实验八 图实验

8.1 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) 熟练掌握图的两种遍历算法、遍历生成树及遍历算法的应用。

8.2 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．打印出图（网）的两种遍历序。

实验测试数据基本要求：

第一组数据： udg8.grp

第二组数据： udg115.grp

第三组数据： dg6.grp

第四组数据： f14.grp

2．求给定图中的边（或弧）的数目。 实验测试数据基本要求：第一组数据： udg8.grp

第二组数据： udg115.grp

第三组数据： dg6.grp

第四组数据： f14.grp

3．对给定的图G及出发点v0，设计算法从V0出发深度优先遍历图G，并构造出相应的生成树或生成森林。

实验测试数据基本要求：

第一组数据： udg8.grp

第二组数据： dg6.grp

第三组数据： un8.grp

第四组数据： dn10.grp

4．对给定的图G及出发点v0，设计算法从V0出发广度优先遍历图G，并构造出相应的生成树或生成森林。

第一组数据： udg8.grp

第二组数据： dg6.grp

第三组数据： un8.grp

第四组数据： dn10.grp

5．实现Prim算法，求解下列给定图G的最小生成树。 实验测试数据基本要求：第一组数据： udn6.grp

第二组数据： un8.grp

6．实现Kruskal算法，求解下列给定图G的最小生成树。 实验测试数据基本要求：第一组数据： udn6.grp

第二组数据： un8.grp

7．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

8．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

9．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

10．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： dn8.grp

第二组数据： dn10.grp

8.3 实验说明

这里介绍一种从文本文件读入图的数据创建图的方法，这样我们可以按照指定的格式，先从容地准备好数据，然后由程序自动读入数据来创建图。

createGrpAdjMatrix.h，文件创建邻接矩阵表示的图。

createGrpAdjLinkedList.h，文件创建邻接表表示的图。

**（以下给出的图的创建方法仅供参考，实验者可自行设计其它创建方法）**

**1. 数据文件格式设计**

这里数据用文本文件保存，文件扩展名可自行指定，比如g8.grp，只要数据按文本文件格式读写即可。下面给出一种数据文件格式，其实读者可以自行设计图的数据文件格式。

① 标识行1：Graph

标识这是一个图的数据文件，这一行也可以不要。

② 标识行2：UDG、或UDN、或DG、或DN

这一行用来标识此图是无向图（UDG）、无向网（UDN）、有向图（DG）、还是有向网（DN）。

③ 顶点行

这一行将图中所有顶点列出，顶点之间用空格进行分割。这些顶点数据读出后存放到图的顶点数组中。

例如，图6-21(a)所示的图的顶点行数据为：a b c d。

图的各种算法都是用顶点的编号来引用顶点的，所以这一行顶点的排列顺序是很重要的，顶点的排列顺序决定了顶点的编号。比如上例中，顶点a、b、c、d对应的编号就为1、2、3、4。

④ 边数据行

一条边一行，边的2个顶点之间用空格分割。如果是网，每一行再加边的权值，也以空格分割。如果是无向图和无向网，每条边会重复一次。

例如图6-18(a)无向图的边的数据为：

a b

a c

a d

b a

b c

c a

c b

c d

d a

d c

图6-21(a)无向网边的数据为：

a b 4

a c 5

a d 3

b a 4

b c 2

c a 5

c b 2

c d 6

d a 3

d c 6

⑤ 其它行

如果程序强大一点，还可以在文件中加注释行，允许出现空行等，当然这是非必须的。

举一个完整的图的数据文件的例子，对图6-18(a)的无向图，完整的数据文件如下：

//文件可以加注释行，注释以“//”开始

//Graph为图标志，否则判定格式不对

//标志行后，第一行为图的类型。UDG--无向图；UDN--无向网；DG--有向图；DN--有向网

//标志行后，第二行为顶点元素

//顶点行以下图的边或弧。用顶点表示，第一列为起始顶点；第二列为邻接点；在网中再增加一列表示权值。

//本图具有4个顶点5条边

//下一行为图的标识行

Graph

//图的类型标识，此为无向图

UDG

//顶点元素数据

a b c d

//以下为边的数据，共10行数据，表示5条边

a b

a c

a d

b a

b c

c a

c b

c d

d a

d c

文件名不妨叫做Gudg4.grp。

再举一个有向网的例子，对图6-22所示的有向网，完整的数据文件如下：

图6-22 一个有向网实例

a

c

g

h

b

7

8

3

1

2

e

f

20

2

1

2

1

6

15

4

3

i

j

d

2

1



//标识为图数据

Graph

//标识有向网

DN

//顶点数据

a b c d e f g h i j

//以下为边数据，共15条边

a b 2

a d 20

b e 1

c a 3

d c 8

d f 6

d g 4

e c 7

e h 3

f c 1

g h 1

h f 2

h j 2

i g 2

j i 1

不妨设文件名为Gdn10.grp

**2. 从数据文件创建邻接矩阵表示的图**

指定图的数据文件名，然后逐行读出数据并处理，自动创建邻接矩阵表示的图。本程序可以自动处理注释行和空行，程序实现如下：

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*文件创建图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* 函数功能：从文本文件创建邻接矩阵表示的图 \*//

//\* 入口参数 char fileName[]，文件名 \*//

//\* 出口参数：Graph &G，即创建的图 \*//

//\* 返 回 值：bool，true创建成功；false创建失败 \*//

//\* 函 数 名：CreateGraphFromFile(char fileName[], Graph &G) \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int CreateGraphFromFile(char fileName[], Graph &G)

{

FILE\* pFile; //定义文件指针

char str[1000]; //存放读出一行文本的字符串

char strTemp[10]; //判断是否注释行

cellType eWeight; //边的信息，常为边的权值

GraphKind graphType; //图类型枚举变量

pFile=fopen(fileName,"r");

if(!pFile)

{

printf("错误：文件%s打开失败。\n",fileName);

return false;

}

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义的函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //跳过注释行

continue;

else //非注释行、非空行，跳出循环

break;

}

//循环结束，str中应该已经是图的标识Graph，判断标识是否正确

if(strstr(str,"Graph")==NULL)

{

printf("错误：打开的文件格式错误！\n");

fclose(pFile); //关闭文件

return false;

}

//读取图的类型，跳过空行

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //注释行，跳过，继续读取下一行

continue;

else //非空行，也非注释行，即图的类型标识

break;

}

//设置图的类型

if(strstr(str,"UDG"))

graphType=UDG; //无向图

else if(strstr(str,"UDN"))

graphType=UDN; //无向网

else if(strstr(str,"DG"))

graphType=DG; //有向图

else if(strstr(str,"DN"))

graphType=DN; //有向网

else

{

printf("错误：读取图的类型标记失败！\n");

fclose(pFile); //关闭文件

return false;

}

//读取顶点行数据到str。跳过空行

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //注释行，跳过，继续读取下一行

continue;

else //非空行，也非注释行，即图的顶点元素行

break;

}

//顶点数据放入图的顶点数组

char\* token=strtok(str," ");

int nNum=0;

while(token!=NULL)

{

G.Data[nNum]=\*token;

token = strtok( NULL, " ");

nNum++;

}

//图的邻接矩阵初始化

int nRow=0; //矩阵行下标

int nCol=0; //矩阵列下标

if(graphType==UDG || graphType==DG)

{

for(nRow=0;nRow<nNum;nRow++)

for(nCol=0;nCol<nNum;nCol++)

G.AdjMatrix[nRow][nCol]=0;

}

else

{

for(nRow=0;nRow<nNum;nRow++)

for(nCol=0;nCol<nNum;nCol++)

G.AdjMatrix[nRow][nCol]=INF; //INF表示无穷大

}

//循环读取边的数据到邻接矩阵

int edgeNum=0; //边的数量

elementType Nf, Ns; //边或弧的2个相邻顶点

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //注释行，跳过，继续读取下一行

continue;

char\* token=strtok(str," "); //以空格为分隔符，分割一行数据，写入邻接矩阵

if(token==NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

Nf=\*token; //获取边的第一个顶点

token = strtok( NULL, " "); //读取下一个子串，即第二个顶点

if(token==NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

Ns=\*token; //获取边的第二个顶点

//从第一个顶点获取行号

for(nRow=0;nRow<nNum;nRow++)

{

if(G.Data[nRow]==Nf) //从顶点列表找到第一个顶点的编号

break;

}

//从第二个顶点获取列号

for(nCol=0;nCol<nNum;nCol++)

{

if(G.Data[nCol]==Ns) //从顶点列表找到第二个顶点的编号

break;

}

//如果为网，读取权值

if(graphType==UDN || graphType==DN)

{ //读取下一个子串，即边的附加信息，常为边的权重

token = strtok( NULL, " ");

if(token==NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

eWeight=atoi(token); //取得边的附加信息

}

if(graphType==UDN || graphType==DN)

G.AdjMatrix[nRow][nCol]=eWeight;

//如果为网，邻接矩阵中对应的边设置权值，否则置为1

else

G.AdjMatrix[nRow][nCol]=1;

edgeNum++; //边数加1

}

G.VerNum=nNum; //图的顶点数

if(graphType==UDG || graphType==UDN)

G.ArcNum=edgeNum / 2; //无向图或网的边数等于统计的数字除2

else

G.ArcNum=edgeNum;

G.gKind=graphType; //图的类型

fclose(pFile); //关闭文件

return true;

}

**3. 从数据文件创建邻接表表示的图**

程序实现如下：

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*文件创建图\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

//\* 函数功能：从文本文件创建邻接表表示的图 \*//

//\* 入口参数 char fileName[]，文件名 \*//

//\* 出口参数：Graph &G，即创建的图 \*//

//\* 返 回 值：bool，true创建成功；false创建失败 \*//

//\* 函 数 名：CreateGraphFromFile(char fileName[], Graph &G) \*//

//\* 备注：本函数使用的数据文件格式以边（顶点对）为基本数据 \*//

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int CreateGraphFromFile(char fileName[], Graph &G)

{

FILE\* pFile; //定义文件指针

char str[1000]; //存放读出一行文本的字符串

char strTemp[10]; //判断是否注释行

char\* ss;

int i=0, j=0;

int edgeNum=0; //边的数量

eInfoType eWeight; //边的信息，常为边的权值

GraphKind graphType; //图类型枚举变量

pFile=fopen(fileName,"r");

if(!pFile)

{

printf("错误：文件%s打开失败。\n",fileName);

return false;

}

while(fgets(str,1000,pFile)!=NULL) //跳过空行和注释行

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //跳过注释行

continue;

else //非注释行、非空行，跳出循环

break;

}

//循环结束，str中应该已经是图的标识Graph，判断标识是否正确

if(strstr(str,"Graph")==NULL)

{

printf("错误：打开的文件格式错误！\n");

fclose(pFile); //关闭文件

return false;

}

//读取图的类型，跳过空行及注释行

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //注释行，跳过，继续读取下一行

continue;

else //非空行，也非注释行，即图的类型标识

break;

}

//设置图的类型

if(strstr(str,"UDG"))

graphType=UDG; //无向图

else if(strstr(str,"UDN"))

graphType=UDN; //无向网

else if(strstr(str,"DG"))

graphType=DG; //有向图

else if(strstr(str,"DN"))

graphType=DN; //有向网

else

{

printf("错误：读取图的类型标记失败！\n");

fclose(pFile); //关闭文件

return false;

}

//读取顶点数据到str。跳过空行

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //注释行，跳过，继续读取下一行

continue;

else //非空行，也非注释行，即图的顶点元素行

break;

}

//顶点数据放入图的顶点数组

char\* token=strtok(str," ");

int nNum=0;

while(token!=NULL)

{

G.VerList[nNum].data=\*token;

G.VerList[nNum].firstEdge=NULL;

token = strtok( NULL, " ");

nNum++;

}

//循环读取边（顶点对）数据

int nRow=0; //矩阵行下标

int nCol=0; //矩阵列下标

EdgeNode\* eR; //边链表尾指针

EdgeNode\* p;

elementType Nf, Ns; //边或弧的2个相邻顶点

while(fgets(str,1000,pFile)!=NULL)

{

strLTrim(str); //删除字符串左边空格，这是一个自定义函数

if (str[0]=='\n') //空行，继续读取下一行

continue;

strncpy(strTemp,str,2);

if(strstr(strTemp,"//")!=NULL) //注释行，跳过，继续读取下一行

continue;

char\* token=strtok(str," "); //以空格为分隔符，分割一行数据

if(token==NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

Nf=\*token; //获取边的第一个顶点

token = strtok( NULL, " "); //读取下一个子串，即第二个顶点

if(token==NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

Ns=\*token; //获取边的第二个顶点

//从第一个顶点获取行号

for(nRow=0;nRow<nNum;nRow++)

{

if(G.VerList[nRow].data==Nf) //从顶点列表找到第一个顶点的编号

break;

}

//从第二个顶点获取列号

for(nCol=0;nCol<nNum;nCol++)

{

if(G.VerList[nCol].data==Ns) //从顶点列表找到第二个顶点的编号

break;

}

//如果为网，读取权值

if(graphType==UDN || graphType==DN)

{ //读取下一个子串，即边的附加信息，常为边的权重

token = strtok( NULL, " ");

if(token==NULL) //分割为空串，失败退出

{

printf("错误：读取图的边数据失败！\n");

fclose(pFile); //关闭文件

return false;

}

eWeight=atoi(token); //取得边的附加信息，即权值

}

eR=G.VerList[nRow].firstEdge;

while(eR!=NULL && eR->next!=NULL)

{

eR=eR->next; //后移边链表指针，直至尾节点

}

p=new EdgeNode; //申请一个边链表结点

p->adjVer=nCol+1; //顶点的编号，从1开始

//边的附加信息（权值），对有权图保存权值，无权图为1

if(graphType==UDN || graphType==DN)

p->eInfo=eWeight;

else

p->eInfo=1;

p->next=NULL;

if(G.VerList[nRow].firstEdge==NULL)

{

G.VerList[nRow].firstEdge=p;

eR=p;

}

else

{

eR->next=p;

eR=p; //新的尾指针

}

edgeNum++; //边数加1

}

G.VerNum=nNum; //图的顶点数

if(graphType==UDG || graphType==UDN)

G.ArcNum=edgeNum / 2; //无向图或网的边数等于统计的数字除2

else

G.ArcNum=edgeNum;

G.gKind=graphType; //图的类型

fclose(pFile); //关闭文件

return true;

}

**4. 图的销毁**

以邻接矩阵为存储结构的图，因为使用矩阵存储图的数据，不存在销毁（释放内存）问题。但是以邻接表为存储结构的图，由于在创建图的过程中使用malloc()函数或new操作符动态申请了内存，当这个图不再需要时，必须手工释放动态申请的内存，否则造成内存泄漏。下面给出一个销毁邻接表表示的图的程序。

void DestroyGraph(Graph &G)

{

EdgeNode \*p,\*u; //边链表结点指针

int vID;

for(vID=1; vID<=G.VerNum; vID++) //循环删除每个顶点的边链表

{

p=G.VerList[vID-1].firstEdge;

G.VerList[vID-1].firstEdge=NULL;

while(p) //循环删除当前顶点所有的关联边

{

u=p->next; //u指向下一个边结点

delete(p); //删除当前边结点

p=u;

}

}

p=NULL;

u=NULL;

G.VerNum=-1; //标识图已经销毁

}