**《数据结构》**

**实验报告**

专 业 软件工程

年 级 2018级

姓 名 谢梓聪

学 号 201824100445

实 验 室 信工北414

使用日期 2019年12月5日

郑州大学信息工程学院

# 实验七 图的深度优先遍历

**一、实验目的及要求：**

熟悉图的数组表示法和邻接表存储结构，掌握构造有向图、无向图的算法 ，在掌握以上知识的基础上，熟悉图的深度优先遍历算法，并实现。

**二、实验方法：**

1．使用windows平台。

2．使用DEV-C或Visual C软件。

**三、实验内容：**

（1）图的数组表示法定义及基本操作的实现。

（2）图的邻接表表示法定义及基本操作的实现。

（3）写函数实现图的深度优先遍历（分别在两种结构上）

（4）在邻接表上实现拓扑排序、关键路径的求法，在邻接矩阵上实现最短路经、最小生成树的求法。

**四、实验结果：**

***base.h***

#ifndef BASE\_H

#define BASE\_H

#include<stdio.h>

#include<stdlib.h>

#include<limits.h> //INT\_MAX的所在库

#include<string.h>

#define OVERFLOW -2

#define OK 1

#define ERROR 0

#define INFINITY 1000000

#define MAX\_VERTEX\_NUM 20

#define STACK\_INIT\_SIZE 50

#define STACKINCREMENT 10

typedef int Status;

typedef int InfoType;

typedef char VertexType;

typedef int SElemType;

typedef struct SqStack{

SElemType \*base;

SElemType \*top;

int stackSize;

}SqStack;

typedef struct Edge{

int lowcost;

VertexType adjvex;

}Edge; //代表一整条边，下标为一个结点，adj域为与该点邻接的权值最小的邻接点，lowcost对应权值

Status VexOutput(VertexType vex);

Status InitStack(SqStack &S); //栈相关函数

Status Pop(SqStack &S,SElemType &del);

Status Push(SqStack &S,SElemType add);

Status StackEmpty(SqStack &S);

#endif

***ALGraph.h***

/\* ALGraph.h \*/

#ifndef ALGRAPH\_H

#define ALGRAPH\_H

#include "base.h"

typedef struct ArcNode{

int adjvex; //该弧所指向的弧头的顶点位置

struct ArcNode \*nextarc; //该弧指向的下一条弧的指针

InfoType \*info; //该弧相关信息的指针

}ArcNode;

typedef struct VNode{ //顶点结点类型

VertexType data; //顶点信息

ArcNode \*firstarc; //指向第一条依附该顶点的弧的指针

}VNode,AdjList[MAX\_VERTEX\_NUM];

typedef struct{

AdjList vexs; //邻接表的头结点数组

int vexnum,arcnum;

int kind;

}ALGraph;

int LocateVex\_AL(ALGraph &G,VertexType vex);

Status CreateUDN\_AL(ALGraph &G);

Status CreateDN\_AL(ALGraph &G);

Status CreateUDG\_AL(ALGraph &G);

Status CreateDG\_AL(ALGraph &G);

Status CreateALGraph(ALGraph &G);

void DFSTraverse\_AL(ALGraph G,Status(\*Visit)(VertexType vex)); //邻接表方式的深度优先遍历

void DFS\_AL(ALGraph G,int vex,Status(\*Visit)(VertexType vex)); //对每个顶点的邻接点进行遍历的递归函数

#endif // ALGRAPH\_H

***MGraph.h***

#ifndef MGRAPH\_H

#define MGRAPH\_H

#include "base.h"

typedef struct ArcCell{

int adj; //VRType是顶点关系类型，无权图：1 相邻,0：不相邻；有权图：权值

InfoType \*info; //该弧相关信息的指针

}ArcCell,AdjMatrix[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM]; //Adjacency Matrix 邻接矩阵

typedef struct{

VertexType vexs[MAX\_VERTEX\_NUM]; //顶点向量

AdjMatrix arcs; //弧所构成的矩阵

int vexnum,arcnum; //顶点个数和弧或边的个数

int kind; //图的种类

}MGraph;

Status CreateMGraph(MGraph &G);

Status CreateUDN\_M(MGraph &G);

Status CreateUDG\_M(MGraph &G);

Status CreateDN\_M(MGraph &G);

Status CreateDG\_M(MGraph &G);

int LocateVex(MGraph &G,VertexType vex);

void DFSTraverse(MGraph G,Status(\*Visit)(VertexType vex)); //邻接矩阵的深度优先遍历

void DFS(MGraph G,int vex,Status(\*Visit)(VertexType vex)); //对每个顶点的邻接点进行遍历的递归函数

#endif // MGRAPH\_H

***ALGraph.cpp***

#include "ALGraph.h"

int visited\_AL[MAX\_VERTEX\_NUM]={0}; //顶点访问标志数组

void DFSTraverse\_AL(ALGraph G,Status(\*Visit)(VertexType vex))

{

int i;

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

{

if(!visited\_AL[i])

DFS\_AL(G,i,Visit);

}

}

void DFS\_AL(ALGraph G,int vex,Status(\*Visit)(VertexType vex))

{

int i=vex;

ArcNode \*temp=G.vexs[i].firstarc->nextarc;

visited\_AL[i]=1;

Visit(G.vexs[i].data);

while(temp!=NULL)

{

if(!visited\_AL[temp->adjvex])

{

DFS\_AL(G,temp->adjvex,Visit);

}

temp=temp->nextarc;

}

}

int LocateVex\_AL(ALGraph &G,VertexType vex)

{

int i;

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

{

if(G.vexs[i].data==vex)

return i;

}

return 0;

}

Status CreateUDN\_AL(ALGraph &G) //创建无向网

{

int i,loc1,loc2,\*weight;

char vex1,vex2;

ArcNode \*ins=NULL;

printf("请输入该图的顶点个数和弧的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请依次输入各顶点：");

for(i=0;i<G.vexnum;i++) //顶点向量初始化

{

scanf("%c",&G.vexs[i].data);

G.vexs[i].firstarc=(ArcNode\*)malloc(sizeof(ArcNode));

G.vexs[i].firstarc->nextarc=NULL;

}

getchar();

printf("请输入各弧所对应的两个顶点值和该弧对应的权值：");

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

{

weight=(int \*)malloc(sizeof(int));

scanf("%c %c %d",&vex1,&vex2,weight);

getchar();

loc1=LocateVex\_AL(G,vex1); //只计算弧尾所对应的顶点序号

loc2=LocateVex\_AL(G,vex2);

ins=(ArcNode\*)malloc(sizeof(ArcNode));

ins->adjvex=loc2; //存弧头的顶点的序号

ins->info=weight; //存该弧所对应的权值

ins->nextarc=G.vexs[loc1].firstarc->nextarc; //用头插法将新结点插入

G.vexs[loc1].firstarc->nextarc=ins;

}

return OK;

}

Status CreateDN\_AL(ALGraph &G) //创建有向网

{

int i,\*weight,loc1,loc2;

char vex1,vex2;

ArcNode \*ins=NULL;

printf("请输入该图的顶点个数和弧的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请依次输入各顶点：");

for(i=0;i<G.vexnum;i++) //顶点向量初始化

{

scanf("%c",&G.vexs[i].data);

G.vexs[i].firstarc=(ArcNode\*)malloc(sizeof(ArcNode));

G.vexs[i].firstarc->nextarc=NULL;

}

getchar();

printf("请输入各弧所对应的两个顶点值和该弧所对应的权值：");

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

{

weight=(int \*)malloc(sizeof(int));

scanf("%c %c %d",&vex1,&vex2,weight);

getchar();

loc1=LocateVex\_AL(G,vex1); //只计算弧尾所对应的顶点序号

loc2=LocateVex\_AL(G,vex2);

ins=(ArcNode\*)malloc(sizeof(ArcNode));

ins->adjvex=loc2; //存弧头的顶点的序号

ins->info=weight; //存该弧所对应的权值

ins->nextarc=G.vexs[loc1].firstarc->nextarc; //用头插法将新结点插入

G.vexs[loc1].firstarc->nextarc=ins;

}

return OK;

}

Status CreateDG\_AL(ALGraph &G)

{

int i,loc1,loc2;

char vex1,vex2;

ArcNode \*ins=NULL;

printf("请输入该图的顶点个数和弧的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请依次输入各顶点：");

for(i=0;i<G.vexnum;i++) //顶点向量初始化

{

scanf("%c",&G.vexs[i].data);

G.vexs[i].firstarc=(ArcNode\*)malloc(sizeof(ArcNode));

G.vexs[i].firstarc->nextarc=NULL;

}

getchar();

printf("请输入各弧所对应的两个顶点值：");

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

{

scanf("%c %c",&vex1,&vex2);

getchar();

loc1=LocateVex\_AL(G,vex1); //只计算弧尾所对应的顶点序号

loc2=LocateVex\_AL(G,vex2);

ins=(ArcNode\*)malloc(sizeof(ArcNode));

ins->adjvex=loc2; //存弧头的顶点的序号

ins->nextarc=G.vexs[loc1].firstarc->nextarc; //用头插法将新结点插入

G.vexs[loc1].firstarc->nextarc=ins;

}

return OK;

}

Status CreateUDG\_AL(ALGraph &G)

{

int i,loc1,loc2;

char vex1,vex2;

ArcNode \*ins=NULL;

printf("请输入该图的顶点个数和边的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请依次输入各顶点：");

for(i=0;i<G.vexnum;i++) //顶点向量初始化

{

scanf("%c",&G.vexs[i].data);

G.vexs[i].firstarc=(ArcNode\*)malloc(sizeof(ArcNode));

G.vexs[i].firstarc->nextarc=NULL;

}

getchar();

printf("请输入各弧所对应的两个顶点值：");

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

{

scanf("%c %c",&vex1,&vex2); //将两条边均连入链表

getchar();

loc1=LocateVex\_AL(G,vex1);

loc2=LocateVex\_AL(G,vex2);

ins=(ArcNode\*)malloc(sizeof(ArcNode));

ins->adjvex=loc2;

ins->nextarc=G.vexs[loc1].firstarc->nextarc;

G.vexs[loc1].firstarc->nextarc=ins;

ins=(ArcNode\*)malloc(sizeof(ArcNode));

ins->adjvex=loc1;

ins->nextarc=G.vexs[loc2].firstarc->nextarc;

G.vexs[loc2].firstarc->nextarc=ins;

}

return OK;

}

Status CreateALGraph(ALGraph &G)

{

printf("请输入要创建的图的类型：0，1，2，3分别表示有向图和网、无向图和网\n");

scanf("%d",&G.kind);

switch(G.kind)

{

case 0: CreateDG\_AL(G);return OK;

case 1: CreateDN\_AL(G);return OK;

case 2: CreateUDG\_AL(G);return OK;

case 3: CreateUDN\_AL(G);return OK;

}

return OK;

}

***MGraph.cpp***

#include "MGraph.h"

int visited[MAX\_VERTEX\_NUM]={0}; //顶点访问标志数组

void DFSTraverse(MGraph G,Status(\*Visit)(VertexType vex))

{

int i;

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

{

if(!visited[i])

DFS(G,i,Visit);

}

}

void DFS(MGraph G,int vex,Status(\*Visit)(VertexType vex))

{

int i=vex,j;

visited[i]=1;

Visit(G.vexs[i]);

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

{

if((((G.kind==0||G.kind==2)&&G.arcs[i][j].adj)||((G.kind==1||G.kind==3)&&G.arcs[i][j].adj!=INFINITY))&&!visited[j])

DFS(G,j,Visit);

}

}

int LocateVex(MGraph &G,VertexType vex)

{

int i;

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

{

if(G.vexs[i]==vex)

return i;

}

return -1;

}

Status CreateUDN\_M(MGraph &G) //创建无向网

{

int i,j,weight,loc1,loc2;

char vex1,vex2;

printf("请输入顶点的个数和边的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请依次输入各顶点：");

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

scanf("%c",&G.vexs[i]);

getchar();

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

{

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

{

G.arcs[i][j].adj=INFINITY;

G.arcs[i][j].info=NULL;

}

}

printf("请依次输入各条边所对应的两顶点值和权值：");

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

{

scanf("%c %c %d",&vex1,&vex2,&weight);

getchar();

loc1=LocateVex(G,vex1);

loc2=LocateVex(G,vex2);

G.arcs[loc1][loc2].adj=weight;

G.arcs[loc2][loc1]=G.arcs[loc1][loc2];

}

return OK;

}

Status CreateDN\_M(MGraph &G) //创建有向网

{

int i,j,weight,loc1,loc2;

char vex1,vex2;

printf("请输入顶点的个数和边的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

printf("请依次输入各顶点：");

getchar();

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

scanf("%c",&G.vexs[i]);

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

{

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

{

G.arcs[i][j].adj=INFINITY;

G.arcs[i][j].info=NULL;

}

}

getchar();

printf("请依次输入各条弧所对应的顶点值和权值：");

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

{

scanf("%c %c %d",&vex1,&vex2,&weight);

getchar();

loc1=LocateVex(G,vex1);

loc2=LocateVex(G,vex2);

G.arcs[loc1][loc2].adj=weight;

}

return OK;

}

Status CreateUDG\_M(MGraph &G)

{

int i,j,loc1,loc2;

char vex1,vex2;

printf("请输入顶点的个数和弧的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

printf("请依次输入各顶点：");

getchar();

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

scanf("%c",&G.vexs[i]);

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

{

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

{

G.arcs[i][j].adj=0;

G.arcs[i][j].info=NULL;

}

}

getchar();

printf("请依次输入各条弧的两个顶点值：");

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

{

scanf("%c %c",&vex1,&vex2);

getchar();

loc1=LocateVex(G,vex1);

loc2=LocateVex(G,vex2);

G.arcs[loc1][loc2].adj=1;

G.arcs[loc2][loc1]=G.arcs[loc1][loc2];

}

return OK;

}

Status CreateDG\_M(MGraph &G)

{

int i,j,loc1,loc2;

char vex1,vex2;

printf("请输入顶点的个数和弧的个数：");

scanf("%d %d",&G.vexnum,&G.arcnum);

getchar();

printf("请依次输入各顶点：");

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

scanf("%c",&G.vexs[i]);

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

{

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

{

G.arcs[i][j].adj=0;

G.arcs[i][j].info=NULL;

}

}

getchar();

printf("请依次输入各条弧所对应的顶点值：");

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

{

scanf("%c %c",&vex1,&vex2);

getchar();

loc1=LocateVex(G,vex1);

loc2=LocateVex(G,vex2);

G.arcs[loc1][loc2].adj=1;

}

return OK;

}

Status CreateMGraph(MGraph &G)

{

printf("请输入要创建的图的类型：0，1，2，3分别表示有向图和网、无向图和网\n");

scanf("%d",&G.kind);

switch(G.kind)

{

case 0: CreateDG\_M(G);return OK;

case 1: CreateDN\_M(G);return OK;

case 2: CreateUDG\_M(G);return OK;

case 3: CreateUDN\_M(G);return OK;

}

return OK;

}

***main.cpp***

#include "base.h"

#include "MGraph.h"

#include "ALGraph.h"

void MiniSpanTree\_PRIM(MGraph G,VertexType vex); //利用prim算法去求一个图的最小生成树(MST)

int minEdge(Edge \*closedge,MGraph G);

Status TopologicalSort(ALGraph G);

void FindInDegree(int \*indegree,ALGraph G);

Status TopologicalOrder(ALGraph G,SqStack &T,int \*ves);

Status CriticalPath(ALGraph &G);

void ShortestPath\_DIJ(MGraph &G,VertexType vex0,int P[][MAX\_VERTEX\_NUM],int D[]);

/\*样例输入：

8 9

ABCDEFGH

图输入：AB AC BD BE DH EH CF CG FG

网输入：AB 1 AC 2 BD 1 BE 3 DH 3 EH 2 CF 1 CG 2 FG 3

输出： 邻接矩阵存储法深度优先遍历：ABDHECFG

邻接表存储法深度优先遍历(头插法，先输入的边后输出邻接点）：ACGFBEHD

最小生成树输入： （无向网）

6 10

ABCDEF

AB 6 AC 1 AD 5 BC 5 CD 5 BE 3 CE 6 CF 4 EF 6 DF 2

输出：AC CF FD CB BE,从不同起始点得出的最小生成树是相同的

拓扑排序输入： （有向无环图）

6 8

ABCDEF

AB AC AD CD EB EF BF CF

输出：E A B C D F

有环图： AB AC DA CD EB EF BF CF

关键路径输入： （有向无环网）

6 8

ABCDEF

AB 3 AC 2 BD 2 BE 3 CD 4 CF 3 DF 2 EF 1

输出：关键路径：A->C->D->F

9 11

123456789

12 6 13 4 14 5 25 1 35 1 46 2 57 9 58 7 68 4 79 2 89 4

输出：关键活动： 12 25 57 58 79 89

最短路径输入： （有向网）

6 8

012345

05 100 04 30 02 10 12 5 23 50 43 20 35 10 45 60

输出：0 1000000 10 50 30 60

\*/

int main(void)

{

int P[MAX\_VERTEX\_NUM][MAX\_VERTEX\_NUM]={0},D[MAX\_VERTEX\_NUM]={0},i;

MGraph MG;

CreateMGraph(MG);

//ALGraph ALG;

//CreateALGraph(ALG);

// 邻接矩阵深度优先遍历

//DFSTraverse(MG,VexOutput);

// 邻接表深度优先遍历

//DFSTraverse\_AL(ALG,VexOutput);

//邻接矩阵上实现最小生成树的求法,创建图为无向网

//MiniSpanTree\_PRIM(MG,MG.vexs[1]);

//邻接表上实现拓扑排序

//if(!TopologicalSort(ALG))

// printf("该有向图不为无环图");

//if(!CriticalPath(ALG))

// printf("该有向图不为无环图\n");

//最短路径

ShortestPath\_DIJ(MG,MG.vexs[0],P,D);

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

{

printf("%d\t",D[i]);

}

return 0;

}

void ShortestPath\_DIJ(MGraph &G,VertexType vex0,int P[][MAX\_VERTEX\_NUM],int D[])

{

int final[MAX\_VERTEX\_NUM]={0};

int i,j,v,min;

int v0=LocateVex(G,vex0);

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

{

D[v]=G.arcs[v0][v].adj;

if(D[v]<INFINITY)

{

P[v][v0]=1;

P[v][v]=1;

}

}

D[v0]=0;

final[v0]=1; //v0顶点属于S集合

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

{

min=INFINITY;

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

{

if(!final[j])

{

if(D[j]<min)

{

min=D[j];

v=j;

}

}

}

final[v]=1;

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

{

if(!final[j]&&(D[j]>G.arcs[v][j].adj+min))

{

D[j]=G.arcs[v][j].adj+min;

P[j][j]=1;

}

}

}

}

Status CriticalPath(ALGraph &G)

{

int ve[MAX\_VERTEX\_NUM]={0},vl[MAX\_VERTEX\_NUM]={0},i,k;

char tag[30];

ArcNode \*temp;

SqStack T;

if(!TopologicalOrder(G,T,ve))

return ERROR;

vl[G.vexnum-1]=ve[G.vexnum-1]; //逆拓扑有序序列从n-1开始递推

while(!StackEmpty(T))

{

Pop(T,i);

for(temp=G.vexs[i].firstarc->nextarc;temp;temp=temp->nextarc)

{

if(!temp) break;

if(vl[i]==0) vl[i]=vl[temp->adjvex]-\*(temp->info); //保证vl[i]的值是vl[temp->adjvex]-\*(temp->info)的最小值

if(vl[temp->adjvex]-\*(temp->info)<vl[i])

vl[i]=vl[temp->adjvex]-\*(temp->info);

}

}

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

{

for(temp=G.vexs[i].firstarc->nextarc;temp;temp=temp->nextarc)

{

if(!temp) break;

k=temp->adjvex;

if(ve[i]==vl[k]-\*(temp->info))

{

strcpy(tag,"该活动为关键活动");

}

else

strcpy(tag," ");

printf("活动%c->%c 耗时%d 最早时间%d 最晚时间%d %s\n",G.vexs[i].data,G.vexs[k].data,\*(temp->info),ve[i],vl[k]-\*(temp->info),tag);

}

}

return OK;

}

Status TopologicalOrder(ALGraph G,SqStack &T,int \*ve)

{

int indegree[MAX\_VERTEX\_NUM]={0},i,count=0;

ArcNode \*temp;

SqStack S;

InitStack(S);

InitStack(T);

FindInDegree(indegree,G);

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

{

if(!indegree[i])

Push(S,i);

}

while(!StackEmpty(S))

{

Pop(S,i);

Push(T,i);

count++;

for(temp=G.vexs[i].firstarc->nextarc;temp;temp=temp->nextarc)

{

if(!temp) break;

if(!--indegree[temp->adjvex])

{

Push(S,temp->adjvex);

}

if(ve[i]+\*(temp->info)>ve[temp->adjvex])

ve[temp->adjvex]=ve[i]+\*(temp->info);

}

}

if(count<G.vexnum)

return ERROR;

else

return OK;

}

Status TopologicalSort(ALGraph G)

{

int indegree[MAX\_VERTEX\_NUM]={0},i,count=0;

ArcNode \*temp;

SqStack S;

InitStack(S);

FindInDegree(indegree,G);

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

{

if(!indegree[i])

Push(S,i);

}

while(!StackEmpty(S))

{

Pop(S,i);

printf("%c ",G.vexs[i].data);

count++;

for(temp=G.vexs[i].firstarc->nextarc;temp;temp=temp->nextarc)

{

if(!temp) break;

if(!--indegree[temp->adjvex])

{

Push(S,temp->adjvex);

}

}

}

if(count<G.vexnum)

return ERROR;

else

return OK;

}

void FindInDegree(int \*indegree,ALGraph G)

{

int i;

ArcNode \*temp;

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

{

for(temp=G.vexs[i].firstarc->nextarc;temp;temp=temp->nextarc)

{

if(!temp) break;

indegree[temp->adjvex]++;

}

}

}

void MiniSpanTree\_PRIM(MGraph G,VertexType vex)

{

int k,i,j,fix;

Edge closedge[MAX\_VERTEX\_NUM];

fix=k=LocateVex(G,vex);

for(i=0;i<G.vexnum;i++) //辅助数组初始化

{

if(i!=k)

{

closedge[i].adjvex=vex; //依附到U集合中的顶点值

closedge[i].lowcost=G.arcs[k][i].adj;

}

}

closedge[k].lowcost=0;

closedge[k].adjvex=vex;

for(i=0;i<G.vexnum;i++) //开始依次找与第k个结点依附的权值最小的边并开始并入集合U中

{

if(i!=fix)

{

k=minEdge(closedge,G); //找到第k个结点依附的权值最小的边所对应的邻接点序号

printf("%c %c\n",closedge[k].adjvex,G.vexs[k]);

closedge[k].lowcost=0; //并入集合U中

for(j=0;j<G.vexnum;j++) //更新closedge的值,从U到V-U中的相对应的V-U中的顶点最小权值序偶可能改变

{

if(G.arcs[k][j].adj<closedge[j].lowcost)

{

closedge[j].lowcost=G.arcs[k][j].adj;

closedge[j].adjvex=G.vexs[k];

}

}

}

}

}

int minEdge(Edge \*closedge,MGraph G)

{

int i,minVex;

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

{

if(closedge[i].lowcost!=0) //找一个符合条件即lowcost不为0的minVex值

{

minVex=i;

break;

}

}

for(i=minVex+1;i<G.vexnum;i++)

{

if(minVex!=i&&closedge[i].lowcost!=0&&closedge[minVex].lowcost>closedge[i].lowcost)

minVex=i;

}

return minVex;

}

Status VexOutput(VertexType vex)

{

printf("%c",vex);

return OK;

}

Status InitStack(SqStack &S)

{

S.base=(SElemType\*)malloc(STACK\_INIT\_SIZE\*sizeof(SElemType));

if(!S.base) exit(ERROR);

S.stackSize=STACK\_INIT\_SIZE;

S.top=S.base;

return OK;

}

Status Pop(SqStack &S,SElemType &del)

{

if(S.top==S.base) exit(ERROR);

del=\*--S.top;

return OK;

}

Status Push(SqStack &S,SElemType add)

{

if(S.top-S.base>=S.stackSize)

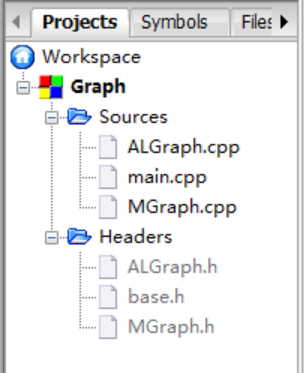
{

S.base=(SElemType \*)realloc(S.base,(S.stackSize+STACKINCREMENT)\*sizeof(SElemType));

if(!S.base) exit(OVERFLOW);

S.stackSize+=STACKINCREMENT;

S.top=S.base+STACKINCREMENT;

 }

\*S.top++=add;

return OK;

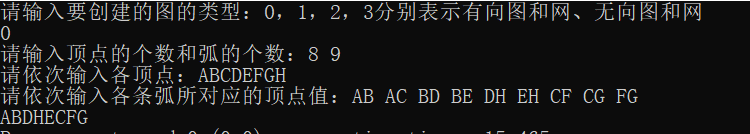
}

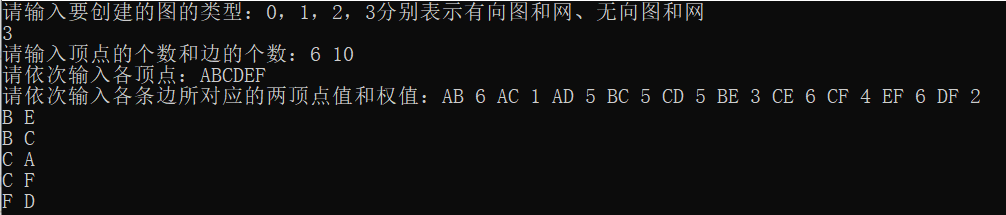
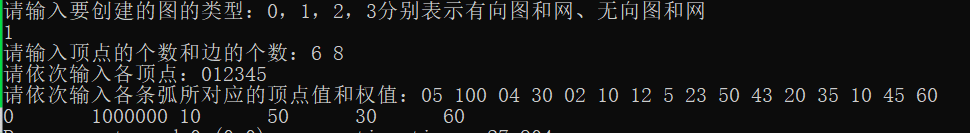
Status StackEmpty(SqStack &S)

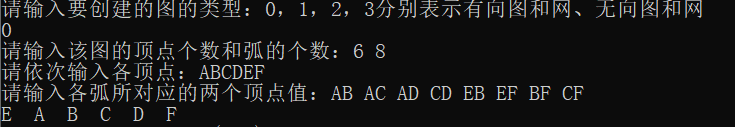
{

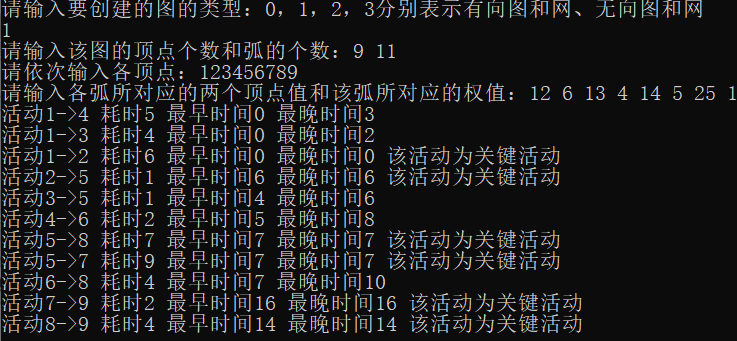
return S.top==S.base;

}









**五、实验体会：**

该程序使用CodeBlocks编辑器进行编写一个项目程序，包含3个头文件和3个源文件，包括邻接矩阵图和邻接表图的头和源文件以及通用头文件base.h和主程序源文件main.cpp，main.cpp内包含深度遍历，最小生成树生成，关键路径，最短路径以及拓扑排序的函数定义，图为一种多对多的较为复杂逻辑结构，可使用邻接矩阵和邻接表两种物理存储方式存储，邻接矩阵易得任意两顶点的关系和边的权值，邻接表易得任意一个顶点的邻接点以及相关边信息。图主要分为有向图、无向图、有向网、无向网，图主要遍历方式为深度和广度优先遍历，然后图有着广泛的工程应用，包含最小生成树来生成一个地区多个城市之间花费最小的路径组的建立，拓扑排序可判断一个图是否为有向无环图来判断工程各事件是否存在自己以自己为优先的矛盾，关键路径可得出一个工程内部的关键活动和关键路径以便缩短工期，最短路径可得到两个城市间的多条路径的最小花费选择等等。