/*求图 G 中顶点 s 到其他顶点的最短路径中的最大值并返回*/
int PPaths(MGraph G,int s)
{
    int i,j,k,v;
    int d[MAX]={0}; //存放顶点 s 到其他顶点的最短路径
    int p[MAX]={0}; //判断村庄是否属于集合 V 的辅助数组
    p[s]=1;
    int min;
    int max=0;
    for(i=0;i<G.n;i++)
    {
        d[i]=G.w[s][i];
    }
    for(k=0;k<G.n-1;k++)
    {
        min=INFINITY;
        for(i=0;i<G.n;i++) //从未求得最短路径的顶点（不在集合 V）中
            选择路径长度最小的终点 v：即求得 s 到 v 的最短路径
            {
                if(p[i]==0&&d[i]<min)

```

```

        {
            min=d[i];
            v=i;
        }
    }
    p[v]=1;//将顶点 v 加入集合 V
    for(j=0;j<G.n;j++)//修改最短路径: 计算 v 的邻接点的最短路径,
    若  $(s, \dots, v) + (v, j) < (s, \dots, j)$ , 则以  $(s, \dots, v, j)$  代替
    {
        if (p[j]==0&& d[j]>d[v]+G.w[v][j])
        {
            d[j]=d[v]+G.w[v][j];
        }
    }
}
for(i=0;i<G.n;i++)
{
    if(max<d[i])max=d[i];
}
return max;
}

```

源代码:

```

#include<iostream>
#include<climits>
#include<stdio.h>
using namespace std;

#define INFINITY 32767
const int MAX=20;
struct MGraph
{
    int n;
    int m;
    int w[MAX][MAX];
};
/*用邻接矩阵创建图 G*/
int CreateUDN(MGraph &G)
{

```

```

        cout<<"请输入村庄数： ";
        cin>>G.n;
        cout<<"请输入村庄之间道路数： ";
        cin>>G.m;
        int i,j,k;
        for(i=0;i<G.n;i++)
        {
            for(j=0;j<G.n;j++)//初始化邻接矩阵
            {
                if(j!=i)
                {
                    G.w[i][j]=INFINITY;
                }
                else
                {
                    G.w[i][j]=0;
                }
            }
        }
        cout<<"村庄依次用数字 0 到"<<G.n-1<<"代替。 \n";
        cout<<"村庄 a "<<"村庄 b "<<"距离\n";
        for(k=0;k<G.m;k++)
        {
            cin>>i;
            cin>>j;
            cin>>G.w[i][j];
            G.w[j][i]=G.w[i][j];
        }
        return 1;
    }
    /*求图 G 中顶点 s 到其他顶点的最短路径中的最大值并返回*/
    int PPaths(MGraph G,int s)
    {
        int i,j,k,v;
        int d[MAX]={0}; //存放顶点 s 到其他顶点的最短路径
        int p[MAX]={0}; //判断村庄是否属于集合 V 的辅助数组
        p[s]=1;
        int min;

```

```

        int max=0;
        for(i=0;i<G.n;i++)
        {
            d[i]=G.w[s][i];
        }
        for(k=0;k<G.n-1;k++)
        {
            min=INFINITY;
            for(i=0;i<G.n;i++)//从未求得最短路径的顶点（不在集合 V）中
            选择路径长度最小的终点 v：即求得 s 到 v 的最短路径
            {
                if(p[i]==0&&d[i]<min)
                {
                    min=d[i];
                    v=i;
                }
            }
            p[v]=1;//将顶点 v 加入集合 V
            for(j=0;j<G.n;j++)//修改最短路径：计算 v 的邻接点的最短路径，
            若  $(s, \dots, v) + (v, j) < (s, \dots, j)$ ，则以  $(s, \dots, v, j)$  代替
            {
                if(p[j]==0&&d[j]>d[v]+G.w[v][j])
                {
                    d[j]=d[v]+G.w[v][j];
                }
            }
        }
        for(i=0;i<G.n;i++)
        {
            if(max<d[i])max=d[i];
        }
        return max;
    }

    int main()
    {
        MGraph G;
        CreateUDN(G);
        cout<<"图的邻接矩阵为：\n";
    }

```

```

        for(int i=0;i<G.n;i++)
        {
            for(int j=0;j<G.n;j++)
            {
                if(G.w[i][j]==INFINITY)
                {
                    cout<<"∞\t";
                }
                else
                {
                    cout<<G.w[i][j]<<"\t";
                }
            }
            cout<<endl;
        }
        int min=INFINITY;
        int locate;
        for(int i=0;i<G.n;i++)//求图 G 中各顶点到其他顶点的最短路径中的
        最大值的最小值
        {
            if(min>=PPaths(G,i))
            {
                min=PPaths(G,i);
                locate=i;
            }
        }
        cout<<"医院应建在村庄";
        cout<<locate;
        cout<<"\n";
        return 0;
    }

```

三、主要仪器设备及耗材

1. 实验设备

PC 机

2. 开发环境

VS 2015 Pro

第二部分：实验调试与结果分析（可加页）

三、调试过程（包括调试方法描述、实验数据记录，实验现象记录，实验过程发现的问题等）

1. 调试方法描述

完成代码输入后，运行程序，出现内存溢出 bug。使用断点依照主函数代码运行流程逐代码块，逐句调试。直至解决所有 bug，程序运行通过。

2. 实验输入/输出数据记录

```
E:\编程练习2016\shiyang6_3.exe
请输入村庄数: 3
请输入村庄之间道路数: 3
村庄依次用数字0到2代替。
村庄a 村庄b 距离
0 1 2
0 2 3
1 2 6
图的邻接矩阵为:
0      2      3
2      0      6
3      6      0
医院应建在村庄0

-----
Process exited after 15.98 seconds with return value 0
请按任意键继续. . .
```

```
E:\编程练习2016\shiyang6_3.exe
请输入村庄数: 5
请输入村庄之间道路数: 6
村庄依次用数字0到4代替。
村庄a 村庄b 距离
1 2 4
2 3 5
3 4 6
1 3 7
1 4 9
0 1 3
图的邻接矩阵为:
0      3      ∞      ∞      ∞
3      0      4      7      9
∞      4      0      5      ∞
∞      7      5      0      6
∞      9      ∞      6      0
医院应建在村庄1

-----
Process exited after 34.37 seconds with return value 0
请按任意键继续. . .
```

四、实验结果小结体会

经过问题分析，算法设计，程序编写，实验最终取得成功。经过备查实验加深了我对图的相关知识的了解，对最短路径问题有了更为深刻的认识，编写过程中出现很多意想不到的问题，主要是循环变量把握不准，指针指向不正确等造成的内存溢出，死循环，结果错误等。数据结构是一门很难的课，需要不断的实践和学习，单独看代码是远远不够的，最重要的就是上机动手实践，在编码过程中发现问题解决问题。

