|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 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：2021.07.08**

目 录·

[Prim算法 3](#_Toc76456258)

[1.任务说明 3](#_Toc76456259)

[2.概要设计 3](#_Toc76456260)

[3.详细设计 列出主程序清单。（子程序略） 4](#_Toc76456261)

[} 6](#_Toc76456262)

[4.系统测试 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） 8](#_Toc76456267)

[4.测试 14](#_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 15](#_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 23](#_Toc76456278)

[心得体会 24](#_Toc76456279)

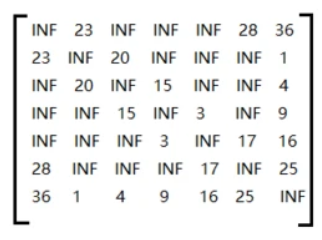
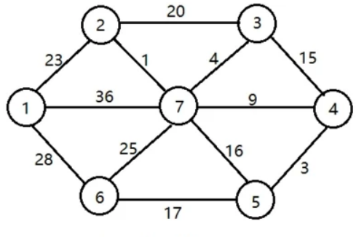
### Prim算法

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;

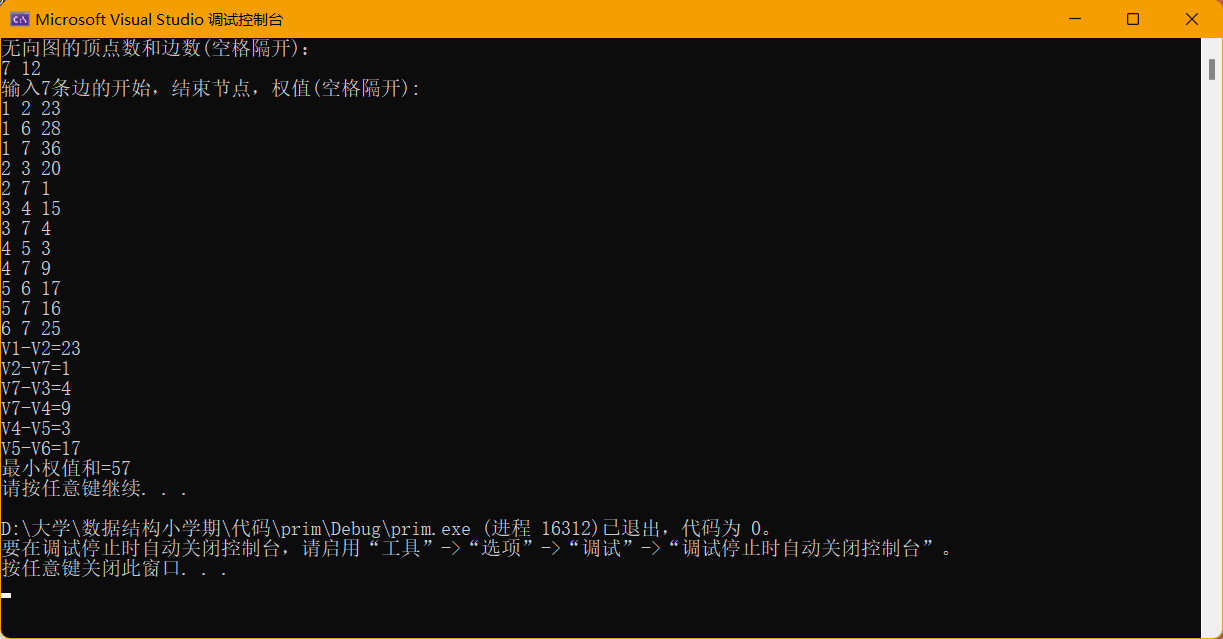
system("pause");

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()

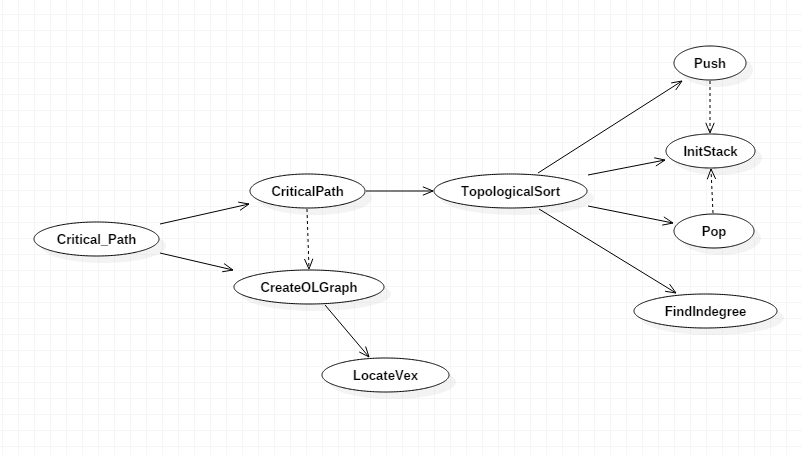
返回图中顶点位置 LocateVex()

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

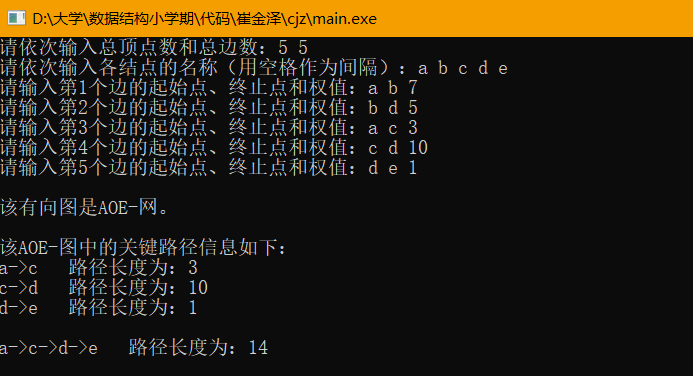
1）关键路径算法

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()

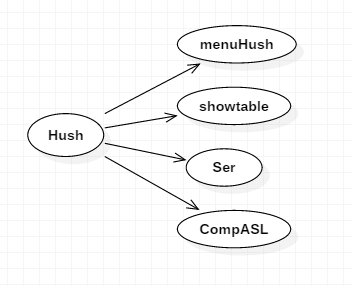
菜单函数menuHush()

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。

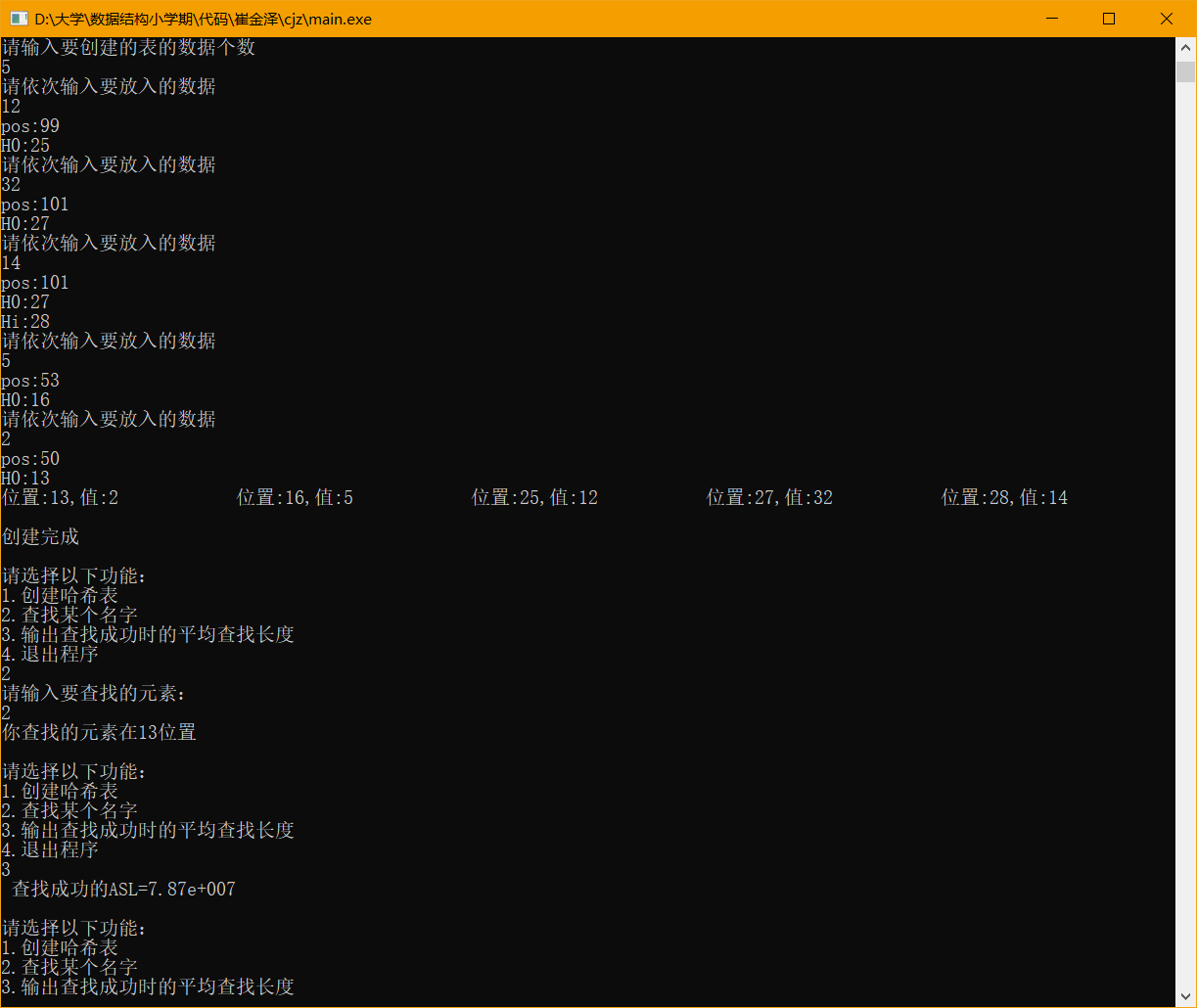
② 输出的形式：

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);

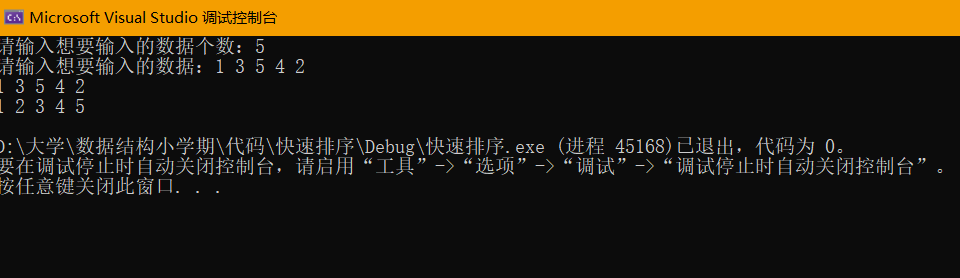
quickSort(a, 0, m-1);//最后一个参数是n-1

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。