|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 日期 | 学号 | 班级 | 姓名 | 今日完成情况总结 | 明日计划 |
| 2021-6-28 | 20192163 | 信1901-3 | 崔金泽 | 1学习prim算法 | 1学会查找aoe网 |
| 2编写prim算法 | 2学会topo排序 |
| 3学习关键路径算法 | 3编写关键路径代码 |
| 2021-6-29 | 20192163 | 信1901-3 | 崔金泽 | 1、编写关键路径代码 | 1、编写哈希生成代码 |
|  | 2、编写哈希查找代码 |
| 2021-6-30 | 20192163 | 信1901-3 | 崔金泽 | 1、编写关键路径算法 | 1、继续完成哈希查找 |
| 2、编写哈希算法 | 2、完成哈希成功查找ASL |
| 2021-7-1 | 20192163 | 信1901-3 | 崔金泽 | 1、编写完成了哈希算法 | 1、了解解压缩文件 |
| 2、编写完成了快速排序 |  |
| 2021-7-2 | 20192163 | 信1901-3 | 崔金泽 | 1、学习了哈夫曼树的排列方法 | 1、编写哈夫曼生成树 |
| 2、学习讲文件中文本进行哈夫曼编码 | 2、编写哈夫曼编码 |
| 2021-7-3 | 20192163 | 信1901-3 | 崔金泽 | 1、编写压缩功能函数代码 | 1.编写解压缩的代码 |
| 2021-7-4 | 20192163 | 信1901-3 | 崔金泽 | 1.编写解压缩的代码 | 准备验收 |
| 2021-7-5 | 20192163 | 信1901-3 | 崔金泽 | 1、验收 | 寻找bug，完善代码 |
| 2、完善解压缩功能 |  |
| 3、使解压缩出来的文件可读 |  |
| 2021-7-6 | 20192163 | 信1901-3 | 崔金泽 | 1、完成实验报告 | 2、完善实验报告 |
| 2021-7-7 | 20192163 | 信1901-3 | 崔金泽 | 1、整理归纳上交实验报告。 |  |

**《算法与数据结构综合训练》**

**实验一数据结构基本算法演示程序**

**实习报告**

**班级：信1901-3**

**学号：20192163**

**姓名：崔金泽**

**小组成员：陈顺鹏、韩英杰、杨子腾、崔金泽**

**日期：2021.07.06**

目 录

[Prim算法 3](#_Toc76456258)

[1.任务说明 3](#_Toc76456259)

[2.概要设计 4](#_Toc76456260)

[3.详细设计 列出主程序清单。（子程序略） 5](#_Toc76456261)

[4.测试 6](#_Toc76456263)

[关键路径算法 7](#_Toc76456264)

[1.任务说明 7](#_Toc76456265)

[2.概要设计 7](#_Toc76456266)

[3.详细设计 列出主程序清单。（子程序略） 9](#_Toc76456267)

[4.测试 15](#_Toc76456268)

[哈希表生成及哈希查找算法 15](#_Toc76456269)

[1.任务说明 15](#_Toc76456270)

[2.概要设计 16](#_Toc76456271)

[3.详细设计 列出主程序清单。（子程序略） 16](#_Toc76456272)

[4.测试 20](#_Toc76456273)

[快速排序 21](#_Toc76456274)

[1.任务说明 21](#_Toc76456275)

[2.概要设计 21](#_Toc76456276)

[3.详细设计 列出主程序清单。（子程序略） 22](#_Toc76456277)

[4.测试 24](#_Toc76456278)

[心得体会 24](#_Toc76456279)

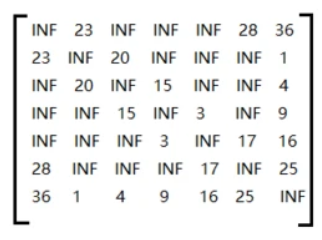
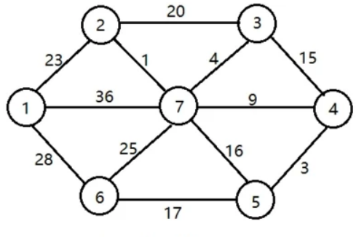
### Prim算法

1.任务说明

输入：无向图（顶点序列，边序列）

功能要求：输出最小生成树的各组成边及最小生成树的权值

2.概要设计

抽象数据类型：

lowcost[i]:表示以i为终点的边的最小权值,当lowcost[i]=0说明以i为终点的边的最小权值=0,也就是表示i点加入了MST

closest[i]:表示对应lowcost[i]的起点，即说明边<mst[i],i>是MST的一条边，当mst[i]=0表示起点i加入MST

所有点默认起点是V1



初始化：选中23并将节点2加入集合U，将2号节点加入集合U，和它邻接集合V-U中的节点是3和7。更新节点3和7：  
c2 = 20 < lowcost[3] = INF,更新lowcost[3] = 20,同时更新closest[3] = 2;  
c2 = 1 < lowcost[7] = 36, 更新lowcost[7] = 1,同时更新closest[7] = 2;

图的抽象类型定义

Graph{

数据对象：V是具有相同特性的数据元素的集合，称为顶点集；

数据关系：R={VR}

VR={<v,w>|v,w∈V且P(v,w)<v,w>表示从v到w的弧}

谓词P（v,w）定义了弧<v,w>的意义与信息}

基本操作：

Init

操作结果：创建无向图

Prim(&G,u)

初始条件：无向图G已存在。

操作结果：从顶点u出发生成最小生成树并运算返回结果

}

（2）本程序共包含 3函数

1、主函数main()

2、Prim()

3、Init()

各函数之间的调用关系如下：

main() Init() Prim()

3.详细设计 列出主程序清单。（子程序略）

#include<iostream>

#include<fstream>

using namespace std;

#define MAX 1000

#define MAXCOST 10000

int graph[MAX][MAX];

int prim(int graph[][MAX], int n)

{

int lowcost[MAX];

int mst[MAX];

int i, j, min, minid, sum = 0;

for (i = 2; i <= n; i++)

{

lowcost[i] = graph[1][i];

mst[i] = 1;

}

mst[1] = 0;

for (i = 2; i <= n; i++)

{

min = MAXCOST;

minid = 0;

for (j = 2; j <= n; j++)

{

if (lowcost[j] < min && lowcost[j] != 0)

{

min = lowcost[j];

minid = j;

}

}

cout << "V" << mst[minid] << "-V" << minid << "=" << min << endl;

sum += min;

lowcost[minid] = 0;

for (j = 2; j <= n; j++)

{

if (graph[minid][j] < lowcost[j])

{

lowcost[j] = graph[minid][j];

mst[j] = minid;

}

}

}

return sum;

}

int main()

{

int i, j, k, m, n;

int x, y, cost;

cout << "无向图的顶点数和边数(空格隔开)：" << endl;

cin >> m >> n;

//初始化图G

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

{

for (j = 1; j <= m; j++)

{

graph[i][j] = MAXCOST;

}

}

cout << "输入" << m << "条边的开始，结束节点，权值(空格隔开):" << endl;

//构建图G

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

{

cin >> i >> j >> cost;

graph[i][j] = cost;

graph[j][i] = cost;

}

//求解最小生成树

cost = prim(graph, m);

//输出最小权值和

cout << "最小权值和=" << cost << endl;

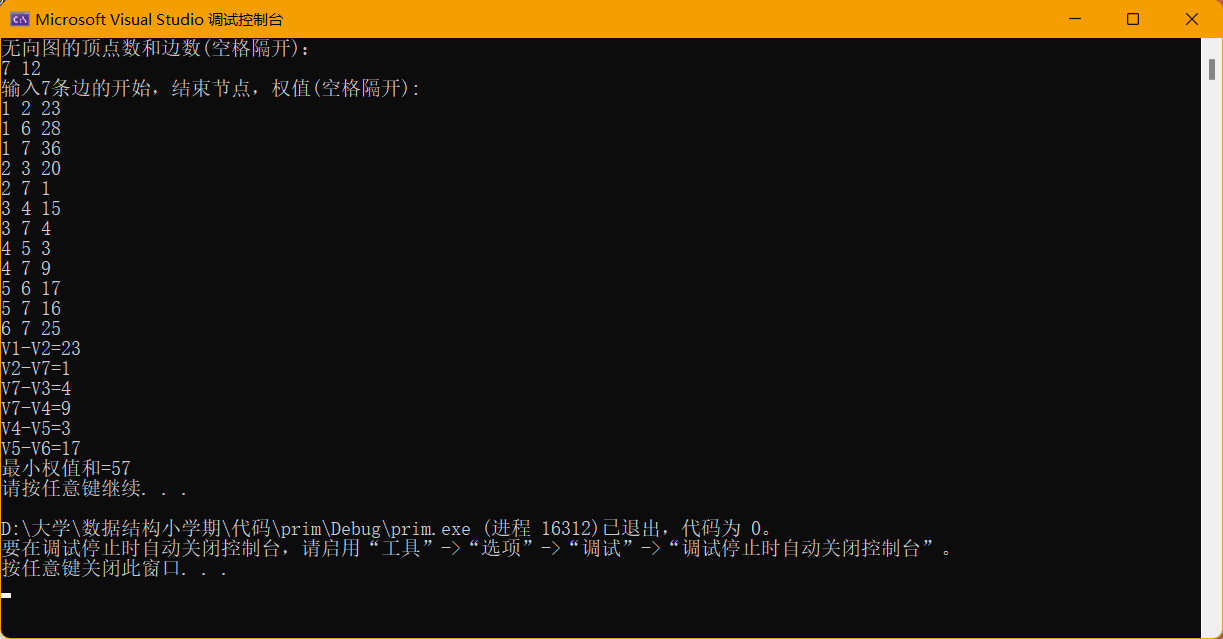
system("pause");

return 0;

}

4.测试

rim算法遍历了每一个点，符合条件的点进行判断是否有符合条件的边，其中用优先队列优化了获取最短边的操作，时间复杂度为ELogV，空间复杂度为 n \* n。



### 关键路径算法

1.任务说明

8、关键路径算法

输入：有向图（顶点序列，有向边序列）

功能要求：能判断是否是AOE 网；输出各关键活动或输出关键路径（包括关键路径的长度）

2.概要设计

抽象数据类型 ：

（1）为了实现上述程序功能，需要定义栈、有向图的抽象数据类型：

ADT Stack{

数据对象：D={ai|ai∈ElemSet,i=0,1,2,…,n,n≥0}

数据关系：R={<ai,ai+1>|ai,ai+1∈D,i=0,1,2,…,n}

基本操作：

IntiStack(&s)

操作结果：初始化一个空的栈；

Push(&s,e)

初始条件：栈s已存在

操作结果：插入元素e为新的栈顶元素

Pop(&s,e)

初始条件：栈s已存在

操作结果：删除s的栈顶元素，并用e返回其值

}

ADT OLGraph{

数据对象：D={ai|ai∈ElemSet,i=0,1,2,…,n,n≥0}

数据关系：R={<ai,ai+1>|ai,ai+1∈D,i=0,1,2,…,n}

基本操作：

CreateOLGraph(&G)

操作结果：采用邻接表示法，创建有向图G

FindIndegree(G,indegree[])

初始条件：有向图G存在，数组indegree存在

操作结果：求出有向图G中各顶点的入度存入数组indegree中

LocateVex(G,u)

初始条件：有向图G存在，u和G中顶点有相同特征

操作结果：若G中存在定点u则返回改顶点在图中的位置

TopologicalSort(G,topo[])

初始条件：存在有向图G并且存在数组topo[]

操作结果：对有向图进行拓扑排序，判断有向图G是否为AOE-网，并且将出栈顶点保存在topo拓扑数组中

CriticalPath(G)

初始条件：存在有向图G

操作结果：通过关键路径算法计算出关键路径并且将关键路径输出

CriticalPath()

操作结果：完成总的输入输出

1. 本模块包含9个函数

功能调用函数Critical\_Path()

初始化栈IntiStack()

入栈Push()

出栈Pop()

邻接矩阵表示法创建有向图CreateOLGraph()

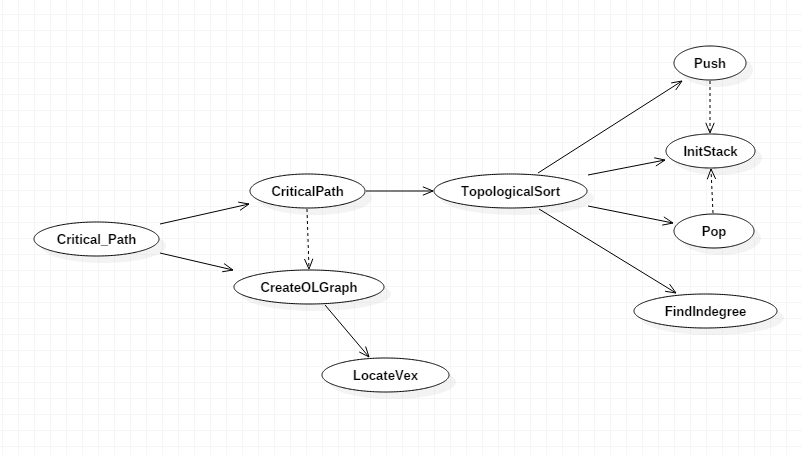
返回图中顶点位置 LocateVex()

计算每个顶点入度 FindIndegree()

拓扑排序函数 TopologicalSort()

关键路径算法 CriticalPath()

各函数关系为：



3.详细设计 列出主程序清单。（子程序略）

head.h

#include <iostream>

#include <cstring>

#include <cstdio>

#include <cctype>

#include <stack>

#include <string>

using namespace std;

#define MAXSIZE 100 //顺序栈存储空间的初始分配量

#define MVNum 100 //最大顶点数

源文件.cpp

#include "head.h"

struct Stack

{

int\* base;

int\* top;

int stacksize;

};

//初始化栈

void IntiStack(Stack& s)

{

s.base = new int[MAXSIZE];

s.top = s.base;

s.stacksize = MAXSIZE;

}

//入栈

void Push(Stack& s, int e)

{

if (s.top - s.base != s.stacksize) //栈未满

\*s.top++ = e; //元素e压入栈顶，栈顶指针加1

}

//出栈

void Pop(Stack& s, int& e)

{

if (s.top != s.base)

e = \*--s.top; //栈顶指针减1，将栈顶元素赋给e

}

/\*--------------------图的邻接表存储表示--------------------\*/

//边结点

struct ArcNode

{

int adjvex; //该边所指向的顶点的位置

ArcNode\* nextarc;

int weight; //设置边的权值类型为整型

};

//顶点(表头结点)信息

struct VNode

{

char data; //定义结点的数据类型为char型

ArcNode\* firstarc; //指向第一条依附该顶点的边的指针

};

//定义图

struct OLGraph

{

VNode vertices[MVNum];

int vexnum, arcnum; //图的当前顶点数和边数

};

//返回在图G中顶点U的位置

int LocateVex(OLGraph G, VNode u)

{

for (int i = 0; i < G.vexnum; i++)

{

if (G.vertices[i].data == u.data)

return i; //第一个点从0开始算起

}

}

/\*--------------------采用邻接表创建有向图--------------------\*/

void CreateGraph(OLGraph& G)

{

cout << "请依次输入总顶点数和总边数：";

cin >> G.vexnum >> G.arcnum;

//输入各点，构造表头结点表

cout << "请依次输入各结点的名称（用空格作为间隔）：";

for (int i = 0; i < G.vexnum; i++)

{

cin >> G.vertices[i].data;

G.vertices[i].firstarc = NULL; //初始化表头结点的指针域为NULL

}

//输入各边，构造邻接表

for (int i = 0; i < G.arcnum; i++)

{

VNode v1, v2;

int weight;

cout << "请输入第" << i + 1 << "个边的起始点、终止点和权值：";

cin >> v1.data >> v2.data >> weight;

int m = LocateVex(G, v1);

int n = LocateVex(G, v2);

ArcNode\* p = new ArcNode;

p->adjvex = n; //邻接点序号为n

p->weight = weight;

//将结点P插入到顶点Vm的后边的第一个

p->nextarc = G.vertices[m].firstarc;

G.vertices[m].firstarc = p;

}

}

//求出各顶点的入度存入数组indegree中

void FindIndegree(OLGraph G, int indegree[])

{

for (int i = 0; i < G.vexnum; i++)

{

indegree[i] = 0;

ArcNode\* p = new ArcNode;

for (int j = 0; j < G.vexnum; j++)

{

p = G.vertices[j].firstarc;

while (p != NULL)

{

if (p->adjvex == i) (indegree[i])++;

p = p->nextarc;

}

}

}

}

/\*--------------------拓扑排序判断是否为AOE-网--------------------\*/

bool TopologicalSort(OLGraph G, int topo[])

{

int\* indegree = new int[G.vexnum];

Stack s;

int i;

FindIndegree(G, indegree); //求出各顶点入度

IntiStack(s);

for (i = 0; i < G.vexnum; i++)

if (!indegree[i]) Push(s, i);

int m = 0; //对输出顶点计数

while (s.top != s.base) //栈非空

{

Pop(s, i); //栈顶顶点出栈

topo[m] = i; //出栈顶点保存在topo拓扑数组中

++m;

ArcNode\* p = G.vertices[i].firstarc;

while (p != NULL)

{

int k = p->adjvex;

indegree[k]--; //每一个邻接点入度减1

if (indegree[k] == 0) Push(s, k); //入度减为0则入栈

p = p->nextarc;

}

}

if (m < G.vexnum) return false;

else return true;

}

/\*--------------------关键路径算法实现--------------------\*/

void CriticalPath(OLGraph G)

{

int topo[MAXSIZE];

int\* ve = new int[G.vexnum];

int\* vl = new int[G.vexnum];

cout << endl;

if (!TopologicalSort(G, topo)) //若不是AOE-图，直接退出

{

cout << "该有向图不是AOE-网。" << endl;

return;

}

else

cout << "该有向图是AOE-网。" << endl;

int n = G.vexnum;

int i;

for (i = 0; i < n; i++) ve[i] = 0; //初始化最早发生时间为0

//按拓扑排序次序求每个事件发生的最早时间

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

{

int k = topo[i];

ArcNode\* p = G.vertices[k].firstarc;

while (p != NULL)

{

int j = p->adjvex;

if (ve[j] < ve[k] + p->weight)

ve[j] = ve[k] + p->weight;

p = p->nextarc;

}

}

for (i = 0; i < n; i++) vl[i] = ve[n - 1]; //初始化每件事情发生的最早时间是ve[n-1]

//按拓扑排序的逆序求每件事情发生的最迟时间

for (i = n - 1; i >= 0; i--)

{

int k = topo[i];

ArcNode\* p = G.vertices[k].firstarc;

while (p != NULL)

{

int j = p->adjvex;

if (vl[k] > (vl[j] - p->weight))

vl[k] = vl[j] - p->weight;

p = p->nextarc;

}

}

//判断每一个事件是否为关键活动

cout << endl;

cout << "该AOE-图中的关键路径信息如下：" << endl;

char map[100], map1[100];

int x = 0, dat = 0;

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

{

ArcNode\* p = G.vertices[i].firstarc;

while (p != NULL)

{

int j = p->adjvex;

int e = ve[i];

int l = vl[j] - p->weight;

if (e == l)

{

cout << G.vertices[i].data << "->" << G.vertices[j].data << " 路径长度为：" << p->weight << endl;

map[x] = G.vertices[i].data;

//cout << map[x] << endl;

map1[x] = G.vertices[j].data;

dat = dat + p->weight;

//cout << map1[x] << endl;

x++;

}

p = p->nextarc;

}

}

int t = 0;

cout << endl;

while (t < x - 1)

{

cout << map[t] << "->";

t++;

}

cout << map[t] << "->" << map1[t] << " 路径长度为：" << dat << endl;

}

void Critical\_Path()

{

OLGraph G;

CreateGraph(G);

CriticalPath(G);

}

4.测试

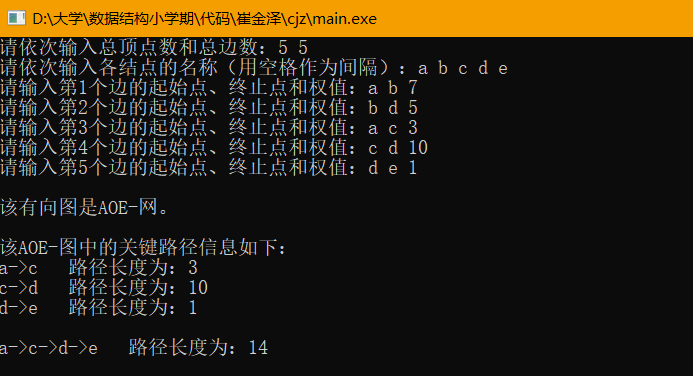
1）关键路径算法

① 输入的形式和输入值的范围： 有向图的顶点数和边数，有向图（顶点序列，有向边序列）。先通过拓扑排序判断有向图是否为有向无环的AOE-网，如果是AOE-网就可以进行关键路径算法，否则无法进行关键路径的算法。

② 输出的形式：能判断是否是AOE-网，输出个关键路径。

③ 程序所能达到的功能：在用户输入有向图之后，程序能够通过拓扑排序判读那是否为AOE-网，如果是AOE-网就通过关键路径算法计算之后输出该有向图中的关键路径。

1. 测试数据：



### 哈希表生成及哈希查找算法

1.任务说明

10、哈希表生成及哈希查找算法

输入：待哈希数据序列

功能要求：输出哈希方法和解决冲突的方法（文字输出），输出哈希表概要设计

2.概要设计

**2.哈希表**

（1）为了实现上述程序功能，需要定义哈希表的抽象数据类型：

ADT hushtable{

数据对象：D={ai|ai∈ElemSet, i=1,2,...n, n≥0}

数据关系：R1={<ai-1,ai>|ai-1∈D, i=1,2,...n}

基本操作：

CompASL(hushtable\* ha)

初始条件：已经建立哈希表

操作结果：输出哈希表的平均查找长度

Ser (hushtable\* m, string x)

初始条件：哈希表已创建

操作结果：返回所给数值的位置

showtable(hushtable\* ha)

初始条件：哈希表已创建

操作结果：输出各个表项和他们在表中的位置

menuHush()

操作结果：展示函数功能选项菜单

Hush()

操作结果：创建哈希表并调用各个函数

（2）本模块包含5个函数

主体函数Hush()

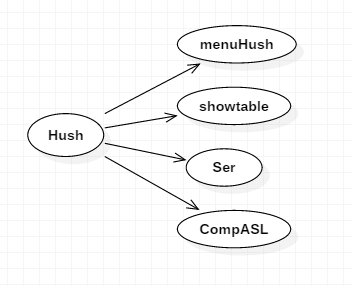
菜单函数menuHush()

哈希表展示函数showtable()

查找元素函数Ser()

平均查找长度CompASL()

函数关系：



3.详细设计 列出主程序清单。（子程序略）

Head.h

#include <iostream>

#include <cstring>

#include <cstdio>

#include <cctype>

#include <stack>

#include <string>

using namespace std;

#define M 40 //哈希表长度

#define D 37 //哈希表中的待除余

#define NULLKEY "/wu" //哈希表中没有内容的表

/\*-----------------------------------哈希表---------------------------------------\*/

bool iscreate;

struct hushtable {

string key; //关键字域

int count; //探查次数域

};

int Ser(hushtable\* m, string x)

{

int H0, pos = x[0] + x[1];

int Hi;

H0 = pos % D;

if (m[H0].key == NULLKEY)

return -1;

else if (m[H0].key == x)

return H0;

else

{

for (int i = 1; i < M; ++i)

{

Hi = (H0 + i) % M;

if (m[Hi].key == NULLKEY) return -1;

else if (m[Hi].key == x) return Hi;

}

return -1;

}

}

void CompASL(hushtable\* ha)

{

int i;

int s = 0, n = 0;

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

if (ha[i].key != NULLKEY)

{

s = s + ha[i].count;

n++;

}

printf(" 查找成功的ASL=%.3g\n", s \* 1.0 / n);

}

void showtable(hushtable\* ha)

{

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

if (ha[i].key != NULLKEY)

{

cout << "位置:" << i << "," << "值:" << ha[i].key << " ";

}

cout << endl;

}

void menuHush()

{

cout << endl;

cout << "请选择以下功能：" << endl;

cout << "1.创建哈希表" << endl;

cout << "2.查找某个名字" << endl;

cout << "3.输出查找成功时的平均查找长度" << endl;

cout << "4.退出程序" << endl;

}

int Hush()

{

hushtable r[M];

int m;

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

{

r[i].key = NULLKEY;

}

menuHush();

cin >> m;

while (m != 4)

{

if (m == 1)

{

int ra;

cout << "请输入要创建的表的数据个数" << endl;

cin >> ra;

for (int m = 0; m < ra; m++)

{

string x;

cout << "请依次输入要放入的数据" << endl;

cin >> x;

int H0, pos = x[0] + x[1];

cout << "pos:" << pos << endl;

int Hi;

H0 = pos % D;

cout << "H0:" << H0 << endl;

if (r[H0].key == NULLKEY)

{

r[H0].key = x;

r[H0].count++;

}

else

for (int i = 1; i < M; i++)

{

Hi = (H0 + i) % M;

cout << "Hi:" << Hi << endl;

if (r[Hi].key == NULLKEY)

{

r[Hi].key = x;

r[H0].count++;

break;

}

else if (r[Hi].key != NULLKEY)

{

r[H0].count++;

}

}

}

iscreate = true;

showtable(r);

cout << "创建完成" << endl;

}

else if (m == 2 && iscreate)

{

string ra;

cout << "请输入要查找的元素：" << endl;

cin >> ra;

if (Ser(r, ra) != -1)

cout << "你查找的元素在" << Ser(r, ra) << "位置" << endl;

else

cout << "你查找的元素不存在" << endl;

}

else if (m == 3 && iscreate)

{

CompASL(r);

}

menuHush();

cin >> m;

}

return 0;

}

4.测试

2）哈希表算法

① 输入的形式和输入值的范围：

创建哈希表时：输入创建的数据个数和各个放入表中的数据；

查找哈希表时：输入想要查找的数据；

求平均查找长度时：在查找后选择此项功能即可。

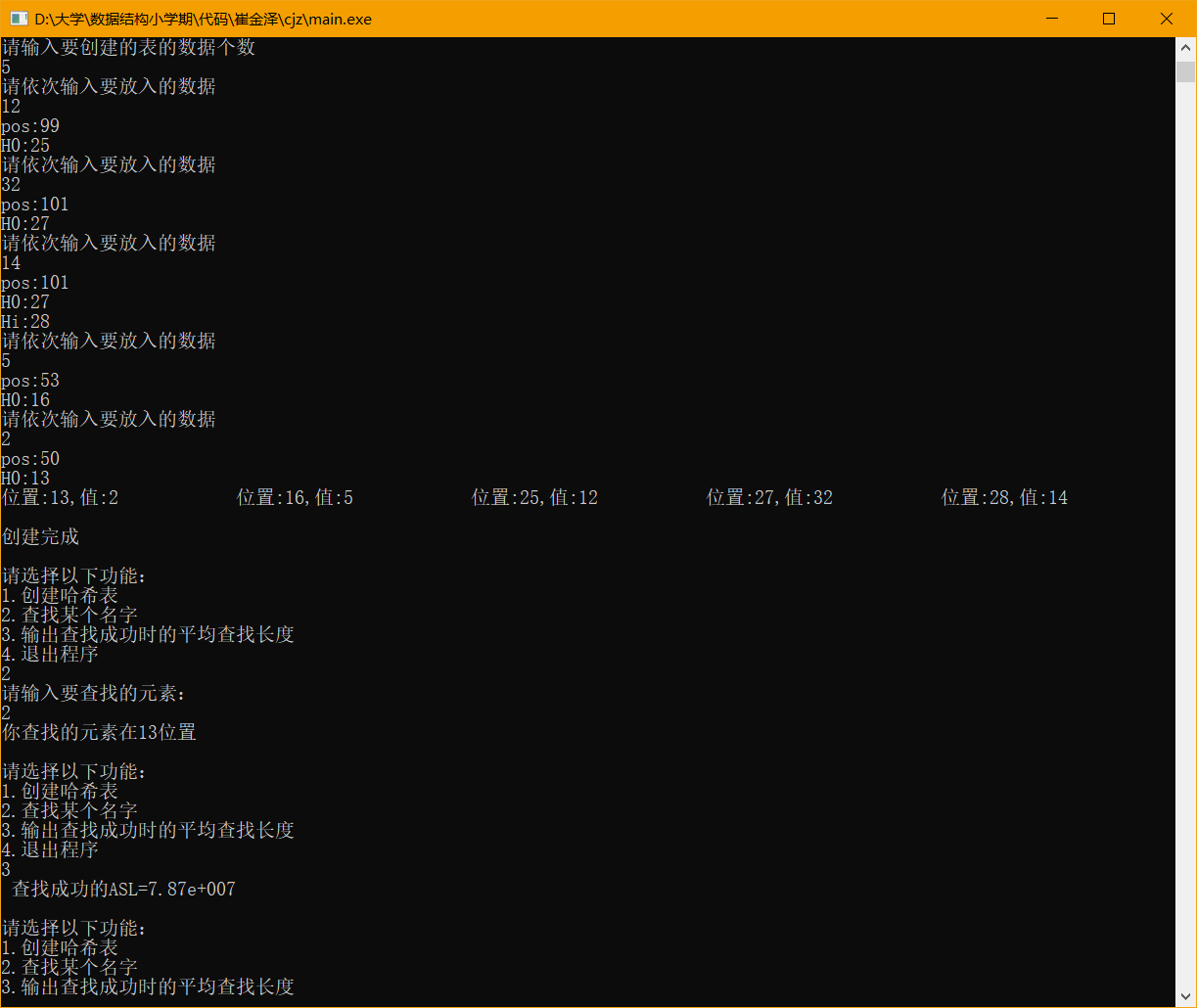
② 输出的形式：

创建哈希表时：每次插入会展示运算过程，最后提示创建成功，并输出所有位置；

查找哈希表时：无此数据会提示，有此数据会提示查找成功并输出位置；

求平均查找长度时：输出ASL为多少。

1. 程序所能达到的功能：创建哈希表并查找其中的数据，同时在查找成功后输出ASL。
2. 测试数据：



### 快速排序

1.任务说明

12、快速排序

输入：待排序数据序列

功能要求：输出每步骤的枢轴选择和排序情况；希望能进行排序方向的选择（从大到小或从小到大）

2.概要设计

假设一开始序列{xi}是：5，3，7，6，4，1，0，2，9，10，8 初始ref=5从后往前找，第一个比5小的数是x8=2，因此序列为：2，3，7，6，4，1，0，5，9，10，8。

此时i=1，j=8，从前往后找，第一个比5大的数是x3=7，因此序列为：2，3，5，6，4，1，0，7，9，10，8

此时，i=3，j=8，从第8位（7）往前找，第一个比5小的数是x7=0，因此：2，3，0，6，4，1，5，7，9，10，8

此时，i=3，j=7，从第3位往后找，第一个比5大的数是x4=6，因此：2，3，0，5，4，1，6，7，9，10，8

此时，i=4，j=7，从第7位往前找，第一个比5小的数是x6=1，因此：2，3，0，1，4，5，6，7，9，10，8

此时，i=4，j=6，从第4位往后找，直到第6位才有比5大的数，这时，i=j=6，ref成为一条分界线，它之前的数都比它小，之后的数都比它大，对于前后两部分数，可以采用同样的方法来排序

数组类型定义

ADT quickSortPartition {

数据对象：D={ai|ai∈IntegerSet,i=0,1,2,…,n,n≥0}

数据关系：R={<ai,ai+1>|ai,ai+1 ∈D}

}

本程序共包含 4函数

1、主函数main()

2、P() 数组打印

3、quickSortPartition () 对数组进行快速排序

4、quickSort 划分数组

各函数之间的调用关系如下：



3.详细设计 列出主程序清单。（子程序略）

#include <iostream>

using namespace std;

//数组打印

void P(int a[], int n)

{

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

cout << a[i] << " ";

cout << endl;

}

int quickSortPartition(int s[], int l, int r) {

int i = l, j = r, x = s[l]; //将最左元素记录到x中

while (i < j)

{

// 从右向左找第一个<x的数

// 不考虑下标越界

while (i < j && s[j] >= x)

j--;

if (i < j)

s[i++] = s[j]; //直接替换掉最左元素（已在x中存有备份）

// 从左向右找第一个>x的数

while (i < j && s[i] <= x)

i++;

if (i < j)

//替换掉最右元素(已在最左元素中有备份）

//最左元素一定被覆盖过，若没有，则表明右侧所有元素都>x，那么算法将终止

s[j--] = s[i];

}

s[i] = x; //i的位置放了x，所以其左侧都小于x，右侧y都大于x

return i;

}

void quickSort(int s[], int l, int r)

{

//数组左界<右界才有意义，否则说明都已排好，直接返回即可

if (l >= r) {

return;

}

// 划分，返回基准点位置

int i = quickSortPartition(s, l, r);

// 递归处理左右两部分，i处为分界点，不用管i了

quickSort(s, l, i - 1);

quickSort(s, i + 1, r);

}

int main()

{

int m,i;

cout << "请输入想要输入的数据个数：";

cin >> m;

int a[10];

cout << "请输入想要输入的数据：";

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

cin >> a[i];

}

P(a, m);

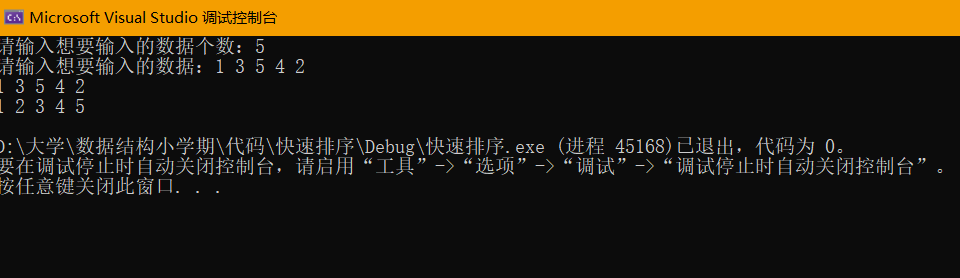
quickSort(a, 0, m-1);//最后一个参数是n-1

P(a, m);

return 0;

}

4.测试



### 心得体会

本次四个算法的内容在选择上考虑到了各种数据结构和思想均有涉及，选择了一个图相关的关键路径，查找相关的哈希表，排序相关的快速排序，最小生成树的普利姆算法。因为哈希表的整体架构在大二第一学期中的实验中有所涉猎，只是在原先的基础添加一些内容，并没有遇到什么困难，主要的困难在于关键路径，虽然说书上都有代码，但c和c++还是有大量不同，在编译时也会遇到很多问题，比如书上中缀表达式逐个获得字符用的getchar()但在c++中必须要使用cin才能避免莫名其妙发生的运算字符读两次的问题，关键路径上是否有环的判断也在网上参考了很多资料。

编写了四个算法，并体验了组内其他组员的算法，最后还汇总了大家的算法，大家给了我很多的帮助，结构体冲突以及版本不兼容的问题，意识到我们在程序设计正确性、健壮性、完整性、可用性、[可移植性](https://www.baidu.com/s?wd=%E5%8F%AF%E7%A7%BB%E6%A4%8D%E6%80%A7&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao)、可再用性、互运行性上的不足，程序漏洞和死角很多，还很不完善，未来也会继续改进。同时也温习了算法和数据结构的思想，提升了许多。

石家庄铁道大学课程设计

压缩解压文件

**二○一九 届 信息科学与技术 学院**

**专 业 计算机科学与技术**

**学 号 20192163**

**学生姓名 崔金泽**

**指导教师 陈娜**

**完成日期 2021 年7 月7 日**

摘 要

我们都知道根据人类使用文字对应的每个字符都是有特定的频率的，比如说英文，一般来说字母a或者e的使用频率很高。如果我们能给出现频率最高的字符很短的编码，出现最少的字符最长的编码，而且保证每个编码都不是任意一个编码的前缀码，不会产生歧义，这样能保证根据压缩的编码来解压不会出现错误。如果我们用这样的编码来对一个文件进行压缩，那么出现最多的字符将会有最短的编码，可以节省很多的空间，将一个大的文件压缩成一个比较小的文件。而哈夫曼树就是一个可以用来生成这样一种编码的树，以树的叶子节点代表一个字符，从根节点到叶节点的路径形成一个编码，左子树表示比特1，右子树表示比特0，可以交换。这样每个字符对应一个唯一的编码而且任意一个编码都不是其他编码的前缀码。如果我们要压缩一个文件，需要先统计这个字符文件中每个字符出现的频率，然后根据频率来生成一颗哈夫曼树，使频率最高的字符对应于最短的编码。

**关键词**：哈夫曼编码 ;文件的解压缩;数据结构算法

目 录

[第一章 引言 2](#_Toc76452094)

[第二章 系统功能需求分析 2](#_Toc76452095)

[2.1 问题描述: 2](#_Toc76452096)

[2.2 需求分析 3](#_Toc76452097)

[3.1 抽象数据类型(ADT) 3](#_Toc76452098)

[第四章 详细设计 5](#_Toc76452099)

[4.1 重要程序段1 6](#_Toc76452100)

[4.2 重要程序段2 11](#_Toc76452101)

[4.3 重要程序段3 14](#_Toc76452102)

[4.4 重要程序段4 15](#_Toc76452103)

[第五章 调试分析 17](#_Toc76452104)

[第六章 运行结果 18](#_Toc76452105)

## 第一章 引言

数据结构是计算机存储、组织数据的方式。数据结构是指相互之间存在一种或多种特定关系的数据元素的集合。通常情况下，精心选择的数据结构可以带来更高的运行或者存储效率的算法。数据结构往往同高效的检索算法和索引技术有关。 (简要介绍数据结构课程)

本次课程设计，是利用哈夫曼树来实现压缩解压文件。

本文档从概要设计，详细设计，参考源程序代码以及显示程序运行的截屏等方面详尽的介绍了软件开发过程。

## 第二章 系统功能需求分析

2.1 问题描述:

实现一个基于哈夫曼树的文件压缩程序和文件解压程序。

基本要求：

(1) 要求压缩程序读入源文件，分析每种字符的频度，然后建立相应的哈夫曼树，

再求出相应哈夫曼编码，根据编码对源文件进行压缩，得到源文件对应的压缩文

件。

(2) 解压程序读入压缩文件，根据相应的哈夫曼编码解压还原，得到对应的源文件。

(3) 求出压缩率；

2.2 需求分析

压缩:对任意文件进行压缩。

解压缩:对以压缩的文件进行解压缩。

加密:对任意文件进行加密。(MD5加密)

解密:对以解密的文件进行解密。

第3章 概要设计

3.1 抽象数据类型(ADT)

**ADT Huffman**

{ **数据对象：D={ai|ai∈ElemTtpe,i=1,2,…,n,n≥0}**

**数据关系：R={<ai-1,ai>|ai-1,ai∈D，i=1，2，…,n}**

**基本操作：**

(1)Status creat(T a[], int size);

操作结果：创建哈夫曼树

(2) Status print();

初始条件：haffman树已存在

操作结果：输出哈夫曼树

(3) Status preOrder();

初始条件：haffman树已存在

操作结果：前序遍历哈夫曼树

(4) Status inOrder();

初始条件：haffman树已存在

操作结果：中序遍历哈夫曼树

(5) Status postOrder ();

初始条件：haffman树已存在

操作结果：后序遍历哈夫曼树

(6) Status destory();

初始条件：haffman树已存在

操作结果：删除哈夫曼树

}

3.2 存储结构

struct HuffmanTreeNode

{

HuffmanTreeNode<W>\* \_left;

HuffmanTreeNode<W>\* \_right;

HuffmanTreeNode<W>\* \_parent;

W \_w;//权值，也就是哈夫曼树的叶子节点的数据

HuffmanTreeNode(const W& w)

:\_left(NULL)

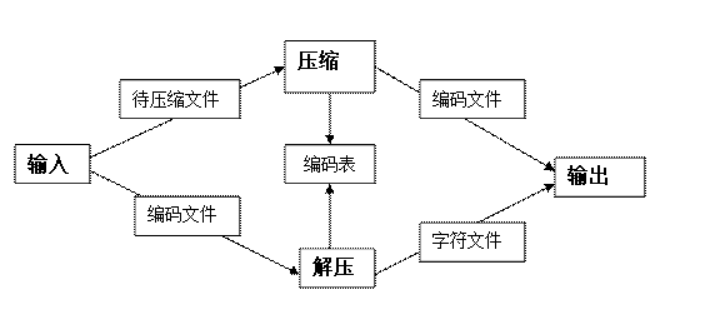
,\_right(NULL)

,\_parent(NULL)

,\_w(w)

{}

};



## 第四章 详细设计

项目主要思路：

1.统计：首先读取一个文件，统计出要压缩的文件中256个字符中各字符出现的次数，利用数据结构中的小堆来构建Huffman树，每次在堆顶选出两个出现次数较小的字符先进行构建，将它们相加的结果又放进堆里面，直到堆里面的数据被取完，这样字符出现次数多的离Huffman树的根节点就比较近，字符出现次数少的离Huffman树的根节点就比较远。

2.建树：根据构建好的Huffman树写出Huffman code，规定向左为“0”向右为“1”，这样字符出现次数多的Huffman code较短，字符出现次数少的Huffman code较长。

3.压缩：再次读取原文件，按照该字符对应的编码压缩文件；Huffman code刚好为“0”、“1”序列，所以可以每8位为一个字节写进压缩文件中，当余下的位数不够8位写不成一个字节时，需要在后面补0够一个字节再写入，这样就完成了文件的压缩，但要注意后面补0的位数为文件解压缩留下了隐患。

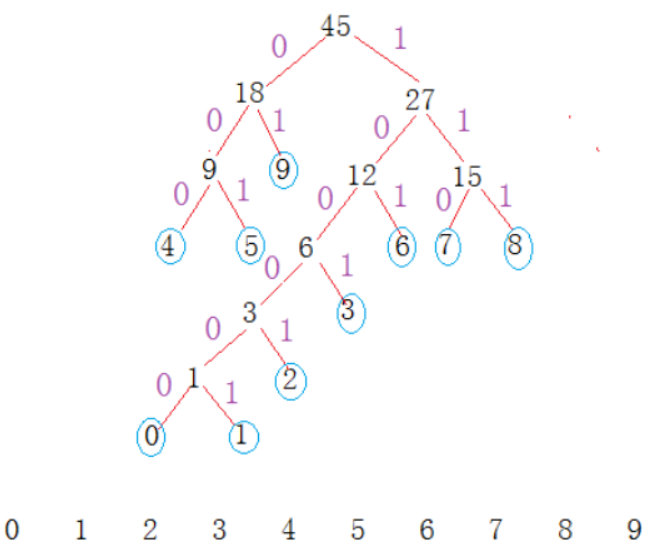
4.加工：将字符出现的总次数以及各个字符出现的次数写进配置文件，因为我们并不知道在进行压缩的时候在后面到底补了多少位0。

5.解压：利用压缩文件以及配置文件恢复原文件；

6.测试：首先对压缩和解压缩两个功能进行测试，看压缩后的文件是否比原文件小，以及解压缩文件与原文件的大小一致；在这个基础上再验证解压缩文件与原文件的内容是否保持一致，这里采用 beyond compare 软件来对文件内容进行验证，这样才验证了程序的大概功能是正确的。其次是压缩不同大小的文件计算出压缩所需要的时间来考量压缩的性能；

7.计算：经过多次压缩不同大小的文件不同类型的文件来计算文件的压缩率；

8.时间：使用 GetTickcount 函数计算完成压缩或者解压缩所需要的时间，GetTickcount 函数它返回从操作系统启动到当前所经过的毫秒数，常常用来判断某个方法执行的时间。



0->100000

1->100001

2->10001

3->1001

4->000

5->001

6->101

7->110

8->111

9->01

由图可以看到，权值越小的，它的Huffman编码越长，权值越大的，它的Huffman编码越短

假设一个文件，假设一个文件的内容是“abbcccdddd”，“a”出现的次数是1次，“b”出现的次数是2次，“c”出现的次数是3次，“d”出现的次数是4次，现在以各字符出现的次数构建一个Huffman树，并为各字符编码，则每个字符所对应的Huffman编码：

‘a’：100 ‘b’：101‘c’：11‘d’：0

根据每个字符的huffman编码对照原文件字符出现的顺序写出Huffman code：

10010110 11111110 000

Huffman code 一共19个比特位，每8位为一个字节写入压缩文件，当余下的不够一个字节时，在后面补0够一个字节再写入，这里需要补上5个0总共3个字节，而原文件是10个字节，少了7个字节，实现了压缩。

4.1 重要程序段1

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

using namespace std;

struct head

{

int b,c; //字符,数

long count; //文件中该字符出现的次数

long parent, lch, rch; //make a tree

char bits[256]; //哈夫曼树编码

};

struct head header[512], tmp; //节点树

//函数：compress()

//作用：读取文件内容并加以压缩

//将压缩内容写入另一个文档

int compress(const char\* filename, const char\* outputfile)

{

char buf[512];

unsigned char c;

long i, j, m, n, f;

long min1, pt1, flength;

FILE\* ifp, \* ofp;

int per = 10;

ifp = fopen(filename, "rb"); //打开原始文件

if (ifp == NULL)

{

printf("打开文件失败:%s\n", filename);

return 0; //如果打开失败，则输出错误信息

}

ofp = fopen(outputfile, "wb"); //打开压缩后存储信息的文件

if (ofp == NULL)

{

printf("打开文件失败:%s\n", outputfile);

return 0;

}

flength = 0;

while (!feof(ifp))

{

fread(&c, 1, 1, ifp);

header[c].count++; //读文件，统计字符出现次数

cout << c;

flength++; //记录文件的字符总数

}

cout << endl;

flength--;

header[c].count--;

for (i = 0; i < 512; i++) //HUFFMAN算法中初始节点的设置

{

if (header[i].count != 0)

header[i].b = (unsigned char)i;

else

header[i].b = -1;

header[i].parent = -1;

header[i].lch = header[i].rch = -1;

}

for (i = 0; i < 256; i++) //将节点按出现次数排序

{

for (j = i + 1; j < 256; j++)

{

if (header[i].count < header[j].count)

{

tmp = header[i];

header[i] = header[j];

header[j] = tmp;

}

}

}

for (i = 0; i < 256; i++) //统计不同字符的数量

{

if (header[i].count == 0)

break;

}

n = i;

m = 2 \* n - 1;

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

{

min1 = 999999999;

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

{

if (header[j].parent != -1) continue;

if (min1 > header[j].count)

{

pt1 = j;

min1 = header[j].count;

continue;

}

}

header[i].count = header[pt1].count;

header[pt1].parent = i;

header[i].lch = pt1;

min1 = 999999999;

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

{

if (header[j].parent != -1) continue;

if (min1 > header[j].count)

{

pt1 = j;

min1 = header[j].count;

continue;

}

}

header[i].count += header[pt1].count;

header[i].rch = pt1;

header[pt1].parent = i;

}

for (i = 0; i < n; i++) //构造HUFFMAN树，设置字符的编码

{

f = i;

header[i].bits[0] = 0;

while (header[f].parent != -1)

{

j = f;

f = header[f].parent;

if (header[f].lch == j)

{

j = strlen(header[i].bits);

memmove(header[i].bits + 1, header[i].bits, j + 1);

header[i].bits[0] = '0';

}

else

{

j = strlen(header[i].bits);

memmove(header[i].bits + 1, header[i].bits, j + 1);

header[i].bits[0] = '1';

}

}

cout << header[i].b << " : " << header[i].bits << endl;

}

//下面的就是读原文件的每一个字符，按照设置好的编码替换文件中的字符

fseek(ifp, 0, SEEK\_SET); //将指针定在文件起始位置

fseek(ofp, 8, SEEK\_SET); //以8位二进制数为单位进行读取

buf[0] = 0;

f = 0;

pt1 = 8;

printf("当前文件有:%d字符\n", flength);

printf("正在压缩\n");

while (!feof(ifp))

{

c = fgetc(ifp);

f++;

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

{

if (c == header[i].b) break;

}

strcat(buf, header[i].bits);

j = strlen(buf);

c = 0;

while (j >= 8) //当剩余字符数量不小于8个时

{

for (i = 0; i < 8; i++) //按照八位二进制数转化成十进制ASCII码写入文件一次进行压缩

{

if (buf[i] == '1') c = (c << 1) | 1;

else c = c << 1;

}

fwrite(&c, 1, 1, ofp);

pt1++;

strcpy(buf, buf + 8);

j = strlen(buf);

}

if (f == flength)

break;

}

if (j > 0) //当剩余字符数量少于8个时

{

strcat(buf, "00000000");

for (i = 0; i < 8; i++)

{

if (buf[i] == '1') c = (c << 1) | 1;

else c = c << 1; //对不足的位数进行补零

}

fwrite(&c, 1, 1, ofp);

pt1++;

}

fseek(ofp, 0, SEEK\_SET); //将编码信息写入存储文件

fwrite(&flength, 1, sizeof(flength), ofp);

fwrite(&pt1, sizeof(long), 1, ofp);

fseek(ofp, pt1, SEEK\_SET);

fwrite(&n, sizeof(long), 1, ofp);

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

{

tmp = header[i];

fwrite(&(header[i].b), 1, 1, ofp);

pt1++;

c = strlen(header[i].bits);

fwrite(&c, 1, 1, ofp);

pt1++;

j = strlen(header[i].bits);

if (j % 8 != 0) //当位数不满8时，对该数进行补零操作

{

for (f = j % 8; f < 8; f++)

strcat(header[i].bits, "0");

}

while (header[i].bits[0] != 0)

{

c = 0;

for (j = 0; j < 8; j++)

{

if (header[i].bits[j] == '1') c = (c << 1) | 1;

else c = c << 1;

}

strcpy(header[i].bits, header[i].bits + 8);

fwrite(&c, 1, 1, ofp); //将所得的编码信息写入文件

pt1++;

}

header[i] = tmp;

}

fclose(ifp);

fclose(ofp); //关闭文件

cout << "压缩后文件有:" << pt1<<"位二进制字节\n";

cout <<"压缩后文件有"<<pt1/8+1<<"位字符\n" ;

float persent = (pt1 / 8 + 1) / 1.0\*flength;

cout << "压缩率是" << persent<<"%"<<endl;

return 1; //返回压缩成功信息

}

4.2 重要程序段2

//函数：uncompress()

//作用：解压缩文件，并将解压后的内容写入新文件

int uncompress(const char\* filename, const char\* outputfile)

{

char buf[255], bx[255];

unsigned char c;

char out\_filename[512];

long i, j, m, n, f, p, l;

long flength;

int per = 10;

int len = 0;

FILE\* ifp, \* ofp;

char c\_name[512] = { 0 };

ifp = fopen(filename, "rb"); //打开文件

if (ifp == NULL)

{

return 0; //若打开失败，则输出错误信息

}

//读取原文件长

if (outputfile)

strcpy(out\_filename, outputfile);

else

strcpy(out\_filename, c\_name);

ofp = fopen(out\_filename, "wb"); //打开文件

if (ofp == NULL)

{

return 0;

}

fseek(ifp, 0, SEEK\_END);

len = ftell(ifp);

fseek(ifp, 0, SEEK\_SET);

printf("当前文件有:%d字符\n", len);

printf("正在解压\n");

fread(&flength, sizeof(long), 1, ifp); //读取原文件长

fread(&f, sizeof(long), 1, ifp);

fseek(ifp, f, SEEK\_SET);

fread(&n, sizeof(long), 1, ifp); //读取原文件各参数

for (i = 0; i < n; i++) //读取压缩文件内容并转换成二进制码

{

fread(&header[i].b, 1, 1, ifp);

fread(&c, 1, 1, ifp);

p = (long)c;

header[i].count = p;

header[i].bits[0] = 0;

if (p % 8 > 0) m = p / 8 + 1;

else m = p / 8;

for (j = 0; j < m; j++)

{

fread(&c, 1, 1, ifp);

f = c;

\_itoa(f, buf, 2);

f = strlen(buf);

for (l = 8; l > f; l--)

{

strcat(header[i].bits, "0"); //位数不足，执行补零操作

}

strcat(header[i].bits, buf);

}

header[i].bits[p] = 0;

}

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

{

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

{

if (strlen(header[i].bits) > strlen(header[j].bits))

{

tmp = header[i];

header[i] = header[j];

header[j] = tmp;

}

}

}

p = strlen(header[n - 1].bits);

fseek(ifp, 8, SEEK\_SET);

m = 0;

bx[0] = 0;

while (1)

{

while (strlen(bx) < (unsigned int)p)

{

fread(&c, 1, 1, ifp);

f = c;

\_itoa(f, buf, 2);

f = strlen(buf);

for (l = 8; l > f; l--)

{

strcat(bx, "0");

}

strcat(bx, buf);

}

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

{

if (memcmp(header[i].bits, bx, header[i].count) == 0) break;

}

strcpy(bx, bx + header[i].count);

c = header[i].b;

fwrite(&c, 1, 1, ofp);

m++;

cout << header[i].b << " : " << header[i].bits << endl;

if (m == flength) break;

}

fclose(ifp);

fclose(ofp);

cout << "解压后文件为" << out\_filename<<endl;

cout << "解压后文件有:" << flength << "位字符\n";

return 1; //输出成功信息

}

4.3 重要程序段3

void text2bin(const char\* filename, const char\* extractfilename2) {

int count = 0;

int ch, a;

char temp;

FILE\* fin = fopen(filename, "r");

FILE\* fout = fopen(extractfilename2, "w");

while (fscanf(fin, "%c", &temp) != EOF) {

ch = temp;

for (a = 7; a >= 0; a--) fprintf(fout, "%d", ch >> a & 1);

}

fclose(fin);

fclose(fout);

}

4.4 重要程序段4

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

{

int flag;

char c;

char filename[100], extractfilename[100], extractfilename2[100];

while (1)

{

printf("\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\t| |\n");

printf("\t| C-压缩文件 |\n");

printf("\t| E-解压缩 |\n");

printf("\t| Q-退出 |\n");

printf("\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\n");

do

{

printf("\n\t请选择功能:");

scanf(" %c", &c);

c = toupper(c); //小写变大写

putchar('\n');

if ('C' != c && 'E' != c && 'Q' != c)

{

printf("\t选项错误,请重新输入!\n");

}

} while ('C' != c && 'E' != c && 'Q' != c);

if ('C' == c)

{

printf("\t请您输入需要压缩的文件:");

fflush(stdin); //fflush(stdin)刷新标准输入缓冲区，把输入缓冲区里的东西丢弃

cin>>filename;

putchar('\n');

printf("\t请您输入压缩后的文件:");

fflush(stdin);

cin >> extractfilename;

putchar('\n');

printf("\t请您输入要保存的压缩后的二进制文件名:");

fflush(stdin);

cin >> extractfilename2;

putchar('\n');

flag = compress(filename, extractfilename);

putchar('\n');

if (-1 == flag)

{

printf("\t文件打开失败!\n");

exit(1);

}

else

{

printf("\n\t压缩操作完成!\n\n");

text2bin(filename, extractfilename2);

}

}

else if ('E' == c)

{

printf("\t请您输入需要解压的文件:");

fflush(stdin);

cin >> extractfilename;

putchar('\n');

printf("\t请您输入解压缩后的文件:");

fflush(stdin);

cin >> filename;

putchar('\n');

flag = uncompress(extractfilename, filename);

putchar('\n');

if (-1 == flag)

{

printf("\t文件打开失败!\n");

exit(1);

}

else

{

printf("\n\t解压缩操作完成!\n\n");

}

}

if ('Q' == c) {

printf("\t感谢使用!\n");

exit(0);

}

}

return 0;

}

## 第五章 调试分析

调试程序分析是课程设计必不可少的环节，调试分析可以在程序运行之后发现原来没有出现的问题，以及做进一步的更正，是程序达到预期的目的。

问题一：如何验证音频和图片压缩是否正确？

beyond compare 对于图片文件和音频文件不识别，这里只能通过人工的方法进行验证，因为在进行解压缩的时候加了后缀信息，所以在进行验证的时候需要将后缀修改为和原文件类型相同的后缀再去打开，通过验证得出结论：图片文件和音频文件也能够正常压缩和解压缩。

## 第六章 运行结果

1、首先进入初始界面，如图1所示。

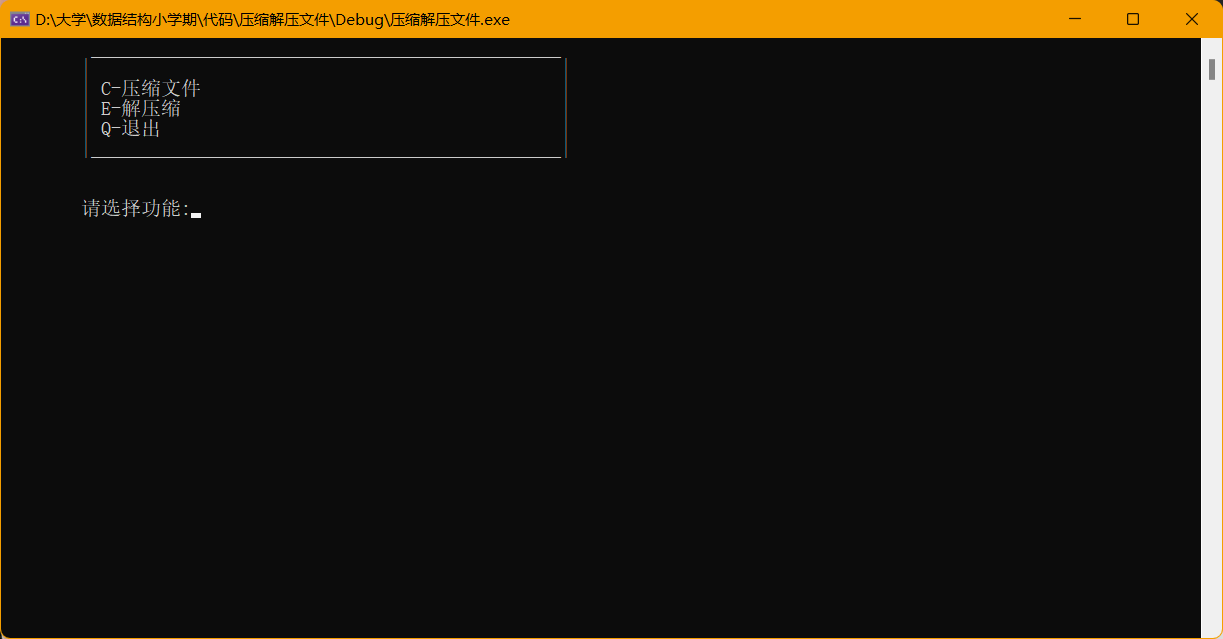


图1 系统初始界面

2、进入压缩功能，如图2所示

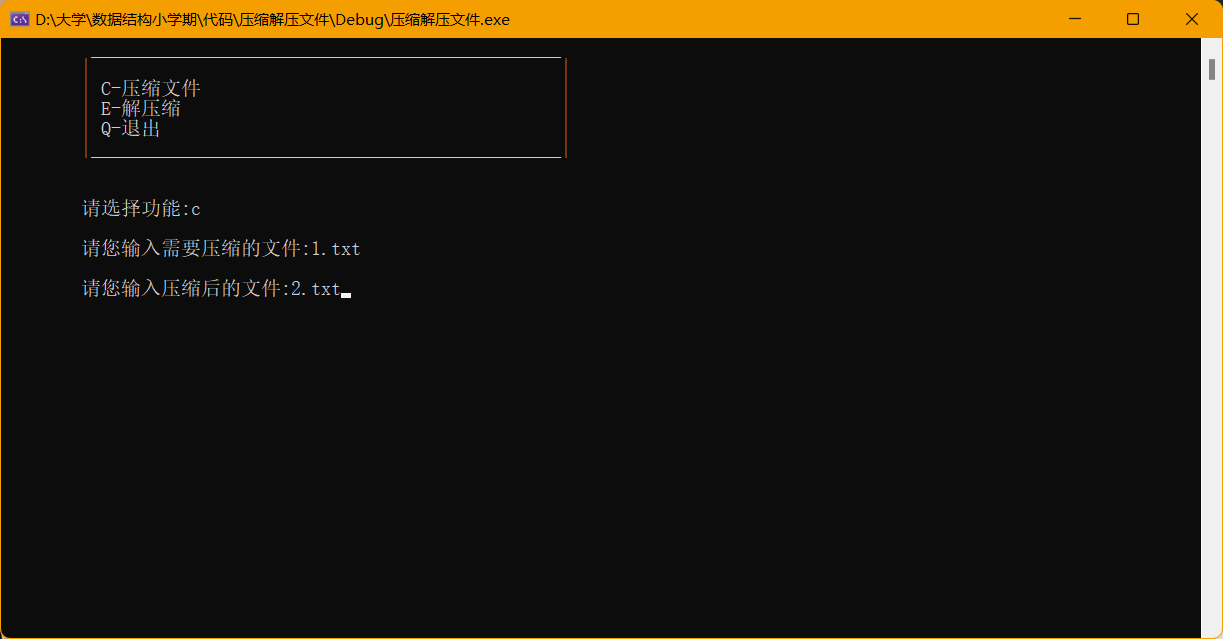


图2 压缩功能

3、压缩成功界面如图3所示。

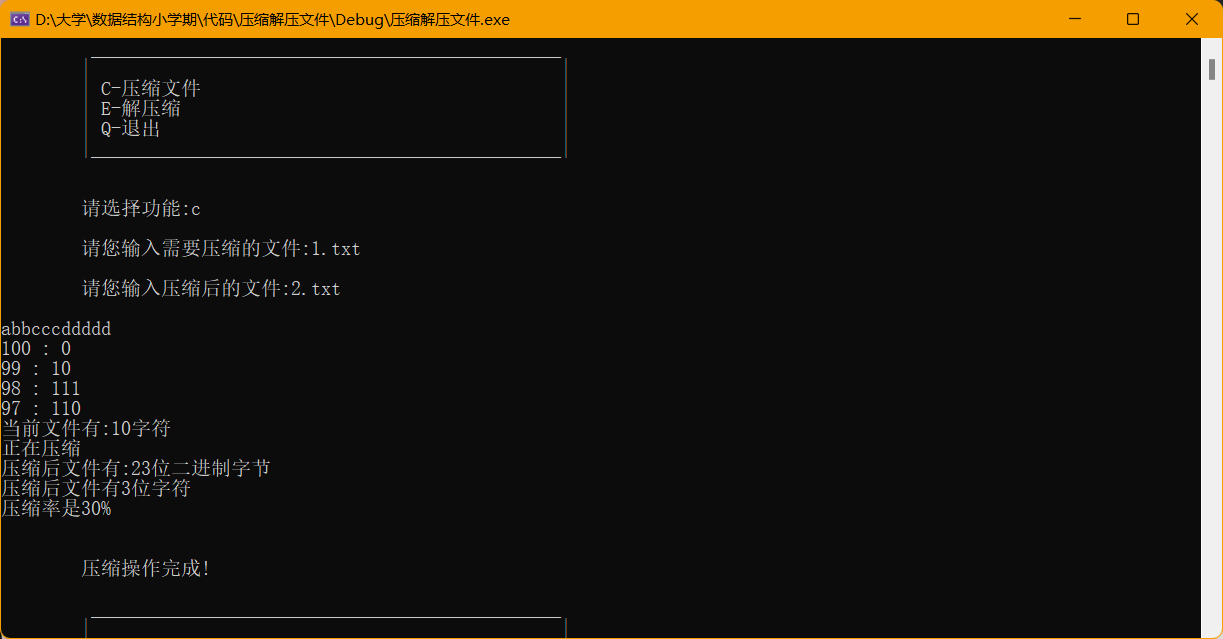


图3 压缩成功界面

4、选择解压缩功能。

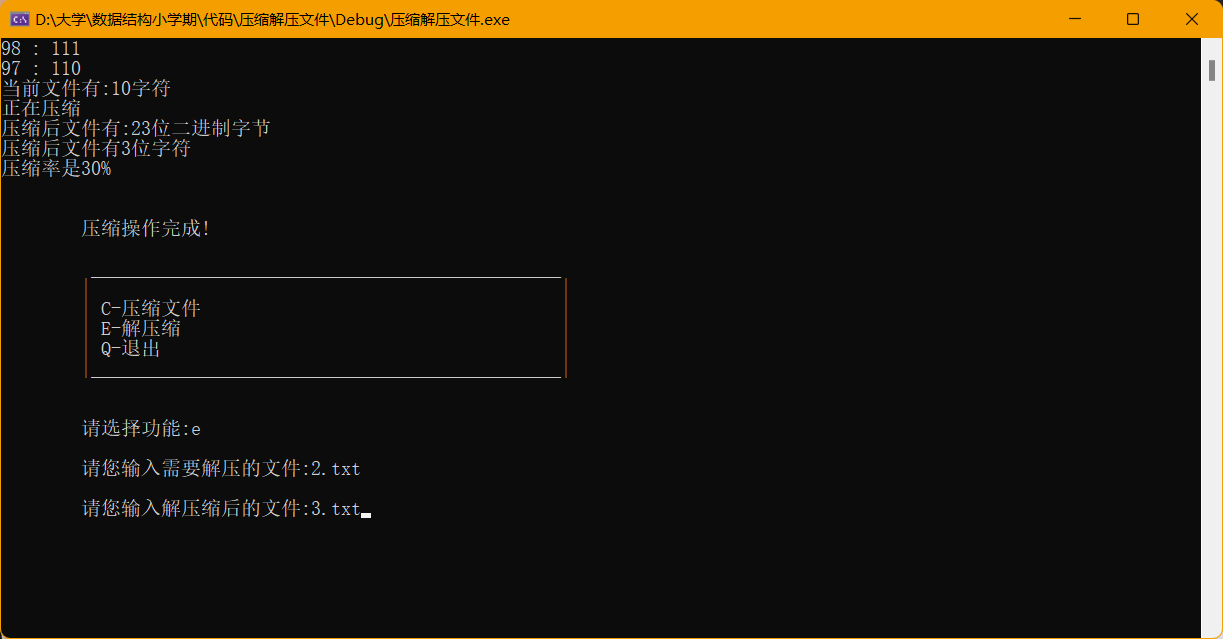


图4 解压缩功能

5、解压缩成功，如图5所示



图5查看记录

6、退出，如图6所示

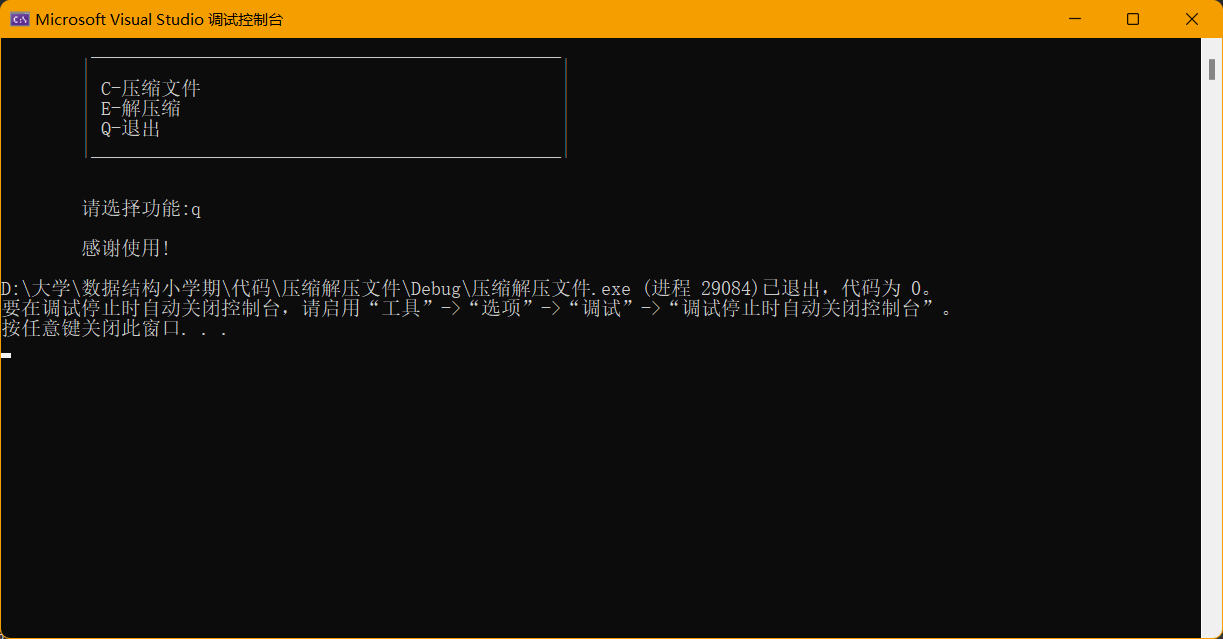
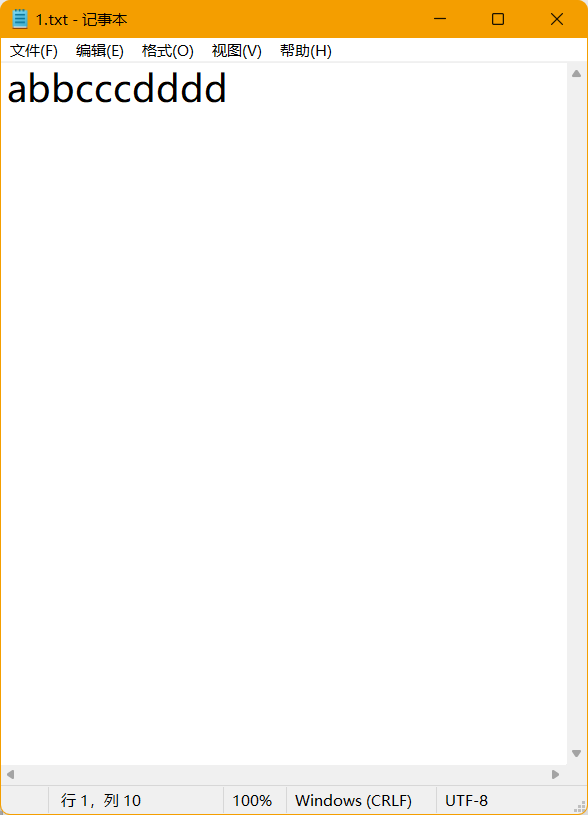
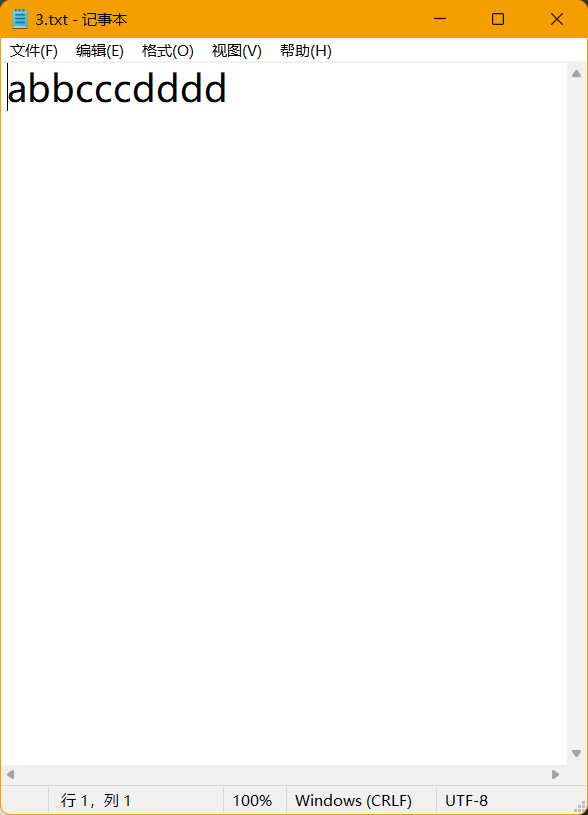
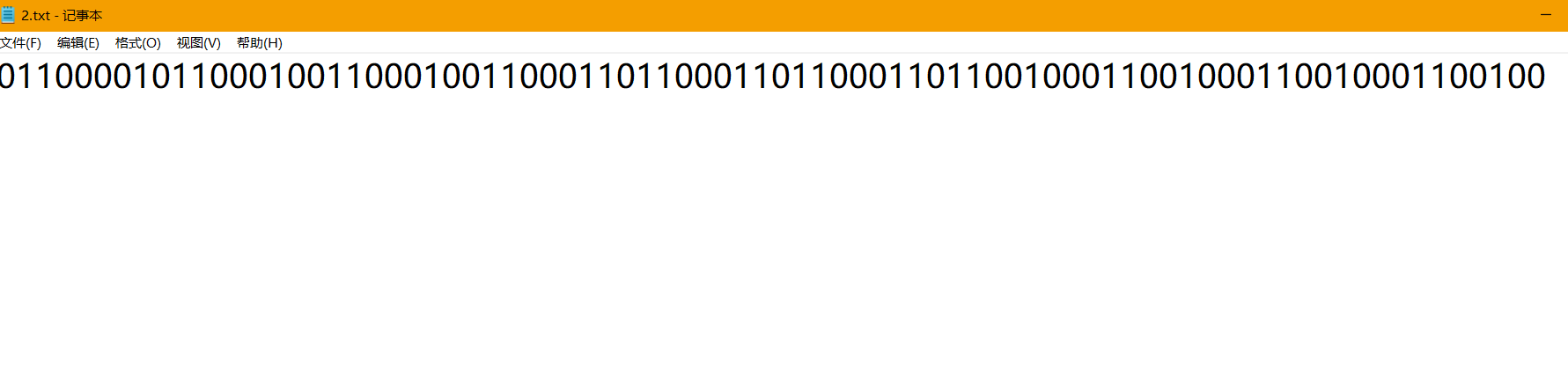


图6 退出

7.打开文件，验证是否正确





## 附录：完整代码

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdlib.h>

#include <stdio.h>

#include <string.h>

#include <ctype.h>

using namespace std;

struct head

{

int b,c; //字符,数

long count; //文件中该字符出现的次数

long parent, lch, rch; //生成树

char bits[256]; //哈夫曼树编码

};

struct head header[512], tmp; //节点树

//函数：compress()

//作用：读取文件内容并加以压缩

//将压缩内容写入另一个文档

int compress(const char\* filename, const char\* outputfile)

{

char buf[512];

unsigned char c;

long i, j, m, n, f;

long min1, pt1, flength;

FILE\* ifp, \* ofp;

int per = 10;

ifp = fopen(filename, "rb"); //打开原始文件,rb 以读/写方式打开一个二进制文件，只允许读/写数据。

if (ifp == NULL)

{

printf("打开文件失败:%s\n", filename);

return 0; //如果打开失败，则输出错误信息

}

ofp = fopen(outputfile, "wb"); //打开压缩后存储信息的文件 wb:只写方式打开或新建一个二进制文件，只允许写数据。

if (ofp == NULL)

{

printf("打开文件失败:%s\n", outputfile);

return 0;

}

flength = 0;

while (!feof(ifp)) //功能是检测流上的文件结束符

{

fread(&c, 1, 1, ifp);

//fread( void \*buffer, size, count, FILE \*stream )

/\*buffer 指向要读取的数组中首个对象的指针 size 每个对象的大小（单位是字节）

count 要读取的对象个数 stream 输入流\*/

header[c].count++; //读文件，统计字符出现次数

cout << c;

flength++; //记录文件的字符总数

}

cout << endl;

flength--;

header[c].count--;

for (i = 0; i < 512; i++) //HUFFMAN算法中初始节点的设置

{

if (header[i].count != 0)

header[i].b = (unsigned char)i; //表示byte时，都用unsigned char

/\*将每个哈夫曼码值及其对应的ASCII码存放在一维数组header[i]中，

且编码表中的下标和ASCII码满足顺序存放关系\*/

else

header[i].b = -1; //哈夫曼初始化节点都是-1

header[i].parent = -1;

header[i].lch = header[i].rch = -1;

}

for (i = 0; i < 256; i++) //将节点按出现次数排序

{

for (j = i + 1; j < 256; j++)

{

if (header[i].count < header[j].count)

{

tmp = header[i];

header[i] = header[j];

header[j] = tmp;

}

}

}

for (i = 0; i < 256; i++) //统计不同字符的数量

{

if (header[i].count == 0)

break;

}

n = i; //外部叶子结点数为n个时，内部结点数为n-1，整个哈夫曼树的需要的结点数为2\*n - 1.

m = 2 \* n - 1;

for (i = n; i < m; i++)//构建哈夫曼树

{

min1 = 999999999;//结点出现的最大次数

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

{

if (header[j].parent != -1) continue;

// parent != -1说明该结点已存在哈夫曼树中，跳出循环重新选择新结点

if (min1 > header[j].count)

{

pt1 = j;

min1 = header[j].count;

continue;

}

}

header[i].count = header[pt1].count;

header[pt1].parent = i;

// 依据parent域值（结点层数）确定树中结点之间的关系

header[i].lch = pt1;//计算左分支权值大小

min1 = 999999999;

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

{

if (header[j].parent != -1) continue;

if (min1 > header[j].count)

{

pt1 = j;

min1 = header[j].count;

continue;

}

}

header[i].count += header[pt1].count;

header[i].rch = pt1;//计算右分支权值大小

header[pt1].parent = i;

}

for (i = 0; i < n; i++) //构造HUFFMAN树，设置字符的无重复前缀编码

{

f = i; //i=1，f=1

header[i].bits[0] = 0; //根节点编码为0

while (header[f].parent != -1)//1的母亲非空

{

j = f;//j=1

f = header[f].parent;//1=1的母亲结点0

if (header[f].lch == j)//如果左孩子是1，置左分支编码0

{

j = strlen(header[i].bits);//j=1

memmove(header[i].bits + 1, header[i].bits, j + 1);//memmove用于拷贝字节

header[i].bits[0] = '0';//左0右1，依次存储连接“0” “1”编码

}

else//置右分支编码1

{

j = strlen(header[i].bits);

memmove(header[i].bits + 1, header[i].bits, j + 1);

header[i].bits[0] = '1';

}

}

cout << header[i].b << " : " << header[i].bits << endl;

}

//读原文件的每一个字符，按照设置好的编码替换文件中的字符

fseek(ifp, 0, SEEK\_SET); //将指针定在文件起始位置

fseek(ofp, 8, SEEK\_SET); //以8位二进制数为单位进行读取

buf[0] = 0;//定义缓冲区

f = 0;

pt1 = 8;

/\*假设原文件第一个字符是"A"，8位2进制为01000001（65），编码后为0110识别编码第一个’0’，

将其左移一位，。下一个是'1'，应该|（或） 1，结果00000001

同理4位都做完，应该是00000110，由于字节中的8位并没有全部用完，们应该继续读下一个字符，

根据编码表继续拼完剩下的4位，如果字符的编码不足4位，还要继续读个字符，

如果字符编码超过4位，那么我们将把剩下的位信息拼接到一个新的字节里\*/

printf("当前文件有:%d字符\n", flength);

printf("正在压缩\n");

while (!feof(ifp))

{

c = fgetc(ifp);

f++;

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

{

if (c == header[i].b) break;

}

strcat(buf, header[i].bits);

j = strlen(buf);

c = 0;

while (j >= 8) //当剩余字符数量不小于8个时，压缩存储

{

for (i = 0; i < 8; i++) //按照八位二进制数转化成十进制ASCII码写入文件一次进行压缩

{

if (buf[i] == '1') c = (c << 1) | 1;

else c = c << 1;

}

fwrite(&c, 1, 1, ofp);

pt1++;//统计压缩后文件的长度

strcpy(buf, buf + 8);//一个字节一个字节的拼接

j = strlen(buf);

}

if (f == flength)

break;

}

if (j > 0) //当剩余字符数量少于8个时

{

strcat(buf, "00000000");

for (i = 0; i < 8; i++)

{

if (buf[i] == '1') c = (c << 1) | 1;

else c = c << 1; //对不足的位数进行补零

}

fwrite(&c, 1, 1, ofp);

pt1++;

}

fseek(ofp, 0, SEEK\_SET); //将编码信息写入存储文件

fwrite(&flength, 1, sizeof(flength), ofp);

fwrite(&pt1, sizeof(long), 1, ofp);

fseek(ofp, pt1, SEEK\_SET);

fwrite(&n, sizeof(long), 1, ofp);

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

{

tmp = header[i];

fwrite(&(header[i].b), 1, 1, ofp);

pt1++;

c = strlen(header[i].bits);

fwrite(&c, 1, 1, ofp);

pt1++;

j = strlen(header[i].bits);

if (j % 8 != 0) //当位数不满8时，对该数进行补零操作

{

for (f = j % 8; f < 8; f++)

strcat(header[i].bits, "0");

}

while (header[i].bits[0] != 0)

{

c = 0;

for (j = 0; j < 8; j++)//字符的有效存储不超过8位，则对有效位数位移实现两字符编码的连接

{

if (header[i].bits[j] == '1') c = (c << 1) | 1;//| 1不改变原位置的0,1值

else c = c << 1;

}

strcpy(header[i].bits, header[i].bits + 8);//把字符的编码按照原先的存储顺序进行连接

fwrite(&c, 1, 1, ofp); //将所得的编码信息写入文件

pt1++;

}

header[i] = tmp;

}

fclose(ifp);

fclose(ofp); //关闭文件

cout << "压缩后文件有:" << pt1<<"位二进制字节\n";

cout <<"压缩后文件有"<<pt1/8+1<<"位字符\n" ;

float persent = (pt1 / 8 + 1) / 1.0\*flength;

cout << "压缩率是" << persent<<"%"<<endl;

return 1; //返回压缩成功信息

}

//函数：uncompress()

//作用：解压缩文件，并将解压后的内容写入新文件

int uncompress(const char\* filename, const char\* outputfile)

{

char buf[255], bx[255];

unsigned char c;

char out\_filename[512];

long i, j, m, n, f, p, l;

long flength;

int per = 10;

int len = 0;

FILE\* ifp, \* ofp;

char c\_name[512] = { 0 };

ifp = fopen(filename, "rb"); //打开文件

if (ifp == NULL)

{

return 0; //若打开失败，则输出错误信息

}

//读取原文件长

if (outputfile)

strcpy(out\_filename, outputfile);

else

strcpy(out\_filename, c\_name);

ofp = fopen(out\_filename, "wb"); //打开文件

if (ofp == NULL)

{

return 0;

}

fseek(ifp, 0, SEEK\_END);

len = ftell(ifp);

fseek(ifp, 0, SEEK\_SET);

printf("当前文件有:%d字符\n", len);

printf("正在解压\n");

fread(&flength, sizeof(long), 1, ifp); //读取源文件长度，对文件进行定位 //读取原文件长

fread(&f, sizeof(long), 1, ifp);

fseek(ifp, f, SEEK\_SET);

fread(&n, sizeof(long), 1, ifp); //读取原文件各参数

for (i = 0; i < n; i++) //读取压缩文件内容并转换成二进制码

{

fread(&header[i].b, 1, 1, ifp);

fread(&c, 1, 1, ifp);

p = (long)c;//读取源文件字符的权值

header[i].count = p;

header[i].bits[0] = 0;

if (p % 8 > 0) m = p / 8 + 1;

else m = p / 8;

for (j = 0; j < m; j++)

{

fread(&c, 1, 1, ifp);

f = c;

\_itoa(f, buf, 2); //将f转换为二进制表示的字符串

f = strlen(buf);

for (l = 8; l > f; l--)

{

strcat(header[i].bits, "0"); //位数不足，执行补零操作

}

strcat(header[i].bits, buf);

}

header[i].bits[p] = 0;

}

for (i = 0; i < n; i++) //根据哈夫曼编码的长短，对结点进行排序

{

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

{

if (strlen(header[i].bits) > strlen(header[j].bits))

{

tmp = header[i];

header[i] = header[j];

header[j] = tmp;

}

}

}

p = strlen(header[n - 1].bits);

fseek(ifp, 8, SEEK\_SET);

m = 0;

bx[0] = 0;

while (1) // 通过哈夫曼编码的长短，依次解码，从原来的位存储还到字节存储

{

while (strlen(bx) < (unsigned int)p)

{

fread(&c, 1, 1, ifp);

f = c;

\_itoa(f, buf, 2);

f = strlen(buf);

for (l = 8; l > f; l--)//在单字节内对相应位置补O

{

strcat(bx, "0");

}

strcat(bx, buf);

}

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

{

if (memcmp(header[i].bits, bx, header[i].count) == 0) break;

}

strcpy(bx, bx + header[i].count);/\*从压缩文件中的按位存储还到按字节存储字符，

字符位置不改变\*/

c = header[i].b;

fwrite(&c, 1, 1, ofp);

m++;

cout << header[i].b << " : " << header[i].bits << endl;

if (m == flength) break;

}

fclose(ifp);

fclose(ofp);

cout << "解压后文件为" << out\_filename<<endl;

cout << "解压后文件有:" << flength << "位字符\n";

return 1; //输出成功信息

}

void text2bin(const char\* filename, const char\* extractfilename2) {

int count = 0;

int ch, a;

char temp;

FILE\* fin = fopen(filename, "r");

FILE\* fout = fopen(extractfilename2, "w");

while (fscanf(fin, "%c", &temp) != EOF) {

ch = temp;

for (a = 7; a >= 0; a--) fprintf(fout, "%d", ch >> a & 1);

}

fclose(fin);

fclose(fout);

}

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

{

int flag;

char c;

char filename[100], extractfilename[100], extractfilename2[100];

while (1)

{

printf("\t \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\t| |\n");

printf("\t| C-压缩文件 |\n");

printf("\t| E-解压缩 |\n");

printf("\t| Q-退出 |\n");

printf("\t|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

printf("\n");

do

{

printf("\n\t请选择功能:");

scanf(" %c", &c);

c = toupper(c); //小写变大写

putchar('\n');

if ('C' != c && 'E' != c && 'Q' != c)

{

printf("\t选项错误,请重新输入!\n");

}

} while ('C' != c && 'E' != c && 'Q' != c);

if ('C' == c)

{

printf("\t请您输入需要压缩的文件:");

fflush(stdin); //fflush(stdin)刷新标准输入缓冲区，把输入缓冲区里的东西丢弃

cin>>filename;

putchar('\n');

printf("\t请您输入压缩后的文件:");

fflush(stdin);

cin >> extractfilename;

putchar('\n');

printf("\t请您输入要保存的压缩后的二进制文件名:");

fflush(stdin);

cin >> extractfilename2;

putchar('\n');

flag = compress(filename, extractfilename);

putchar('\n');

if (-1 == flag)

{

printf("\t文件打开失败!\n");

exit(1);

}

else

{

printf("\n\t压缩操作完成!\n\n");

text2bin(filename, extractfilename2);

}

}

else if ('E' == c)

{

printf("\t请您输入需要解压的文件:");

fflush(stdin);

cin >> extractfilename;

putchar('\n');

printf("\t请您输入解压缩后的文件:");

fflush(stdin);

cin >> filename;

putchar('\n');

flag = uncompress(extractfilename, filename);

putchar('\n');

if (-1 == flag)

{

printf("\t文件打开失败!\n");

exit(1);

}

else

{

printf("\n\t解压缩操作完成!\n\n");

}

}

if ('Q' == c) {

printf("\t感谢使用!\n");

exit(0);

}

}

return 0;

}