**电子科技大学**

**计算机科学与工程学院**

**标 准 实 验 报 告**

**（实验）课程名称 软件开发综合实验**

**电子科技大学教务处制表**

**电 子 科 技 大 学**

# 实 验 报 告 一

**学生姓名：覃翊 学 号： 2016060107008 指导教师：王丽杰**

**实验地点： 主楼B412 实验时间：11.09-11.16**

**一、实验室名称： 计算机学院实验中心A2-412**

**二、实验项目名称：电话号码本**

**三、实验学时：8学时**

**四、实验原理：**

通过c语言文件操作结合数组，用哈希算法实现数据的存储，查找以及压缩功能。

**五、实验目的：**

设计系统，增加编程能力，考验调试能力，学习常见的哈希算法，调试冲突解决策略

**六、实验内容：**

实现一个信息管理系统，具备增加，删除，查找，修改的功能，具有从文件导入，导出至文件的功能，用哈希算法实现快速查找，具备冲突解决策略。能根据姓名快速查找出对应的电话以及地址，若查询失败，则询问是否将此人及其数据加入数据库中。

1. **实验步骤：**

首先，我先编写了界面，创建了一个mode变量，用来控制程序的过程以及结束，创建了一个while循环，mode为0则跳出，mode为1则程序进行。这样能循环显示界面，无限地操作。在while循环里，我先显示菜单，然后用户输入一个指令，用switch函数来识别用户输入的指令。

当用户输入1时，屏幕显示我预先说明的说明书，包含有对本系统的使用，全文如下：操作1与操作2二选一打开电话本，每次需打开电话本后才能进行操作，本电话本系统支持多电话本管理，不同的文件名对应不同的电话本，打开电话本成功后会得到提示，此时可进行列表所有操作，需要注意的是，当一切操作 结束后，需执行操作4以保存，这样才能从内存将数据保存至文件，本系统高效，稳定，祝使用愉快！

当用户输入2时，会新建一个文件，用fopen函数，fp = fopen(listname, "w+"); 参数为w+，是写入文件操作，若没有此文件则创建此文件。打开文件后，指针保留，方便之后的操作。

printf("请输入电话本名字：");

scanf("%s", listname);

fp = fopen(listname, "w+");

printf("新建成功，请继续操作。\n");

当用户输入3时，打开对应名称的文件，存入已有的数组，然后立刻关闭文件，周所有的操作都在内存中进行了。

printf("请输入要打开的电话本名字：");

scanf("%s", listname);

fp = fopen(listname, "r");

store(fp);

printf("打开电话本成功，请继续您的操作。\n");

fclose(fp);

当用户输入4时，会将数组内的所有内容导入文件，这个文件并不是之前打开的文件，而是源文件的补充，重新抹掉之后再把所有的数据导入。

fp = fopen(listname, "w+");

for(i = 0; i < 1000; ++i){

if(Hflag[i] >= 1){

fprintf(fp, "%d\t%d\t%s\t%s\t%s\n", Hnumb[i], Hflag[i], name[i], numb[i], addr[i]);

}

}

fclose(fp);

printf("保存成功！\n");

当用户输入5时，会从数据库中查找，具体步骤是先把名字用哈希算法得到一个基础值，用这个值进行查找，若这个值对应的数组内容就是它本身，就输出，否则用冲突处理函数，用平方再散列法寻找，因为我设置的哈希函数就是用平方再散列法解决的冲突。

printf("要显示联系人的姓名：");

scanf("%s", nametmp);

Hnumbtmp = SearchHTable(nametmp);

if(Hnumbtmp > 0){

printf("姓名：%s\t电话:%s\t地址：%s\n", name[Hnumbtmp], numb[Hnumbtmp], addr[Hnumbtmp]);

}else if(Hnumbtmp == -1){

printf("查找失败\n是否添加此人信息？按1回车确认，否则按0\n");

scanf("%d", &tmp);

if(tmp){

Hnumbtmp = GetHkey(nametmp);

Hnumbtmp = CollisionKey(Hnumbtmp);

Hflag[Hnumbtmp] = 1;

strcpy(&name[Hnumbtmp][0], nametmp);

printf("请输入该联系人住址：");

scanf("%s", addr[Hnumbtmp]);

printf("请输入联系人电话：");

scanf("%s", numb[Hnumbtmp]);

}

}

break;

当用户输入6时，会新建联系人，我先当用户输入要新建的联系人，然后先判断该名称是否在数据库里，判断的具体方法是先取得该名字对应的哈希值，然后对应的冲突处理，以确认该值的确对应的是该名字，然后把这个值对应的flag是否有标记，如果发现这个字符串确认有标记了，意思就是该名字已经在数据库里了。如果名字已经在数据库里了，则询问是否覆盖，然后让用户输入数据进数组，标记flag=1，如果不覆盖，在名字后添加编号以表示序号。如果名字不在数据库，则照常新建输入数据。

printf("请输入姓名：");

scanf("%s", nametmp);

if(Hflag[CollisionKey(GetHkey(nametmp))] == 0){

Hnumbtmp = GetHkey(nametmp);

Hnumbtmp = CollisionKey(Hnumbtmp);

Hnumb[Hnumbtmp] = Hnumbtmp;

Hflag[Hnumbtmp] = 1;

strcpy(&name[Hnumbtmp][0], nametmp);

printf("请输入该联系人住址：");

scanf("%s", addr[Hnumbtmp]);

printf("请输入联系人电话：");

scanf("%s", numb[Hnumbtmp]);

}else{

printf("该联系人已存在，是否覆盖？按1回车确认，否则按0\n");

scanf("%d", &tmp);

if(tmp){

Hnumbtmp = GetHkey(nametmp);

Hflag[Hnumbtmp] = 1;

Hnumb[Hnumbtmp] = Hnumbtmp;

strcpy(&name[Hnumbtmp][0], nametmp);

printf("请输入该联系人住址：");

scanf("%s", addr[Hnumbtmp]);

printf("请输入联系人电话：");

scanf("%s", numb[Hnumbtmp]);

}else{

char plus[2] = {'\*'};

strcat(nametmp, plus);

Hnumbtmp = GetHkey(nametmp);

Hnumbtmp = CollisionKey(Hnumbtmp);

Hflag[Hnumbtmp] = 1;

Hnumb[Hnumbtmp] = Hnumbtmp;

strcpy(&name[Hnumbtmp][0], nametmp);

printf("请输入该联系人住址：");

scanf("%s", addr[Hnumbtmp]);

printf("请输入联系人电话：");

scanf("%s", numb[Hnumbtmp]);

}

}

printf("添加成功！\n");

当用户输入7时，可以修改联系人，就是修改各个数组的数据，直接重新覆盖入原来的数组就好了。

printf("要修改联系人的姓名：");

scanf("%s", nametmp);

Hnumbtmp = SearchHTable(nametmp);

printf("新电话号码：");

scanf("%s", numb[Hnumbtmp]);

printf("新住址：");

scanf("%s", addr[Hnumbtmp]);

printf("修改成功");

当用户输入8时，删除联系人，直接把flag置为0就好了，这样在程序中所有的判断都不会认为这里有数据，导入文件的时候直接就抛掉了，使用到查找的算法里最后一步都是判断flag,所以flag置为0就直接删除 掉数据了。

printf("请输入要删除联系人的名字：");

scanf("%s", nametmp);

DeleteHTable(nametmp);

printf("删除成功！\n");

break;

当用户输入9时，显示联系人信息，直接判断flag值，值大于等于1就显示出来，大于等于1的flag值有1 ，2 ，3. 2对应的是亲情联系人，3对应的是黑名单，黑名单不是删除，所以在显示的时候也会显示出来。

printf("请输入要设为亲情联系人的名字：");

scanf("%s", nametmp);

Hnumbtmp = SearchHTable(nametmp);

Hflag[Hnumbtmp] = 2;

printf("设置成功！\n");

当用户输入10时，设为亲情联系人就把flag置为2.

当用户输入11时，显示flag等于2的联系人，即亲情联系人。

当用户输入12时，设为黑名单就把flag置为2.

当用户输入12时，移除出黑名单就把flag置为1.即把该人设为普通联系人。

当用户输入14时，显示flag等于3的联系人，即黑名单联系人。

当用户输入15时，mode等于0，因为mode是循环控制的把手，所以mode=0就可以跳出循环，结束程序。

取得哈希值的函数，我把字符串对应的ascii码加起来，然后模大于1000的最小素数，获得哈希值。

int GetHkey(char \*q){//根据姓名求出哈希地址

char \*p = q;

int itmp;

int sum = 0;

while(\*p != '\0'){

itmp = \*p;

sum += itmp;

p++;

}

return sum % 1009;

}

冲突处理办法，我用了平方再散列法解决，分别试正负1的平方，正负2的平方，以此类推，在查找时我也是用同样的方法试int CollisionKey(int Hnumbtmp){ //冲突处理

int i = 1;

while(Hnumb[Hnumbtmp] != 0){

if(Hnumb[Hnumbtmp + i\*i] == 0){

Hnumbtmp += i\*i;

}else if(Hnumb[Hnumbtmp - i\*i] == 0){

Hnumbtmp -= i\*i;

}

i++;

}

return Hnumbtmp;

}

查找，我读入姓名，然后通过上面的函数获得哈希值，然后查找，也是使用平方再散列法查找，最多查到正负15的平方，这样就查找了30次，很多了，如果查15次找不到就认为该数据库里无此人。当然，如果数据库很大，重复的哈希值可能性很高的话，可以提高数值，但是如果像本例一样没那么高的话，提高数值，获得所谓的“保证”也只是在浪费性能而已。

int SearchHTable(char \*nametmp ){

int tmp = GetHkey(nametmp);

int i = 1, j = 0;

while(strcmp(name[tmp], nametmp)){

if(strcmp(name[tmp + i\*i], nametmp) == 0){

tmp += i\*i;

}

if(strcmp(name[tmp - i\*i], nametmp) == 0){

tmp -= i\*i;

}

if(++j > 15){

return -1;

}

++i;

}

return tmp;

}

删除数据，我直接把flag置零，之后有数据的话就直接覆盖，如果没有数据的话，在写进文件保存的时候这一串数据会自动抛掉，反正这是最简单删除数据的办法，最简单也最高明，并且，更为高明的是可以恢复数据，如果还没保存的话，法线误删了，可以直接跳过flag，不判断，找到还在内存里的文件。

void DeleteHTable(char \*nametmp){

Hnumbtmp = SearchHTable(nametmp);

Hflag[Hnumbtmp] = 0;

}

展示数据库，flag大于等于1的都能输出，不管是亲情联系人还是黑名单 ，因为黑名单只是拉黑，并没有删除，所以存在黑名单的意义就是在电话号码拨进来的时候进行判断，如果在黑名单里就直接挂断，如果删除数据就没有这个功能

void DisplayHTable(){

int i;

for(i = 0; i < 1000; ++i){

if(Hflag[i] >= 1){

printf("\t姓名：%12s\n\t电话：%12s\n\t住址：%12s\n\n", name[i], numb[i], addr[i]);

}

}

}

存储文件进内存，就是把哈希值，和flag，以及信息放进内存

void store(FILE \*fp){

while(!feof(fp)){

fscanf(fp, "%d", &Hnumbtmp);

fscanf(fp, "%d %s %s %s\n",&Hflag[Hnumbtmp], name[Hnumbtmp], numb[Hnumbtmp], addr[Hnumbtmp]);

// fscanf(fp, "%d %d %s %s %s\n", &Hnumbtmp, &Hflag[Hnumbtmp], name[Hnumbtmp], numb[Hnumbtmp], addr[Hnumbtmp]);

// printf("%d %d %s %s %s\n",Hnumbtmp, Hflag[Hnumbtmp], name[Hnumbtmp], numb[Hnumbtmp], addr[Hnumbtmp]);

Hnumb[Hnumbtmp] = Hnumbtmp;

}

}

完整代码我就不附在文件里了，太长，加上了有28页，我已发给老师，当然，我也放在了我的GitHub上：https://github.com/qinyi0411/Software-experiment-course/blob/master/first/%E7%94%B5%E8%AF%9D%E6%9C%AC.cpp

**八、总结及心得体会：**

通过本实验的练习，学会了文件操作，指针，数组等，实现了很多功能，锻炼了调试能力，程序无bug，以及做出一个完整无bug的管理系统，非常的有成就感。增加了对c语言的熟悉度，锻炼了编程思想。

**九、对本实验过程及方法、手段的改进建议：**

有一个遗憾就是没有加入查电话的功能的，明明超级容易实现，一个for内一个strcmp就完成了，如果相同，就得到了对应的哈希值，然后就把信息输出就好了，非常简单，但是由于程序已经提交了，所以添加不了，很遗憾。本程序是用c语言写的，可以再用c++语言翻译一遍，我会在接下来的时间里用c++复写一次，并加上新功能。

# 实 验 报 告 二

**学生姓名：覃翊 学 号： 2016060107008 指导教师：王丽杰**

**实验地点： 主楼B412 实验时间：11.09-11.16**

**一、实验室名称： 计算机学院实验中心A2-412**

**二、实验项目名称：迷宫**

**三、实验学时：16学时**

**四、实验原理：**

设计并实现一个迷宫游戏，算法能够自动生成具有通路的迷宫地图，寻找迷宫路径并显示通路，能够利用文件形式保存或导入迷宫。

**五、实验目的：**

增加编程能力，考验调试能力，了解图形界面开发技术，理解离散集合的定义、表示方法；并查集的基本操作、路径压缩。掌握栈的定义与基本操作

**六、实验内容：**

采用算法生成具有通路的迷宫地图；寻找迷宫路径并显示通路。

**七、实验步骤：**

首先，我先创造了选择界面，摁1从文件导入地图，摁2就新建一个地图。

如果用户输入1，switch语句将其匹配到1的代码块，让用户输入文件的名字，我给出的例子直接输入1就是文件名，读入用户输入的文件名后，用fopen函数，只读方式，打开文件，然后用while循环，读入信息。

当文件指针没有指向文件尾时，分别读入行数，列数，迷宫开头，迷宫结尾，以及迷宫数据。

当用户输入2时，就新建一个迷宫。

新建迷宫我采用的时dfs算法，用时间作种子，生成随机数，模4后得到四个方向，这四个方向不是固定方向，而是优先方向，分别对应上下左右。

什么叫优先方向呢？例如，随机数取模后得1，对应得优先方向是上，则先判断上能不能走。如果上能走，就走上。如果上不能走，再依次走左，下，右。如果随机数取模后得2，对应得优先方向是右，则判断右能不能走，如果右能走，就走右，如果右不能走，则走其他三个方向。同理，这就是优先方向，为此，我写了很长很长得if语句，还要判有没有走过。

switch(tmp){

case 1:

if(road[i].lasty - 1 >= 0 && ditu[road[i].lastx][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx][road[i].lasty - 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty - 1;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lastx + 1 < w && ditu[road[i].lastx + 1][road[i].lasty] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 2][road[i].lasty] == 1) {

road[i].x = road[i].lastx + 1;

road[i].y = road[i].lasty;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lasty + 1 < r && ditu[road[i].lastx][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx][road[i].lasty + 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty + 1;

ditu[road[i].x][road[i].y] = 0;

}

break;

case 2:

if(road[i].lastx + 1 < w && ditu[road[i].lastx + 1][road[i].lasty] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 2][road[i].lasty] == 1) {

road[i].x = road[i].lastx + 1;

road[i].y = road[i].lasty;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lasty - 1 >= 0 && ditu[road[i].lastx][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx][road[i].lasty - 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty - 1;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lasty + 1 < r && ditu[road[i].lastx][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx][road[i].lasty + 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty + 1;

ditu[road[i].x][road[i].y] = 0;

}

break;

case 3:

if(road[i].lasty + 1 < r && ditu[road[i].lastx][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx][road[i].lasty + 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty + 1;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lasty - 1 >= 0 && ditu[road[i].lastx][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx][road[i].lasty - 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty - 1;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lastx + 1 < w && ditu[road[i].lastx + 1][road[i].lasty] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 2][road[i].lasty] == 1) {

road[i].x = road[i].lastx + 1;

road[i].y = road[i].lasty;

ditu[road[i].x][road[i].y] = 0;

}

break;

case 4:

if(road[i].lasty - 1 >= 0 && ditu[road[i].lastx][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx][road[i].lasty - 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty - 1;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lastx + 1 < w && ditu[road[i].lastx + 1][road[i].lasty] == 1 && ditu[road[i].lastx + 1][road[i].lasty - 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 2][road[i].lasty] == 1) {

road[i].x = road[i].lastx + 1;

road[i].y = road[i].lasty;

ditu[road[i].x][road[i].y] = 0;

}else if(road[i].lasty + 1 < r && ditu[road[i].lastx][road[i].lasty + 1] == 1 && ditu[road[i].lastx + 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx - 1][road[i].lasty + 1] == 1 && ditu[road[i].lastx][road[i].lasty + 2] == 1) {

road[i].x = road[i].lastx;

road[i].y = road[i].lasty + 1;

ditu[road[i].x][road[i].y] = 0;

}

break;

如代码所示，每个很长得if里得判断就是判断有没有走过以及是不是墙壁。

寻路算法与生成迷宫算法类似，用栈与并查集实现记录路径。

数据结构我采用的是结构，每个结构内有当前坐标的值以及上一个点的坐标的值

struct information {

int x;

int y;

int lastx;

int lasty;

};

用结构数组，数组的下标就是天然的栈，要回退，只需要在其他条件判断不成立的时候I - - ；

生成地图时，我先让用户输入长宽，然后默认1，1是入口，出口是右下角，默认全都是墙壁，然后开始挖通路，挖好一条通路后，再复用相同的代码，终点在地图中随机取几个，然后生成几条路径做干扰。迷宫就这样生成了。

寻找路径也类似，用相同的dfs算法寻路，如果进了死胡同就回退，寻找到终点后从终点不断得到自己上一个点的坐标，最后回到原点，打好标记。

然后就是输出，两层for循环打印，遇到路径标记就打印星号。程序结束。

所有的代码因为太长，又六百多行，所以粘在这的话显得不美观，所以我放在了Github上，源代码我也附送在了压缩包里，github上可以看到我每次修改了什么，每次增加了什么。改了哪里，删了哪里，都有高亮语法标注出来，非常方便。github.com:qinyi0411/Software-experiment-course.git

**八、总结及心得体会：**

通过本实验的练习，学会了并查集，以及设计了一个迷宫游戏，极大的提高了自己的编程能力，第一次编出了600行代码的程序，并且生成100 100的迷宫时电脑跑了十分钟还没有生成出来。并且很考验debug的耐心，很考验，代码量一大调试起来都很烦躁很无聊，但幸好最后没有bug了。

**九、对本实验过程及方法、手段的改进建议：**

本程序我还是有些不满意的，因为采用的时bfs算法，所以生成迷宫的时候挖的路是一块一块的，每次回退也只是一格一格的回退，甚至有些时候都挖过了，不剩什么障碍了，但又想不出办法，日后可以与同学讨论，学习其他同学的代码获得提升。

# 实 验 报 告 三

**学生姓名：覃翊 学 号： 2016060107008 指导教师：王丽杰**

**实验地点： 主楼B412 实验时间：12.7-12.21**

**一、实验室名称： 计算机学院实验中心A2-412**

**二、实验项目名称：文本压缩软件**

**三、实验学时：12学时**

**四、实验原理：**

任何文件都可以看作是由字节组成的字节块，将字节看作基本编码单元，一个文件就可以看作是由字节组成的信息串。对文件中各字节的出现频率进行统计，并以出现频率作为每个字节块的权值，就可以用字节为叶结点构造哈夫曼树，进而构造出各字节的对应哈夫曼编码。

对于源文件的字节编码是一种8位定长编码，而将各字节用哈夫曼编码进行重新编码，就有可能使得总的编码长度更短，这样将编码看做二进制位，每个8位作为一个字节写入压缩文件中，根据构建哈弗曼树的算法，在源文件中出现频率高的字符所对应的编码长度小，所以总的文件长度会小于原来的文件长度，从而达到文件压缩的效果。

**五、实验目的：**

了解哈夫曼树，增强动手能力，编程能力。学会用C语言实现一个文本压缩软件，了解树的存储与操作。理解二叉树的表示与存储；二叉树的基本操作。掌握哈夫曼算法；文件操作。

**六、实验内容：**

设计并实现一个基于哈夫曼树编码算法的英文文本编/解码器，其由编码器（compressor）对文件进行压缩编码，由解码器（decompressor）进行解码还原。

1. **实验步骤：**

  创建一个256长度的数组，来记录每一个ASCII码值出现的频率。

将要压缩的文件遍历一遍，统计一下每一个字符出现的频率。

然后调用求Huffman树的函数，建立一棵对应此文件字符出现的频率的Huffman树。

建立哈夫曼树的步骤：

第1步：根据给定的n个权值{w1,w2,……wn}，构造n棵只有根结点的二叉树，令其权值为wj；

第2步：在森林中选取两棵根结点权值最小的树作左右子树，构造一棵新的二叉树，置新二叉树根结点权值为其左右子树根结点权值之和

第3步：在森林中删除这两棵树，同时将新得到的二叉树加入森林中

第4步：重复2-3两步，直到只含一棵树为止，这棵树即哈夫曼树

求出对应每一个ASCII码的Huffamn编码。

然后再遍历一遍，每次读出一个字节，根据其ASCII码值找出对应的Huffman编码，如果Huffman编码长度不足8位时，就再从文件中读出下一个字符，并且得到他的Huffman编码，与之前的没有写入文件的编码作为一个整体；

如果此时的编码长度大于或等于8了，就将它对应的ASCII码值写入压缩文件中。

就这样边读源文件，边进行翻译写入压缩文件，直到读完源文件。

这样就完成了文件的压缩功能。

还要解压文件，所以在进行压缩的时候还应该把Huffman树一块保存起来，所以压缩文件的前面一部分就是压缩文件是建立的Huffman树，而压缩文件的开头四个字节用来记录文件的扩展名。

解压的时候，根据压缩文件的格式，读出源文件的扩展名，然后读出Huffman树，根据Huffman树进行解压。

解压时，同样可以一边解压一边写入解压后的文件中，即每读出一个字节，就根据他的ASCII码值，查找Huffman树对应的叶子节点，叶子节点的下标就是源文件对应的ASCII码值，然后将得到的ASCII码值写入压缩后的文件中，就这样直到读完解压文件，就完成了文件的解压工作。

分块代码如下：

建立哈夫曼树的节点结构：

typedef int ElemType;

typedef struct BTreeNode

{

ElemType data;

ElemType data2;

struct BTreeNode\* left;

struct BTreeNode\* right;

}BTreeNode;

创建哈夫曼树：

BTreeNode\* CreateHuffman(int a[][2], int n)

{

int i, j;

BTreeNode \*\*b, \*q=NULL;

b = (BTreeNode \*\*)malloc(n\*sizeof(BTreeNode));

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

{

b[i] = (BTreeNode \*)malloc(sizeof(BTreeNode));

b[i]->data = a[i][1];

b[i]->data2 = a[i][0];

b[i]->left = b[i]->right = NULL;

}

for (i = 1; i < n; i++)

{

//k1表示森林中具有最小权值的树根结点的下标，k2为次最小的下标

int k1 = -1, k2;

for (j = 0; j < n; j++) //让k1初始指向森林中第一棵树，k2指向第二棵

{

if (b[j] != NULL && k1 == -1)

{

k1 = j;

continue;

}

if (b[j] != NULL)

{

k2 = j;

break;

}

}

for (j = k2; j < n; j++) //构造最优解

{

if (b[j] != NULL)

{

if (b[j]->data < b[k1]->data)

{

k2 = k1;

k1 = j;

}

else if (b[j]->data < b[k2]->data)

k2 = j;

}

}

q = (BTreeNode \*)malloc(sizeof(BTreeNode));

q->data = b[k1]->data + b[k2]->data;

q->left = b[k1];

q->right = b[k2];

b[k1] = q;

b[k2] = NULL;

}

free(b);

return q;

}

解压函数：

BTreeNode\* unzip(BTreeNode \*node,char flag){

if(flag == 0)

return node->left;

if(flag == 1)

return node->right;

}

哈夫曼编码过程：

void HuffManCoding(BTreeNode\* FBT, int len,char HuffNode[][20])

{

static char a[20];

if (FBT != NULL)

{

if (FBT->left == NULL && FBT->right == NULL)

{

int i;

HuffNode[qj][0]=FBT->data2;

for(i=1;i<len+1;i++)

HuffNode[qj][i]= a[i-1];

HuffNode[qj][len+1]='|';

printf("ASCII码为%d的编码：",HuffNode[qj][0]);

for (i = 1; i < len+1; i++)

printf("%d", HuffNode[qj][i]);

printf("\n");

qj++;

}

else

{

a[len] = (00000000);

HuffManCoding(FBT->left, len + 1,HuffNode);

a[len] = (00000001);

HuffManCoding(FBT->right, len + 1,HuffNode);

}

}

}

Main函数：

int main(int argc, char \*argv[])

{

printf("-------------文本压缩软件-------------\n\t\t\t\t制作者：覃翊 2016060107008\n");

int data[256][2];

for(int i=0;i<256;i++)

for(int j=0;j<2;j++)

data[i][j]=0;

int n=0;int i=0;

char m;

freopen("data.txt","rb",stdin);

while(scanf("%c",&m)!=EOF){

int biaoji=0;

m=(int)m;

for(int j=0;j<=n;j++)

{

if(data[j][0]==m)

{

data[j][1]=data[j][1]+1;biaoji=1;break;

}

}

if(biaoji==0)

{

data[n][0]=m;data[n][1]=1;n++;

}

}

/\*

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

for(int j=0;j<2;j++)

printf("%d ",data[i][j]);

printf("\n");

\*/

fclose(stdin);

BTreeNode\* hfm;

hfm=CreateHuffman(data,n);

char HuffNode[256][20];

HuffManCoding(hfm,0,HuffNode);

/\*

for(int i=0;i<=25;i++)

for(int j=0;HuffNode[i][j]!='|';j++)

printf("%c ",HuffNode[i][j]);\*/

int length=0;

int length1=0;

FILE \*fpr = fopen("data.txt","rb");

FILE \*fpw = fopen("zip.txt","wb");

char zifu;

unsigned char savechar=0;

int num=0;

while((zifu=fgetc(fpr))!=EOF){

//printf("%c",zifu);

length++;

for(i=0;i<256;i++){ //变成int i=0后出现bug

if(zifu==HuffNode[i][0]) break;}

//printf("%d ",i);

for(int j=1;HuffNode[i][j]!='|';j++){

savechar=savechar+HuffNode[i][j];

//printf("%d ",savechar);

num++;

if(num==8){

fwrite(&savechar,sizeof(char),1,fpw);

savechar=0;

num=0;

length1++;

//printf("OK");

}

else{

savechar=savechar<<1;

}

}

}

if(num != 8){

savechar = savechar<<(7-num);

fwrite(&savechar,sizeof(char),1,fpw);

length1++;

}

fclose(fpr);

fclose(fpw);

int counttime = 20;

printf("\n");

printf("正在压缩，请稍等\n压缩进度：");

while(counttime > 0){

printf("■");

Sleep(counttime\*10);

counttime--;

}

printf("\n");

printf("\n压缩完成！压缩文件名为：zip.txt\n");

printf("\n下面开始解压\n");

int changdu=0;

FILE \*fpr1= fopen("zip.txt","rb");

// fseek(fpr1,0L,SEEK\_SET);

FILE \*fpw1 = fopen("unzip.txt","w");

unsigned char op=(10000000);unsigned char s;

BTreeNode \*z=hfm;

int x=0;

while(x!=length1){

zifu=fgetc(fpr1);

x++;

for(int k=0;k<8;k++){

s=op&zifu;

s=s>>7;

zifu=zifu<<1;

z=unzip(z,s);

if(z->left == NULL || z->right == NULL){

fprintf(fpw1,"%c",z->data2);

z=hfm;

changdu++;

}

if(changdu==length)break;

}

}

// printf("%d %d",length,changdu);

fclose(fpr1);

fclose(fpw1);

counttime = 20;

printf("\n解压中，请稍等\n解压进度：");

while(counttime > 0){

printf("■");

Sleep(counttime\*10);

counttime--;

}

printf("\n");

printf("\n解压完成！请到文件中查看解压后内容\n");

Sleep(8000);

return 0;

}

此外，我还添加了一些亮点功能，对用户友好的，就是压缩和解压时的进度条功能。

int counttime = 20;

printf("\n");

printf("正在压缩，请稍等\n压缩进度：");

while(counttime > 0){

printf("■");

Sleep(counttime\*10);

counttime--;

}

printf("\n");

printf("\n压缩完成！压缩文件名为：zip.txt\n");

counttime = 20;

printf("\n解压中，请稍等\n解压进度：");

while(counttime > 0){

printf("■");

Sleep(counttime\*10);

counttime--;

}

printf("\n");

printf("\n解压完成！请到文件中查看解压后内容\n");

进度条的加载时间都是不确定的，是逐渐加快的，看着很舒服。

**八、总结及心得体会：**

通过本实验的练习，学会了文件的读取和写入操作，深入了解了哈夫曼编码，哈夫曼树的创建，在编写程序的过程中积累了大量丰富的经验，加强了自身实力，对编程的理解。

曾压缩过一个300M的txt文件，压缩了很长时间，对计算机也是一个挑战，最后也能成功的解压了文件，程序跑了很久，成功时非常有成就感。

**九、对本实验过程及方法、手段的改进建议：**

1. 学习图片是怎么存储的，然后改动本程序，做一个压缩图片的软件。

2.将本程序分开，分成压缩软件与解压软件，并在压缩文件的前面加上校验码，即用本压缩程序所压缩出来的文件，必须用对应的解压软件才能解开，如果读入校验码不对，解压软件就不能进行，达到了保密的效果，比如，两个人在两个不同的地方，一方有压缩软件，一方有对应的解压软件，他们可以通过软件来传输压缩后的文件，即使文件中途被劫持了，也读不出来，因为只有通过唯一的解压软件才能将其解压。

**报告评分：**

**指导教师签字：**