第7章图的基本操作

1.实验目的：

理解图的基本概念，两种主要的存储结构。掌握在邻接链表存储结构下的图的深度优先递归遍历、广度优先遍历。通过选做题"最短路径问题"认识图及其算法具有广泛的应用意义。

2.实验要求：正确调试程序。写出实验报告。

/\*预处理部分\*/

#include <stdio.h>

#include <stdlib.h>

#define INFINITY 2147483647 //4字节有符号整型数最大值

#define MAX 20 //图的最大顶点个数

#define OK 1

#define ERROR 0

/\*结构类型定义部分\*/

typedef char VertexType; //图顶点的数据类型

typedef int Status;

typedef int VRType; //顶点的关系集的数据类型

typedef enum {DG,DN,UDG,UDN}GraphKind; //{有向图，有向网，无向图，无向网}

typedef struct {

VertexType vexs[MAX]; //顶点集

VRType arcs[MAX][MAX]; //关系集

int vexnum; //顶点个数

int arcnum; //关系数

GraphKind kind; //图的类型

}MGraph;

typedef struct {

VertexType adjvex;

VRType lowcost;

}edge;

MGraph G[4]; //全局变量，用以分别存储4中不同类型的图

int visited[MAX]; //全局变量，用以标记访问过的顶点下标

Status (\*VisitFunc) (MGraph G,int v); //全局函数指针

edge closedge[MAX]; //全局变量，构造最小生成树的边

/\*函数声明部分\*/

int LocateVex (MGraph G,VertexType v);

Status CreateDG (MGraph &G);

Status CreateUDG (MGraph &G);

Status CreateDN (MGraph &G);

Status CreateUDN (MGraph &G);

Status PrintVex (MGraph G,int v);

int FirstAdjVex (MGraph G,int v);

int NextAdjVex (MGraph G,int v,int w);

void DFS (MGraph G,int v);

void DFSTraverse (MGraph G,Status (\*Visit) (MGraph G,int v));

void BFSTraverse (MGraph G,Status (\*Visit) (MGraph G,int v));

Status ShortestPath\_DIJ (MGraph G,int v0,int P[][MAX],int \*D);

Status PrintShortestPath (MGraph G,int v0,int P[][MAX],int \*D);

int mininum (edge \*closedge,int v);

Status MiniSpanTree\_PRIM (MGraph G,int v0);

/\*函数定义部分\*/

int LocateVex (MGraph G,VertexType v) {

//定位顶点集中的元素

int i=0;

while ((v!=G.vexs[i])&&(i<G.vexnum)) i++;

return i;

} //end LocateVex

Status CreateDG (MGraph &G) {

//创建有向图

int i=0,j=0,k=0;VertexType v1,v2;

printf ("请输入图中的顶点总数：");

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

printf ("请输入弧总数：");

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

printf ("请输入%d个顶点值：",G.vexnum);

getchar ();

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

scanf ("%c",&G.vexs[i]); //输入图顶点值(注：%c前有空格时输入数据中间加空格如：a b c，无空格时输入数据中间不加空格如：abc)

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

for (j=0;j<G.vexnum;j++) G.arcs[i][j]=0; //初始化关系集

for ( ;k<G.arcnum;k++) { //构造邻接矩阵

printf ("请输入第%d对作为弧尾、弧头的元素：",k+1); //输入有关系的一对顶点

getchar ();

scanf ("%c%c",&v1,&v2);

i=LocateVex (G,v1);j=LocateVex (G,v2); //确定v1,v2在G的顶点集中的位置

if ((i==G.vexnum)||(j==G.vexnum)) {

printf ("弧端元素输入错误\n");return ERROR;

}

G.arcs[i][j]=1;

}

printf ("成功创建有向图\n");

return OK;

} //end CreateDG

Status CreateUDG (MGraph &G) {

//创建无向图

int i=0,j=0,k=0;VertexType v1,v2;

printf ("请输入图中的顶点总数：");

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

printf ("请输入边总数：");

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

printf ("请输入%d个顶点值：",G.vexnum);

getchar ();

for ( ;i<G.vexnum;i++) scanf (" %c",&G.vexs[i]); //输入图顶点值(注：%c前有空格时输入数据中间加空格如：a b c，无空格时输入数据中间不加空格如：abc)

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

for (j=0;j<G.vexnum;j++) G.arcs[i][j]=0; //初始化关系集

for ( ;k<G.arcnum;k++) { //构造邻接矩阵

printf ("请输入第%d对作为边两端的元素：",k+1); //输入有关系的一对顶点

getchar ();

scanf ("%c%c",&v1,&v2);

i=LocateVex (G,v1);j=LocateVex (G,v2); //确定v1,v2在G的顶点集中的位置

if ((i==G.vexnum)||(j==G.vexnum)) {

printf ("边端元素输入错误\n");return ERROR;

}

G.arcs[i][j]=1;G.arcs[j][i]=1;

}

printf ("成功创建无向图\n");

return OK;

} //end CreateUDG

Status CreateDN (MGraph &G) {

//创建有向网

int i=0,j=0,k=0,qz=0;VertexType v1,v2;

printf ("请输入图中的顶点总数：");

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

printf ("请输入弧总数：");

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

printf ("请输入%d个顶点值：",G.vexnum);

getchar ();

for ( ;i<G.vexnum;i++) scanf ("%c",&G.vexs[i]); //输入网顶点值

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

for (j=0;j<G.vexnum;j++) G.arcs[i][j]=INFINITY; //初始化关系集

for ( ;k<G.arcnum;k++) { //构造邻接矩阵

printf ("请输入第%d对作为弧尾、弧头的元素及权值：",k+1); //输入有关系的一对顶点

getchar ();

scanf ("%c%c%d",&v1,&v2,&qz);

i=LocateVex (G,v1);j=LocateVex (G,v2); //确定v1,v2在G的顶点集中的位置

if ((i==G.vexnum)||(j==G.vexnum)) {

printf ("弧端元素输入错误\n");return ERROR;

}

G.arcs[i][j]=qz;

}

printf ("成功创建有向网\n");

return OK;

} //end CreateDN

Status CreateUDN (MGraph &G) {

//创建无向网

int i=0,j=0,k=0,qz=0;VertexType v1,v2;

printf ("请输入图中的顶点总数：");

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

printf ("请输入边总数：");

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

printf ("请输入%d个顶点值：",G.vexnum);

getchar ();

for ( ;i<G.vexnum;i++) scanf ("%c",&G.vexs[i]); //输入网顶点值

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

for (j=0;j<G.vexnum;j++) G.arcs[i][j]=INFINITY; //初始化关系集

for ( ;k<G.arcnum;k++) { //构造邻接矩阵

printf ("请输入第%d对作为边端的两元素及权值：",k+1); //输入有关系的一对顶点

getchar ();

scanf ("%c%c%d",&v1,&v2,&qz);

i=LocateVex (G,v1);j=LocateVex (G,v2); //确定v1,v2在G的顶点集中的位置

if ((i==G.vexnum)||(j==G.vexnum)) {

printf ("边端元素输入错误\n");return ERROR;

}

G.arcs[i][j]=qz;G.arcs[j][i]=qz;

}

printf ("成功创建无向网\n");

return OK;

} //end CreateUDN

Status PrintVex (MGraph G,int v) {

//输出图的第v个顶点

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

return OK;

} //end PrintVex

int FirstAdjVex (MGraph G,int v) {

//在图G中查找第v个顶点的第1个邻接顶点

int i=0;

while ((i<G.vexnum)&&((G.arcs[v][i]==0)||(G.arcs[v][i]==INFINITY))) i++;

if (i==G.vexnum) return -1;

else return i;

} //end FirstAdjVex

int NextAdjVex (MGraph G,int v,int w) {

int i=0,k=0;

for (i=w+1;(i<G.vexnum)&&((G.arcs[v][i]==0)||(G.arcs[v][i]==INFINITY));i++) {}

if (i==G.vexnum) return -1;

else return i;

} //end NextAdjVex

void DFS (MGraph G,int v) {

//从第v个顶点递归地深度优先搜索遍历图G

int w=0;

VisitFunc (G,v);visited[v]=1; //访问顶点，并置标准数组中相应元素值为1

for (w=FirstAdjVex (G,v);w>=0;w=NextAdjVex (G,v,w))

if (!visited[w]) DFS (G,w); //对v的未访问的邻接顶点递归地调用DFS

} //end DFS

void DFSTraverse (MGraph G,Status (\*Visit) (MGraph G,int v)) {

//深度优先搜索遍历

int v=0;char y='\0';

printf ("请输入遍历首顶点：");

getchar ();

scanf ("%c",&y);

printf ("深度优先搜索遍历序列为：");

VisitFunc=Visit; //使用全局变量，使得DFS函数不必设函数参数

for ( ;v<G.vexnum;v++) visited[v]=0; //标志数组初始化

v=LocateVex (G,y);

DFS (G,v); //对首顶点调用DFS

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

if (!visited[v]) DFS (G,v); //对未访问的顶点调用DFS

} //end DFSTraverse

void BFSTraverse (MGraph G,Status (\*Visit) (MGraph G,int v)) {

//深度优先搜索遍历

int w=0,i=0,v=0,u=0;char y='\0';

int Queue[MAX]={0},front=0,rear=0;

printf ("请输入遍历首顶点：");

getchar ();

scanf ("%c",&y);

printf ("深度优先搜索遍历序列为：");

for ( ;v<G.vexnum;v++) visited[v]=0; //标志数组初始化

v=LocateVex (G,y);

i=v;

Visit (G,i);visited[i]=1;

Queue[rear++]=i; //i入队

while (front!=rear) {

u=Queue[front++]; //队头元素出队

for (w=FirstAdjVex (G,u);w>=0;w=NextAdjVex (G,u,w)) {

if (visited[w]==0) { //w为u的尚未访问的邻接顶点

Visit (G,w);visited[w]=1;

Queue[rear++]=w;

}

}

}

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

if (visited[i]==0) { //i尚未访问

Visit (G,i);visited[i]=1;

Queue[rear++]=i; //i入队

while (front!=rear) {

u=Queue[front++]; //队头元素出队

for (w=FirstAdjVex (G,u);w>=0;w=NextAdjVex (G,u,w)) {

if (visited[w]==0) { //w为u的尚未访问的邻接顶点

Visit (G,w);visited[w]=1;

Queue[rear++]=w;

}

}

}

}

} //end BFSTraverse

Status ShortestPath\_DIJ (MGraph G,int v0,int P[][MAX],int \*D) {

//Dijkstra算法求v0顶点到其他各顶点的最短途径及其长度

//P[v][w]为1，表示w是从v0到v的最短路径的当前顶点

//final[v]为1，表示v到v0的最短路径已求得，且v属于S集

int i=0,v=0,w=0,u=0,final[MAX],min=0;

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

final[v]=0;D[v]=G.arcs[v0][v];

for (w=0;w<G.vexnum;w++) P[v][w]=0; //设空路径

if (D[v]<INFINITY) {P[v][v0]=1;P[v][v]=1;}

}

D[v0]=0;final[v0]=1; //初始化,v0顶点包含于S集

for (i=1;i<G.vexnum;i++) { //循环G.vexnum-1次

min=INFINITY;

for (w=0;w<G.vexnum;w++) {

if (final[w]==0)

if (D[w]<min) {v=w;min=D[w];} //w距离v0更近

}

if (v==G.vexnum) return ERROR;

final[v]=1; //v加入S集

for (w=0;w<G.vexnum;w++) //更新当前最短路径及其长度

if ((final[w]==0)&&(G.arcs[v][w]!=INFINITY)&&(min+G.arcs[v][w]<D[w])) { //w属于V-S集，修改D[w]和P[w]

D[w]=min+G.arcs[v][w];

for (u=0;u<G.vexnum;u++) P[w][u]=P[v][u];

P[w][w]=1;

}

}

return OK;

} //end ShortestPath\_DIJ

Status PrintShortestPath (MGraph G,int v0,int P[][MAX],int \*D) {

//输出最短路径及其长度

int i=0,w=0,r=0,u=0,min;int wz[MAX];

printf ("源点%c到其他各顶点的最短路径及其长度为：\n",G.vexs[v0]);

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

if (i!=v0) {

if (D[i]==INFINITY) printf ("%c->%c: 无路径\n",G.vexs[v0],G.vexs[i]);

else {

printf ("%c->%c: %c",G.vexs[v0],G.vexs[i],G.vexs[v0]);

for (w=0;w<G.vexnum;w++) wz[w]=-1;

for (w=0;w<G.vexnum;w++) {

if (P[i][w]==1) wz[w]=w;

}

while (wz[v0]!=i) {

min=INFINITY;

for (w=0;w<G.vexnum;w++) {

if ((wz[w]==w)&&(G.arcs[wz[v0]][w]<min)) {

r=w;min=G.arcs[wz[v0]][r];

}

}

wz[r]=-1;wz[v0]=r;

printf ("->%c",G.vexs[wz[v0]]);

}

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

}

}

}

return OK;

} //end PrintShortestPath

int mininum (edge \*closedge,int v) {

//求辅助数组中代价为正且最小的下标

int i=0,mini=-1;

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

if (closedge[i].lowcost>0) {

if (mini==-1) mini=i;

else if (closedge[mini].lowcost>closedge[i].lowcost) mini=i;

}

}

return mini;

} //end mininum

Status MiniSpanTree\_PRIM (MGraph G,int v0) {

//用Prim算法构造无向连通图的最小生成树

if (G.kind==DG) {printf ("此图为有向图，不合条件\n");return ERROR;}

if (G.kind==DN) {printf ("此图为有向网，不合条件\n");return ERROR;}

if (G.kind==UDG) {printf ("此图为无向图，不合条件\n");return ERROR;}

int k=v0,i=0,j=0,min=0;

for (j=0;j<G.vexnum;j++) { //辅助数组初始化

if (j!=k) {

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

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

}

}

closedge[k].lowcost=0; //初始

printf ("最小生成树的各边及其权值如下：\n");

for (i=1;i<G.vexnum;i++) { //选择其他G.vexnum-1个顶点

k=mininum (closedge,G.vexnum);

if (k<0) {

printf ("此无向网非单连通网，可输入其他顶点，构建另一个连通子图的最小生成树\n");

break;

}

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

closedge[k].lowcost=0;

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

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

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

}

}

}

return OK;

} //end MiniSpanTree\_PRIM

/\*主函数部分\*/

void main () {

char menu=1,wz=0,lx=0,y='\0';int D[MAX],P[MAX][MAX],v0=0;

while (menu) {

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*菜单如下\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf (" 0——————————————结束操作\n");

printf (" 1——————————————建图\n");

printf (" 2——————————————深度优先搜索遍历\n");

printf (" 3——————————————广度优先搜索遍历\n");

printf (" 4——————————————源点的辐散最短路径及其长度\n");

printf (" 5——————————————无向连通网的最小生成树\n");

printf ("请选择菜单：");

scanf ("%d",&menu);

switch (menu) {

case 0:

break;

case 1:

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*开始建图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf ("请输入图的类型（0-DG、1-DN、2-UDG、3-UDN）：");

scanf ("%d",&lx);

printf ("请输入图的类型的存储位置（0-DG、1-DN、2-UDG、3-UDN）：");

scanf ("%d",&wz);

switch (lx) {

case 0:

G[wz].kind=DG;printf ("开始建有向图\n");CreateDG (G[wz]);break;

case 1:

G[wz].kind=DN;printf ("开始建有向网\n");CreateDN (G[wz]);break;

case 2:

G[wz].kind=UDG;printf ("开始建无向图\n");CreateUDG (G[wz]);break;

case 3:

G[wz].kind=UDN;printf ("开始建无向网\n");CreateUDN (G[wz]);break;

default:

printf ("图类型选择错误\n");break;

}

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*结束建图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 2:

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*开始深度优先搜索遍历\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf ("请输入要深度优先搜索遍历的图的存储位置（0-DG、1-DN、2-UDG、3-UDN）：");

scanf ("%d",&wz);

if (G[wz].vexnum!=0) {DFSTraverse (G[wz],PrintVex);printf ("\n");}

else printf ("图为空图\n");

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*结束深度优先搜索遍历\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 3:

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*开始广度优先搜索遍历\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf ("请输入要广度优先搜索遍历的图的存储位置（0-DG、1-DN、2-UDG、3-UDN）：");

scanf ("%d",&wz);

if (G[wz].vexnum!=0) {BFSTraverse (G[wz],PrintVex);printf ("\n");}

else printf ("图为空图\n");

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*结束广度优先搜索遍历\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 4:

printf ("\*\*\*\*\*\*\*\*\*\*\*\*开始求解源点的辐散最短路径及其长度\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf ("请输入图的存储位置（0-DG、1-DN、2-UDG、3-UDN）：");

scanf ("%d",&wz);

getchar ();

printf ("请输入源点值：");

scanf ("%c",&y);

v0=LocateVex (G[wz],y);

if (v0==G[wz].vexnum) {printf ("输入元素不在图中\n");break;}

ShortestPath\_DIJ (G[wz],v0,P,D);

PrintShortestPath (G[wz],v0,P,D);

printf ("\*\*\*\*\*\*\*\*\*\*\*\*结束求解源点的辐散最短路径及其长度\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

case 5:

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*开始构造无向连通网的最小生成树\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf ("请输入网的存储位置：");

scanf ("%d",&wz);

getchar ();

printf ("请输入树根元素：");

scanf ("%c",&y);

v0=LocateVex (G[wz],y);

if (v0==G[wz].vexnum) {printf ("输入元素不在网中\n");break;}

MiniSpanTree\_PRIM (G[wz],v0);

printf ("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*开始构造无向连通网的最小生成树\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

break;

default:

printf ("选择错误\n");

break;

}

printf ("\n");

}

printf ("操作完全结束\n");

} //end main