实验七 图实验二

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 姓名 | 张渊 | 学号 | 2022210000 | 专业班级 | 计科22-0 |
| 指导教师 | 张先宜 | 实验时间 | 2023.6.11 | 实验地点 |  |

7.1 **实验目的**

(1) 掌握图的基本概念。

(2) 掌握图的存储结构的设计与实现，基本运算的实现。

(3) 掌握图的Dijkstra和Floyd两种最短路径算法。

(4) 掌握有向无环图的的拓扑排序和关键路径求解算法。

7.2 **实验要求**

1. 结构定义和算法实现放入库文件，如“graph.h”中；
2. 图的测试数据用文本文件方式给出，例如测试数据名为dn.grp的有向网，可参考发来的图形状和参考存储文件；
3. 图创建方法可自行选择；
4. 可多次连续测试。

7.3 **实验任务**

分别设计图（网）的邻接矩阵、邻接表存储结构，编写算法实现下列问题的求解。

1．实现Dijkstra算法，求解下列给定图G指定顶点到其余顶点之间的最短路径。 实验测试数据基本要求：第一组数据： udn6.grp

第二组数据： un8.grp

第三组数据： dn8.grp

第四组数据： dn10.grp

2．实现Floyd算法，求解下列给定图G各顶点之间的最短路径。 实验测试数据基本要求：第一组数据： udn6.grp

第二组数据： un8.grp

第三组数据： dn8.grp

第四组数据： dn10.grp

3．设计算法求解下列给定图G的拓扑序列。 实验测试数据基本要求：

第一组数据： top6dg1.grp

第二组数据： top7dg1.grp

第三组数据： dn8.grp

第四组数据： dn10.grp

4．设计算法求解下列给定AOE网的关键路径。 实验测试数据基本要求：

第一组数据： dag11.grp

第二组数据： dag12.grp

**7.4数据结构设计**

邻接矩阵

typedef struct GraphAdjMatrix //邻接矩阵

{

elementType VerList[MaxVerNum + 1]; //顶点数组，存放顶点元素的值，Data[0]单元不用

cellType AdjMatrix[MaxVerNum + 1][MaxVerNum + 1]; //邻接矩阵，数组下标为0单元不用，从AdjMatrix[1][1]单元开始

int VerNum; //顶点数

int ArcNum; //弧（边）数

GraphKind gKind; //图的类型:0-无向图；1-无向网；2-有向图；3-有向网

} Graph; //图的类型名

邻接表

typedef struct eNode //边链表结点结构

{

int adjVer; //邻接顶点地址，此处为顶点在顶点表中序号，从1开始

eInfoType eInfo; //边链表中表示边的相关信息，比如表的权值

struct eNode\* next; //指向边链表中的下条边

}EdgeNode; //边

typedef struct vNode //顶点表中元素结构

{

elementType data; //存放图中顶点的数据

EdgeNode\* firstEdge; //指向此顶点关联的第一条边的指针，即边链表的头指针

}VerNode;

typedef struct GraphAdjLinkList //邻接表

{

VerNode VerList[MaxVerNum + 1];//存放顶点的顺序表，数组0单元不用

int VerNum; //顶点数

int ArcNum; //弧（边）数

GraphKind gKind; //图的类型:0-无向图；1-无向网；2-有向图；3-有向网

}Graph; //图的类型名

**7.5算法设计（以邻接矩阵为例）**

1 【算法思想】 实现Dijkstra算法，求解下列给定图G指定顶点到其余顶点之间的最短路径。按照最短路径长度不减的次序求解各顶点的解，即按由远及近的次序递推求解各个顶点的解。vID为选定的起始顶点。Path[]数组保存全图的最短路径信息，下标对应顶点的编号，数组的值表示当前顶点的直接前驱的编号。Dist[]数组用来保存全图顶点到指定起点v0的最短距离值。每次选取dist[]最小的顶点，并且更新其邻接顶点到顶点的最短距离，若dist[i]+这两个顶点间的权值小于原本的dist[]的值，则需要更新，反正则不需要。全部处理完之后，倒着迭代输出路径和最小路径的距离即可。

【算法描述】

//1.Dijkstra算法--基于邻接表

void Dijkstra(Graph& G, int path[], int dist[], int vID)

{

int solved[MaxVerNum]; //标记顶点是否已经求出最短路径（已在集合S中）。1-已求出；0-未求出。

int i, j;

int v; //顶点编号

eInfoType minDist; //保存最短距离值

EdgeNode\* p;

//初始化集合S，距离数组dist[]，路径数组path[]

for (i = 1; i <= G.VerNum; i++)

{

solved[i] = 0; //所有顶点均为处理

dist[i] = INF; //所有顶点初始距离置为无穷大（INF）

path[i] = -1; //所有顶点的前驱置为-1，即无前驱

}

//处理顶点vID

solved[vID] = 1; //标记vID已经处理

dist[vID] = 0;

path[vID] = -1;

//从邻接表初始化dist[]和path[]

p = G.VerList[vID].firstEdge; //顶点vID的边链表指针

while (p)

{

v = p->adjVer; //取得vID邻接顶点编号

dist[v] = p->eInfo; //取得vID与v之间边的权值，赋给dist[v]

path[v] = vID; //顶点v的前驱为vID

p = p->next;

}

//依次找出余下n-1个顶点加入集合S中

for (i = 1; i < G.VerNum; i++)

{

minDist = INF;

//寻找集合V-S中距离vID最近的顶点

for (j = 1; j <= G.VerNum; j++)

{

if (solved[j] == 0 && dist[j] < minDist)

{

v = j; //j为V-S中候选的距离vID最近的顶点

minDist = dist[j];

}

}

if (minDist == INF) //S与V-S没有相邻的顶点，算法退出

return;

cout << "起始顶点"<<vID<<"到选择顶点：" << G.VerList[v].data << "的最短距离：" << minDist << endl;

solved[v] = 1; //标记顶点v以找到最短距离，加入集合S中

//对选中的顶点v，更新集合V-S中所有与v邻接的顶点距离vID的距离

p = G.VerList[v].firstEdge; //取得v的边链表指针

while (p)

{

j = p->adjVer; //取得v的邻接顶点编号

if (solved[j] == 0 && minDist + p->eInfo < dist[j])

{

dist[j] = minDist + p->eInfo; //更新顶点j的最小距离

path[j] = v; //j的前驱改为顶点v

}

p = p->next;

}

}

}

2 【算法思想】实现Floyd算法，求解下列给定图G各顶点之间的最短路径。弗洛伊德算法是通过邻接矩阵的递推更新完成的，所以要用到邻接矩阵。Dist数组表示为i，j之间的最短的距离。Path数组保存的是i到j的路径上的j的直接前驱顶点。初始时更具邻接矩阵初始化dist数组。初始化path数组，若i，j之间存在边，那么j的前驱为i，否则设置为-1。不断循环尝试以顶点1，2….n为跳点，若距离有所缩短则更新dist数组，并更新j的直接前驱为m。最终已合适的方法输出path数组和dist数组就可以得到每个顶点间的最短路径。

【算法描述】

//2.基于邻接表的Floyd算法

void Floyd(Graph1& G1, eInfoType dist[MaxVerNum][MaxVerNum], int path[MaxVerNum][MaxVerNum])

{

int i, j, k;

//初始化距离矩阵和路径矩阵

for (i = 1; i <= G1.VerNum; i++)

{

for (j = 1; j <= G1.VerNum; j++)

{

dist[i][j] = G1.AdjMatrix[i][j]; //距离矩阵初始化为邻接矩阵

//初始化路径矩阵，路径矩阵元素path[i][j]中保存编号j顶点的前驱的顶点编号

if (i != j && G1.AdjMatrix[i][j] < INF) //如果i,j之间存在边，则j的前驱为i。否则前驱置为-1

path[i][j] = i;

else

path[i][j] = -1;

}

}

//从k=1开始，迭代到k=G.verNum。依次选择一个顶点k，作为顶点i、j之间的中转顶点，优化顶点i、j之间的距离

//下面是Floyd算法的核心--三重for循环

for (k = 1; k <= G1.VerNum; k++)

{

for (i = 1; i <= G1.VerNum; i++)

{

for (j = 1; j <= G1.VerNum; j++)

{

if (i != j && dist[i][k] + dist[k][j] < dist[i][j]) //k作为中转跳点，i、j之间距离变小，接收k作为中转点，更新i、j之间的距离

{

dist[i][j] = dist[i][k] + dist[k][j]; //更新距离

path[i][j] = path[k][j]; //修改前驱顶点

}

}

}

}

}

3 【算法思想】设计算法求解下列给定图G的拓扑序列。拓扑排序需要先求得各个顶点度入度，所以需要编写算法实现求入度，邻接链表对于出度的求解较为方便，入度的求解较为复杂，我这里利用两重循环来实现，从每个边链表上找到所求顶点的入度。我这里实现的拓扑算法利用的是栈来实现的。先初始化空栈S，在将AOV网中所有入度为0的顶点入栈。若栈不为空出栈，输出V，并且将V的每个后继入度减一，若后继为0，则压入栈中。重复以上操作。最后若拓扑序列中的顶点数目等于所有顶点数目，则表示无环可以进行拓扑排序返回1，否则则表示有环，不可以拓扑排序返回0。

【算法描述】

//3.拓扑排序算法--基于邻接表

int TopologicalSort(Graph& G, int topoList[])

{

int inds[MaxVerNum]; //存放顶点入度

int solvedlen = 0;

int solved[MaxVerNum]; //标记入度为0的顶点是否已经处理。0--未处理；1--已处理。

int i;

int v = -1; //顶点编号

int vCount = 0; //记录入度为0的定点数

EdgeNode\* p; //指向边链表结点的指针

//初始化

for (i = 1; i <= G.VerNum; i++)

{

inds[i] = 0; //所有顶点初始入度置为0

topoList[i] = -1; //拓扑序列初始化为-1

}

//从邻接表获取各顶点初始入度

GetInDegrees(G, inds);

//取得第一个入度为0的顶点（如果存在），保存到v

for (i = 1; i <= G.VerNum; i++)

{

if (inds[i] == 0)

{

solved[solvedlen++] = i;

}

}

while (solvedlen > 0)

{

v = solved[--solvedlen];

topoList[vCount + 1] = v;

vCount++;

//与顶点v相邻的顶点入度减1

p = G.VerList[v].firstEdge;

while (p)

{

v = p->adjVer;

inds[v]--; //邻接点入度减1

if (inds[v] == 0) {

solved[solvedlen++] = v;

}

p = p->next;

}

}

if (vCount == G.VerNum)

return 1; //拓扑排序成功

else

return 0; //存在回路，拓扑排序失败

}

4 【算法思想】设计算法求解下列给定AOE网的关键路径。关键路径上的顶点需要满足最早发生时间等于最晚发生时间。这题用邻接矩阵实现更为简单。求出每个顶点的最早发生时间和最晚发生时间。最早发生时间的求解要用到拓扑排序的序列来实现。最晚发生时间的求解要用到逆拓扑排序序列来实现。因此先求出拓扑排序序列。最早发生时间是前面一个顶点的最早发生时间与前一个顶点和当前顶点的活动的持续时间之和的最大值。最晚发生时间是后一个顶点的最晚发生时间与该顶点和后一个顶点间的活动的持续时间之差的最小值。关键路径上的顶点需要满足最早发生时间等于最晚发生时间。

【算法描述】

//4.求给定AOE网的关键路径(邻接表)

void PrintKeyPath(Graph1& G, int topoList[], int vet[MaxVerNum], int vlt[MaxVerNum]) {

int v, w;

cout << "其中一条关键路径为：\t";

v = topoList[1];

cout << G.VerList[v] << "\t";

while (v != -1) {

w = firstAdj(G, v);

while (w != -1) {

if (vet[w] == vlt[w]) {

cout << G.VerList[w] << "\t";

break;

}

else {

w = nextAdj(G, v, w);

}

}

v = w;

}

}

void KeyPath(Graph1& G, int topoList[]) {

int i, j;

int vPre;//保存顶点的前驱顶点编号

int vSuc;//保存顶点的后继顶点编号

int vet[MaxVerNum + 1];

int vlt[MaxVerNum + 1];

for (i = 1; i <= G.VerNum; i++) {//初始化最早发生时间为0

vet[i] = 0;

}

for (i = 1; i <= G.VerNum; i++) {

vPre = topoList[i];

for (j = 1; j <= G.VerNum; j++) {

if (G.AdjMatrix[vPre][j] >= 1 && G.AdjMatrix[vPre][j] < INF) {

if (vet[j] < vet[vPre] + G.AdjMatrix[vPre][j]) {

vet[j] = vet[vPre] + G.AdjMatrix[vPre][j];

}

}

}

}

for (i = 1; i <= G.VerNum; i++) {//初始化vlt值为vet[G.VerNum]

vlt[i] = vet[G.VerNum];

}

for (i = G.VerNum; i >= 1; i--) {//按照逆拓扑次序求解最迟发生时间

vSuc = topoList[i];

for (j = G.VerNum; j >= 1; j--) {

if (G.AdjMatrix[j][vSuc] >= 1 && G.AdjMatrix[j][vSuc] < INF) {

if (vlt[j] > vlt[vSuc] - G.AdjMatrix[j][vSuc]) {

vlt[j] = vlt[vSuc] - G.AdjMatrix[j][vSuc];

}

}

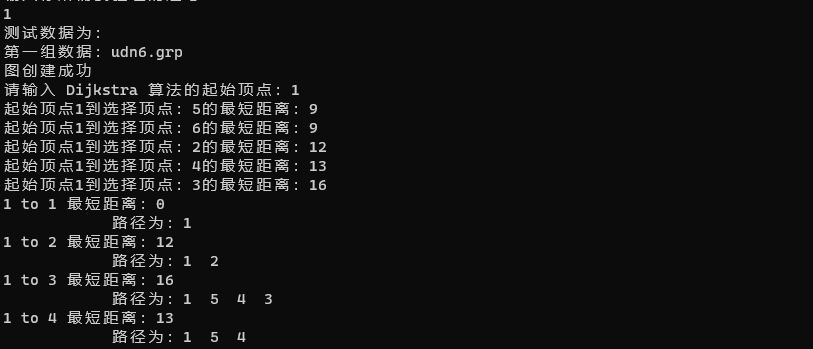
}

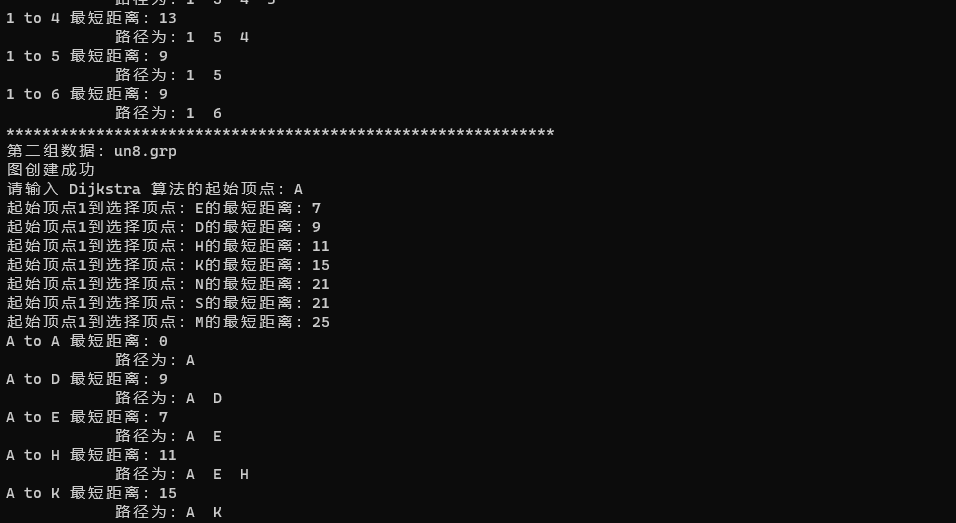
}

PrintKeyPath(G, topoList, vet, vlt);

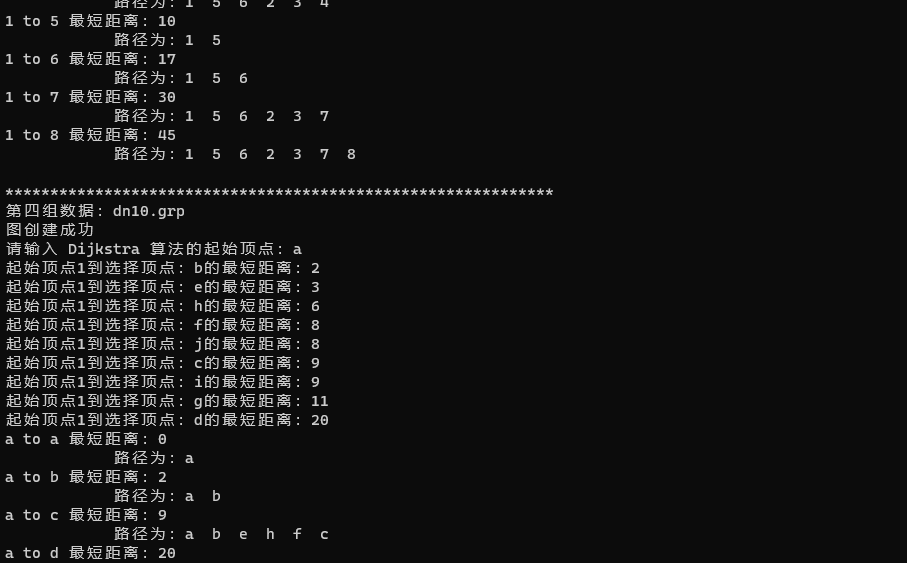
}

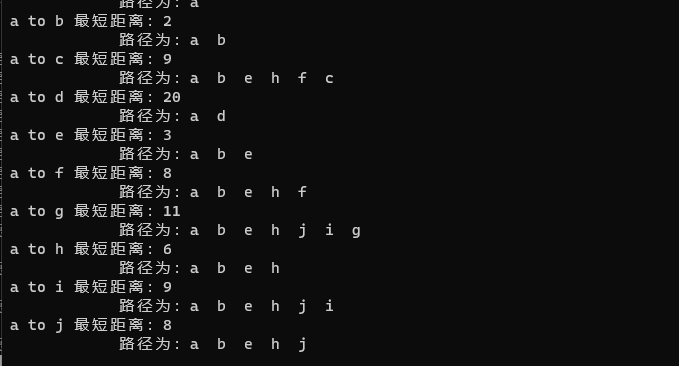
**7.6运行和测试**

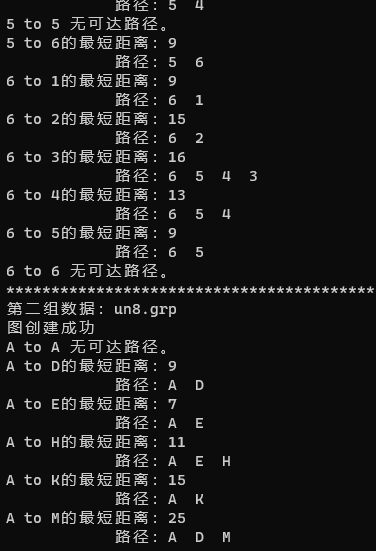
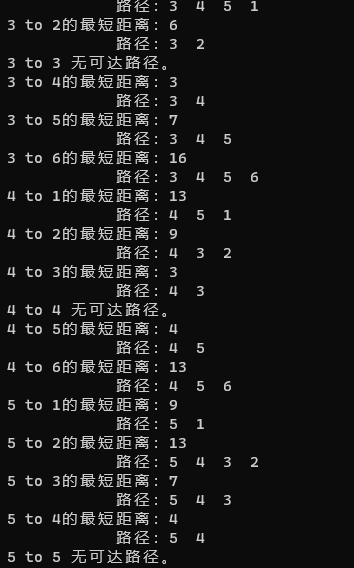
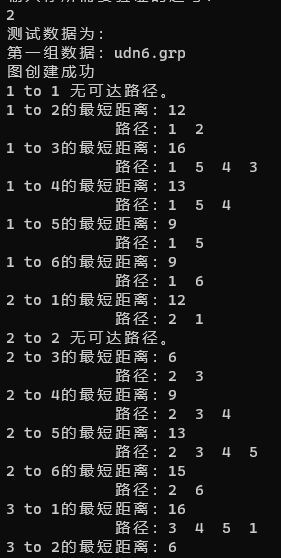
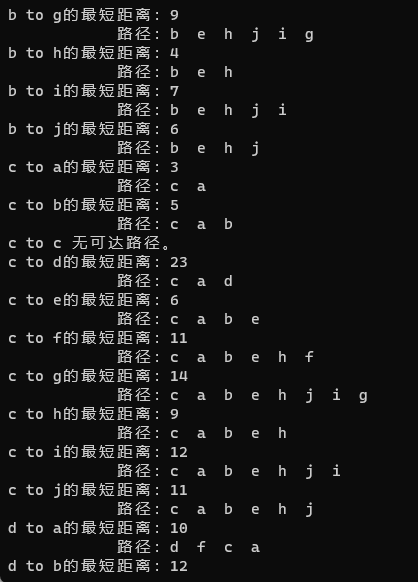
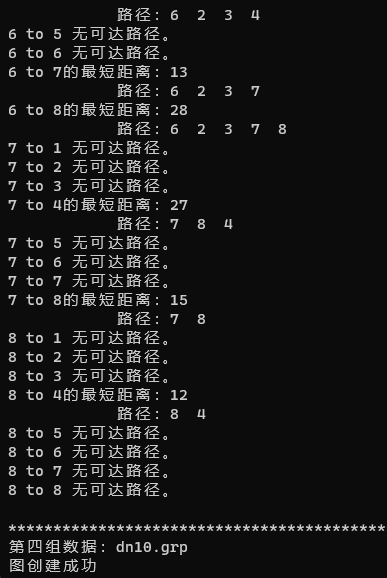
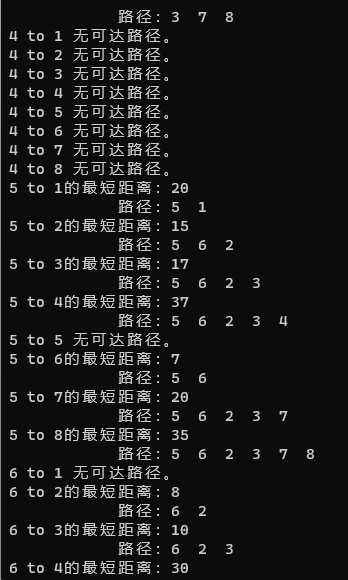
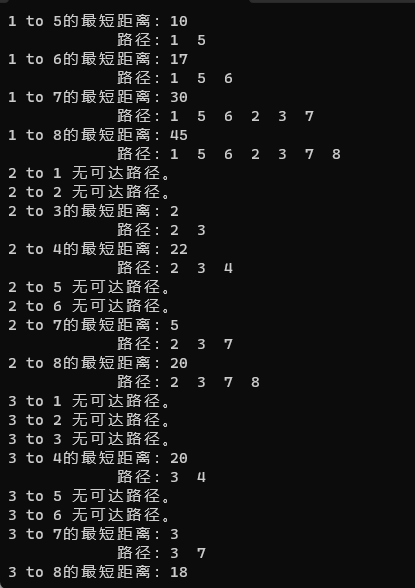
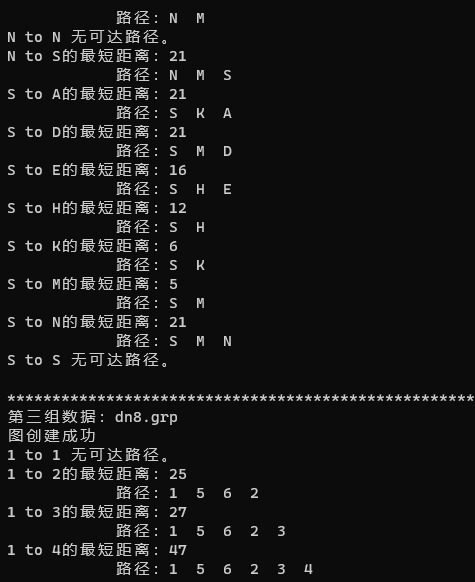
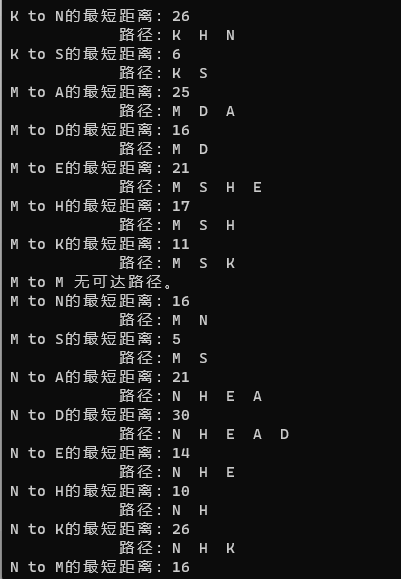
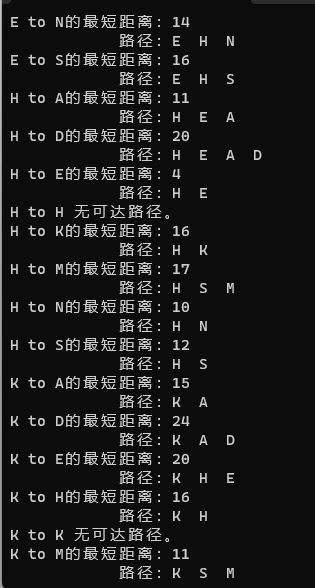
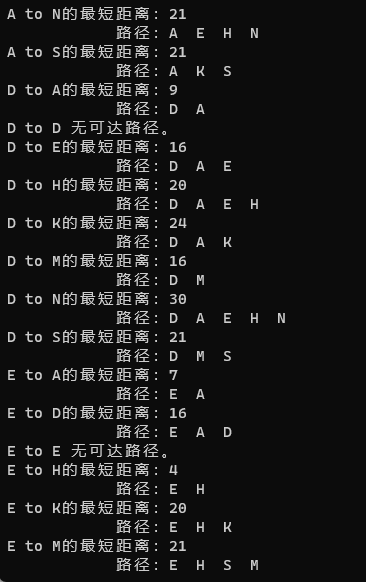


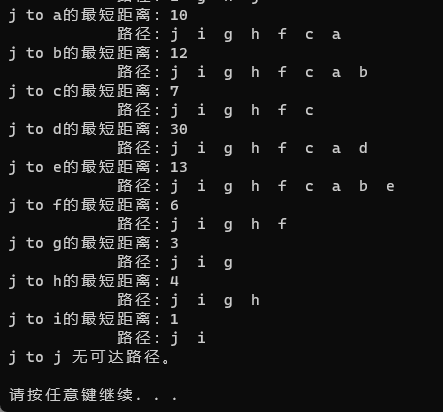
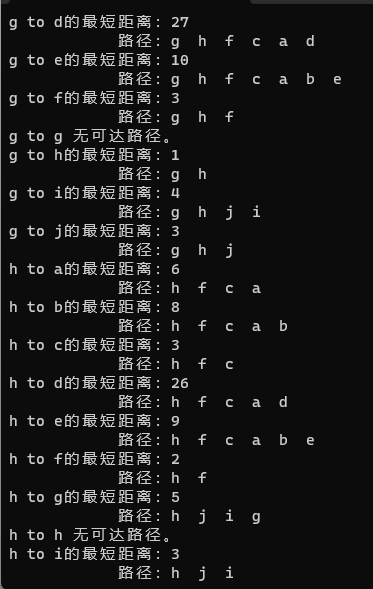
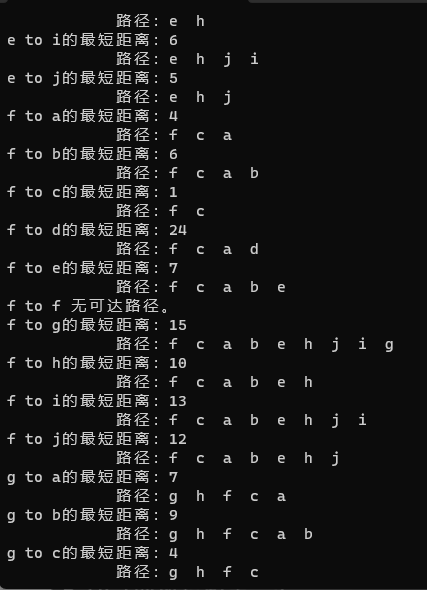


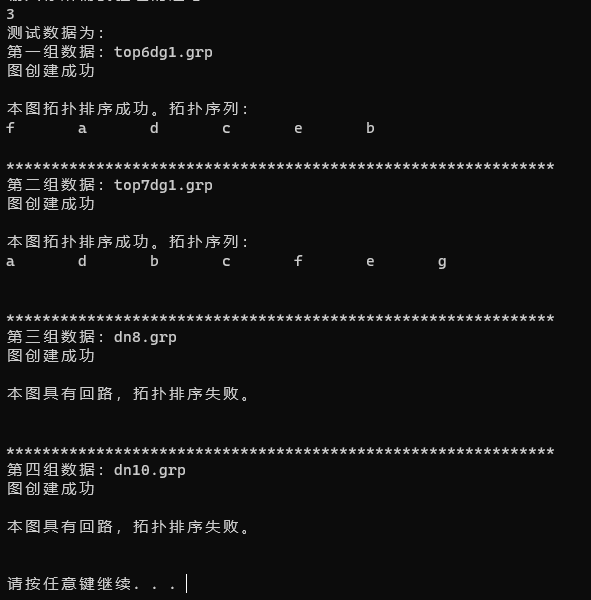


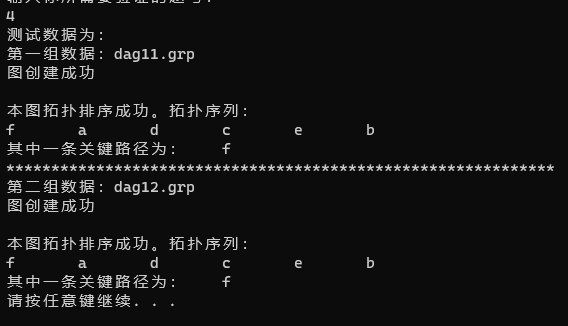










**7.7总结、心得和建议**

通过实验，我深刻理解了图的遍历、Dijkstra算法、Floyd算法、拓扑排序、AOE网的关键路径等内容，同时也掌握了C++语言在数据结构中的应用。

图的遍历是深入理解图结构的基础，掌握了遍历算法后，我们可以更好地理解图的性质和特点。Dijkstra算法是一种常用的求解最短路径的算法，但是需要限制图为非负边权的有向图，同时对于图中节点较多的情况，时间复杂度较高，需要优化。Floyd算法是求解任意两点间最短路径的算法，相对于Dijkstra算法，Floyd算法的时间复杂度较低，但是空间复杂度较高。拓扑排序是一种对有向无环图进行排序的算法，可以得到图的拓扑序列。在实际应用中，拓扑排序可以用于任务调度等问题。AOE网是一种带权有向无环图，可以用于流程管理等方面。求解AOE网的关键路径可以确定一个项目的最短完成时间以及所有关键活动。

在使用Dijkstra算法求最短路径时，需要使用边权非负的有向图，否则可能会导致结果不正确。在使用Floyd算法求最短路径时，需要注意矩阵初始化值和求解过程中的顺序，否则会影响结果的准确性。在求拓扑序列时，需要注意图是否为有向无环图，否则无法得到正确的拓扑序列。在求AOE网的关键路径时，需要先进行拓扑排序和计算最早时间和最晚时间，才能得到关键路径。

本次实验难度很大，通过实验，我加深了对数据结构图算法的认识和了解。我进一步掌握了数据结构的基本原理和实现方法，并深刻认识到合适的数据结构可以大大提高程序的效率和运行速度。我相信这些知识在以后的学习、工作和研究中都会有所帮助。

**[7.8附录]**

（源代码清单。纸质报告不做要求。电子版报告，可直接附源文件，删除编译生成的所有文件）

（邻接矩阵 需要对应的头文件编译才能运行）

（邻接表 需要对应的头文件编译才能运行）