

## 实验 5（图的物理实现）日志

### 一、图的概述

**定义：**图（Graph）是由顶点的有穷非空集合和顶点之间边的集合组成，通常表示为： $G(V, E)$ ，其中， $G$  表示一个图， $V$  是图  $G$  中顶点的集合， $E$  是图  $G$  中边的集合。

图的存储结构分为邻接矩阵和邻接表。

### 二、邻接矩阵存储

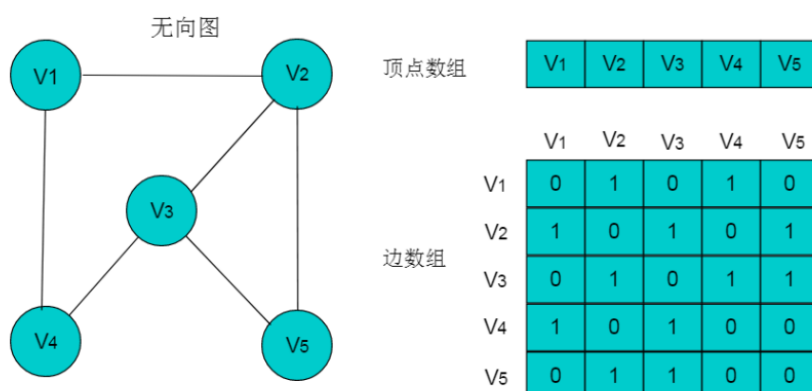
**图的数组存储**方式也称为**邻接矩阵存储**。图中的数据信息包括：顶点信息和描述顶点之间关系的边的信息，将这两种信息存储在数组中即为图的数组存储。

首先，创建顶点数组，顶点数组中存储的是图的顶点信息，采用一维数组的方式即可存储所有的顶点信息。存储图中边的信息时，由于边是描述顶点与顶点之间关系的信息，因此需要采用二维数组进行存储。

**定义：**设图  $G$  有  $n$  个顶点，则邻接矩阵是一个  $n \times n$  的方阵  $A$ ，定义为：

$$A[i][j] = \begin{cases} w_{i,j} & \text{若 } \langle v_i, v_j \rangle \text{ 或 } (v_i, v_j) \in VR \\ 0 & \text{否则} \end{cases}$$

其中，或者  $(v_i, v_j)$  表示顶点  $v_i$  与顶点  $v_j$  邻接。 $w_{i,j}$  表示边的权重值。例如：下图所示的无向图，采用数组存储形式如下。



**无向图的数组存储主要有以下特性：**

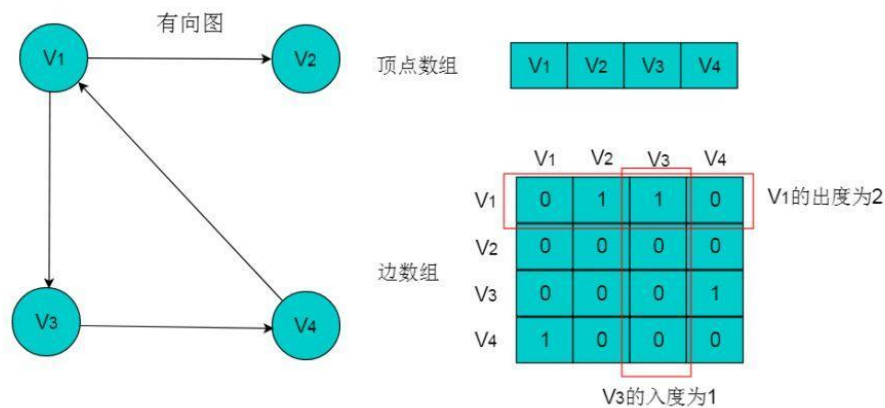
- （1）顶点数组长度为图的顶点数目  $n$ 。边数组为  $n \times n$  的二维数组。
- （2）边数组中， $A[i][j] = 1$  代表顶点  $i$  与顶点  $j$  邻接， $A[i][j] = 0$  代表顶点  $i$  与顶点  $j$  不邻接。
- （3）在无向图中。由于边是无向边，因此顶点的邻接关系是对称的，边数组为对称二维数组。

(4) 顶点与自身之间并未邻接关系，因此边数组的对角线上的元素均为 0。

(5) 顶点的度即为顶点所在的行或者列 1 的数目。例如：顶点 V2 的度为 3，则 V2 所在行和列中的 1 的数目为 3。

当图为有向图时，图的数组存储方式要发生变化。

例如：图 5.2 所示的有向图，采用数组存储形式如下。



### 有向图的数组存储主要有以下特性：

- (1) 顶点数组长度为图的顶点数目  $n$ 。边数组为  $n \times n$  的二维数组。
- (2) 边数组中，数组元素为 1，即  $A[i][j] = 1$ ，代表第  $i$  个顶点与第  $j$  个顶点邻接，且  $i$  为尾， $j$  为头。 $A[i][j] = 0$  代表顶点与顶点不邻接。
- (3) 在有向图中，由于边存在方向性，因此数组不一定为对称数组。
- (4) 对角线上元素为 0。
- (5) 第  $i$  行中，1 的数目代表第  $i$  个顶点的出度。例如：顶点 V1 的出度为 2，则顶点 V1 所在行的 1 的数目为 2。
- (6) 第  $j$  列中，1 的数目代表第  $j$  个顶点的入度。例如：V3 的入度为 1，则 V3 所在列中 1 的数目为 1。

### 数组存储方式优点：

数组存储方式容易实现图的操作。例如：求某顶点的度、判断顶点之间是否有边（弧）、找顶点的邻接点等等。

### 数组存储方式缺点：

采用数组存储方式，图若有  $n$  个顶点则需要  $n^2$  个单元存储边(弧)，空间存储效率为  $O(n^2)$ 。当顶点数目较多，边数目较少时，此时图为稀疏图，这时尤其浪费空间。

例如：图 5.3 所示的图，图中有 9 个顶点，边数为 10，需要  $9 \times 9$  的二维数组，而实际存储边信息空间只有 10，造成空间浪费。

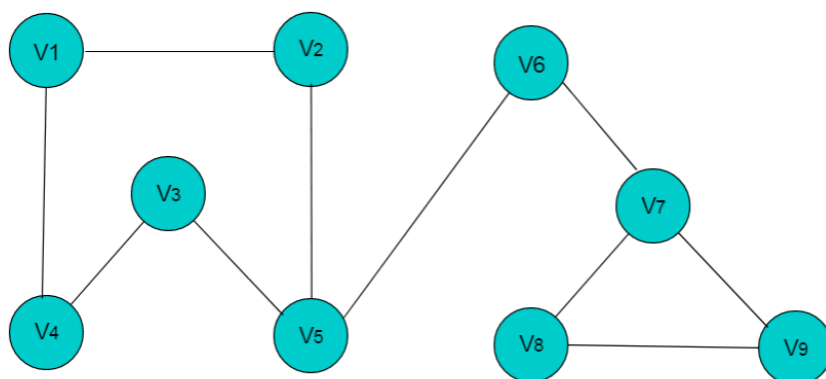


图 5.3 所示无向图的存储数组：

顶点数组		V1	V2	V3	V4	V5	V6	V7	V8	V9
边数组	V1	0	1	0	1	0	0	0	0	0
	V2	1	0	1	0	1	0	0	0	0
	V3	0	1	0	1	1	0	0	0	0
	V4	1	0	1	0	0	0	0	0	0
	V5	0	1	1	0	0	1	0	0	0
	V6	0	0	0	0	1	0	1	0	0
	V7	0	0	0	0	0	1	0	1	1
	V8	0	0	0	0	0	0	1	0	1
	V9	0	0	0	0	0	0	1	1	0

### 三、邻接表存储

当使用数组存储时，主要有以下三个问题：

（1）对于一个图，若图中的顶点数目过大，则无法使用邻接矩阵进行存储。因为在分配数组内存时可能会导致内存分配失败。

（2）对于某些稀疏图（即顶点数目多，边数目少），创建的数组大小很大，而真正存储的有用信息又很少，这就造成了空间上的浪费。

（3）有时两个点之间不止存在有一条边，这时用邻接矩阵就无法同时表示两条以上的边。

针对以上情况，提出了一种特殊的图存储方式，让每个节点拥有的数组大小刚好就等于它所连接的边数，由此建立一种邻接表的存储方式。

**邻接表存储**方法是一种数组存储和链式存储相结合的存储方法。在邻接表中，对图中的每个顶点建立一个单链表，第  $i$  个单链表中的结点依附于顶点  $V_i$  的边（对有向图是以顶点  $V_i$  为尾的弧）。链表中的节点称为表节点，共有 3 个域，具体结构见下图：

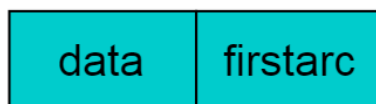
## 表节点



**表结点**由三个域组成，**adjvex** 存储与  $V_i$  邻接的点在图中的位置，**nextarc** 存储下一条边或弧的结点，**info** 存储与边或弧相关的信息如权值。

除表节点外，需要在数组中存储**头节点**，**头结点**由两个域组成，分别指向链表中第一个顶点和存储  $V_i$  的名或其他信息。具体结构如下图：

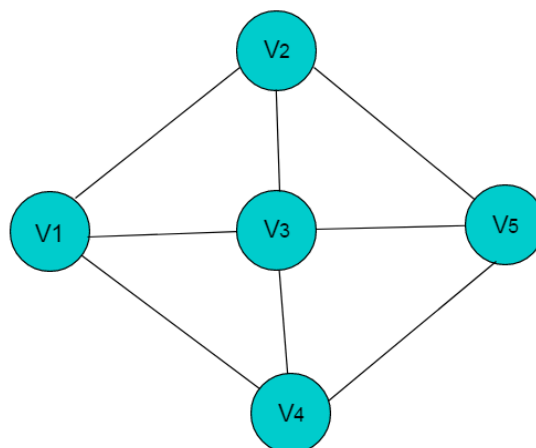
## 头节点



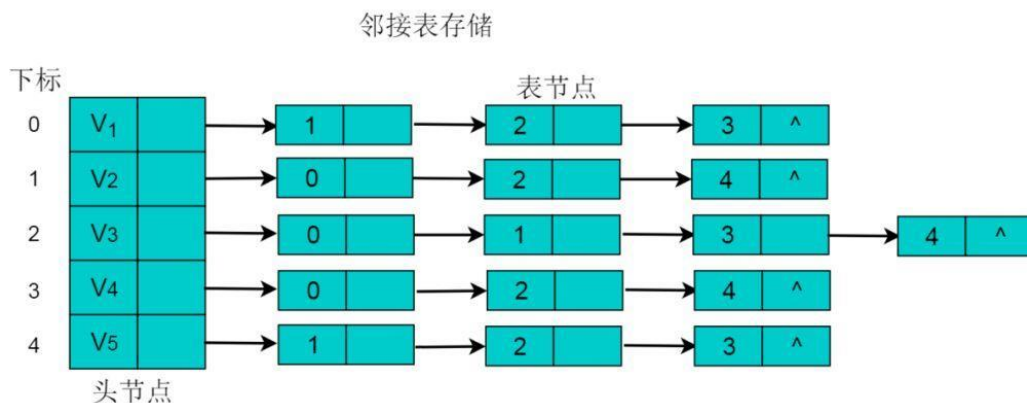
其中，**data** 域中存储顶点相关信息，**firstarc** 指向链表的第一个节点。

### 无向图采用邻接表方式存储

例如：图 6.1 所示的无向图采用邻接表存储。



采用邻接表方式存储图 6.1 中的无向图，绘图过程中忽略边节点的 **info** 信息，头结点中的 **data** 域存储顶点名称。以  $V_1$  顶点为例， $V_1$  顶点的邻接顶点为  $V_2$ 、 $V_3$ 、 $V_4$ ，则可以创建 3 个表节点，表节点中 **adjvex** 分别存储  $V_2$ 、 $V_3$ 、 $V_4$  的索引 1、2、3，按照此方式，得到的邻接表为：



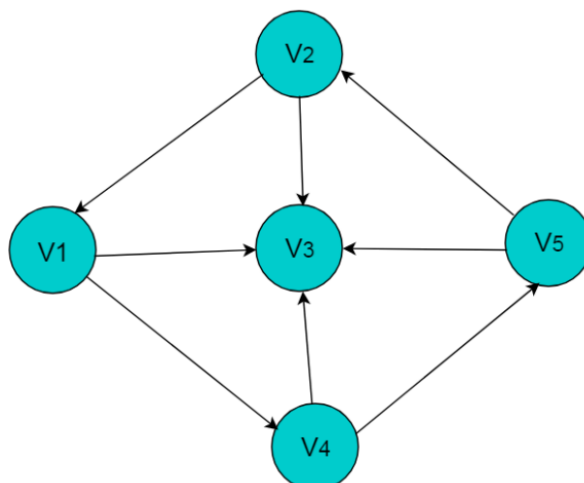
### 无向图的邻接表存储特性：

(1) 数组中头节点的数目为图的顶点数目。

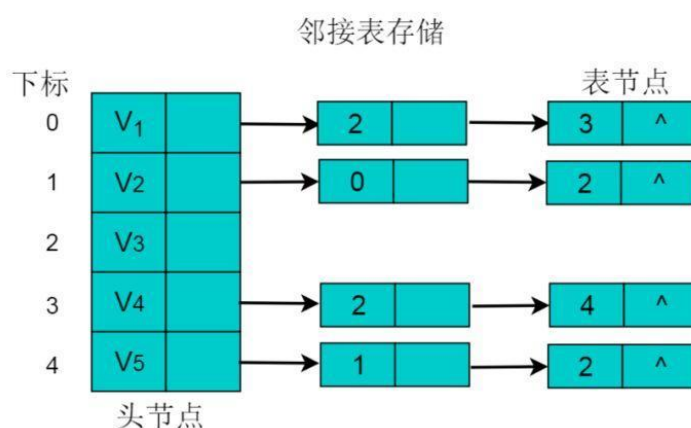
(2) 链表的长度即为顶点的度。例如：V<sub>1</sub> 顶点的度为 3，则以 V<sub>1</sub> 为头节点的链表中表节点的数目为 3。

### 有向图采用邻接表方式存储

例如：图 6.3 所示的有向图采用邻接表存储。



采用邻接表方式存储图 6.3 中的有向图，绘图过程中忽略边节点的 info 信息，头结点中的 data 域存储顶点名称。以 V<sub>1</sub> 顶点为例，V<sub>1</sub> 顶点的邻接顶点为 V<sub>2</sub>、V<sub>3</sub>、V<sub>4</sub>，但是以 V<sub>1</sub> 顶点为尾的边只有两条，即 V<sub>3</sub> 和 V<sub>4</sub>。因此，创建 2 个表节点。表节点中 adjvex 分别存储 V<sub>3</sub>、V<sub>4</sub> 的索引 2、3，按照此方式，得到的邻接表为：



### 有向图的邻接表存储特性：

- (1) 数组中头节点的数目为图的顶点数目。
- (2) 链表的长度即为顶点的出度。例如 V<sub>1</sub> 的出度为 2，V<sub>1</sub> 为头节点的链表中，表节点的数目为 2。
- (3) 顶点 V<sub>i</sub> 的入度为邻接表中所有 `adjvex` 值域为 i 的表结点数。例如：顶点 V<sub>3</sub> 的入度为 4，则链表中所有 `adjvex` 值域为 2 的表结点数目为 4。

**注：**图采用邻接表的方式表示时，其表示方式是不唯一的。这是因为在每个顶点对应的单链表中，各边节点的链接次序可以是任意的，取决于建立邻接表的算法以及边的输入次序。

## 四、实验进程

### 12月2日：

1. 回顾、理解图的基础知识。
2. 结合课本和教学 PPT 学习图的抽象类 ADT。
3. 利用网络学习了图的两种存储结构。
4. 利用学习通上的代码压缩包，完成 `book.h`, `graph.h` 和 `graphm.h` 三个文件。

### 12月3日：

1. 结合 CG 系统的题目要求，完成 `graphm.cpp` 文件。
2. 上交 CG 系统检测代码的正确性，结果错误。
3. 分析错误原因，修改代码，重新上传，结果正确。

### 12月4日：

1. 利用学习通上的代码压缩包，完成 `graph1.h` 文件。

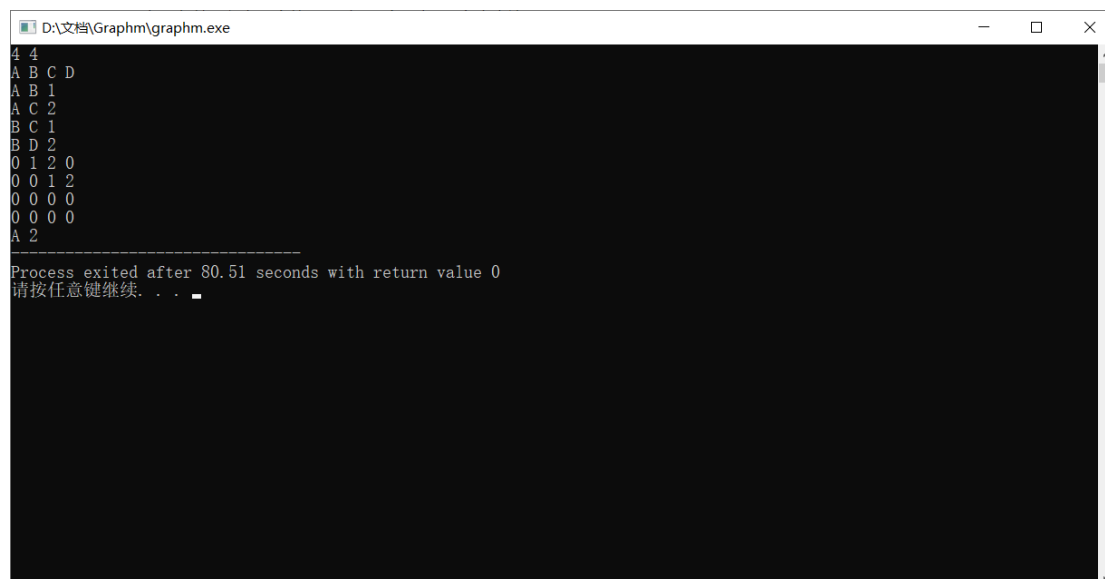
2. 结合之前的实验编写的 link.h,list.h,lhist.h 文件，完成 graphl.cpp 文件。
3. 上交 CG 系统检测代码的正确性，结果正确。

## 12月5日：

1. 通过 CSDN 进一步学习图的相关知识。
2. 修改并完善实验日志，提交到学习通平台。
3. 进一步学习图的十字链表存储结构。

## 五、遇到的问题及解决方法

错误运行结果：



```
D:\文档\Graphm\graphm.exe
4 4
A B C D
A B 1
A C 2
A D 1
B C 1
B D 2
C D 2
D A 1
D B 1
D C 1
D D 1
A 2
B 1
C 2
D 1
Process exited after 80.51 seconds with return value 0
请按任意键继续...
```

错误代码：

```
int maxv=n-1,count=0,maxnum=0;
for(int i=0;i<n;i++)
{
    for(int j=0;j<n;j++)
    {
        cout<<G->weight(i,j)<<' ';
        if(G->isEdge(i,j))
            count++;
    }
    cout<<endl;
    if(count>maxnum)
    {
        maxnum=count;
        maxv=i;
    }
    count=0;
}
cout<<G->getVex(maxv)<<' '<<maxnum;
```

错误原因：

题目要求如果两个点的出度相同，则依据点的字符大小，输出字符大的点，原来的代码中并未考虑到两个点出度相同这一点。

错误改正：

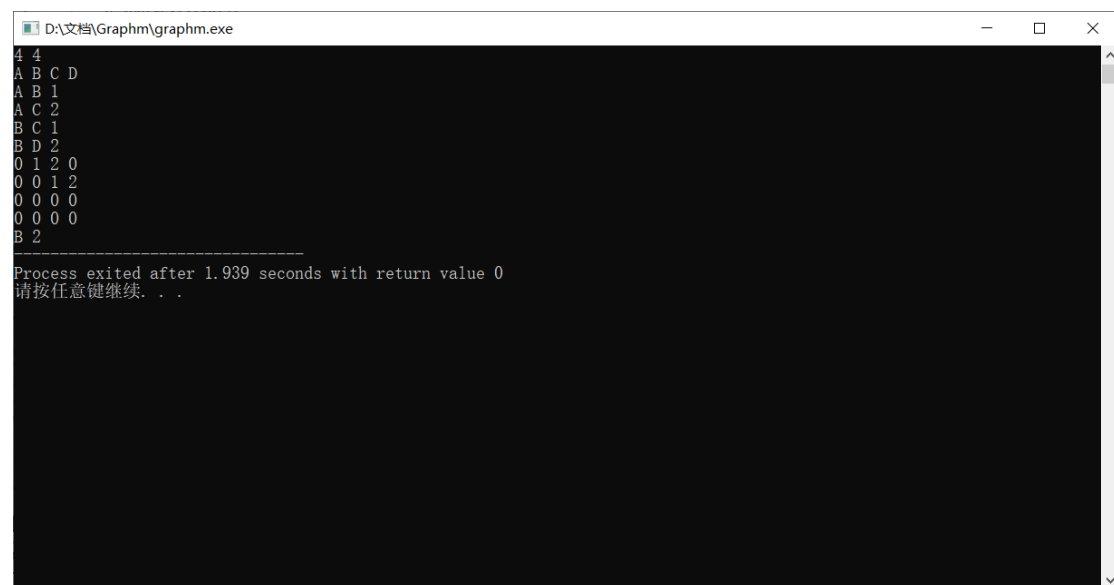
加一个等号，当两个点出度相同时，也进行更新。

正确代码：

```
if(count>=maxnum)
{
    maxnum=count;
    maxv=i;
}
```

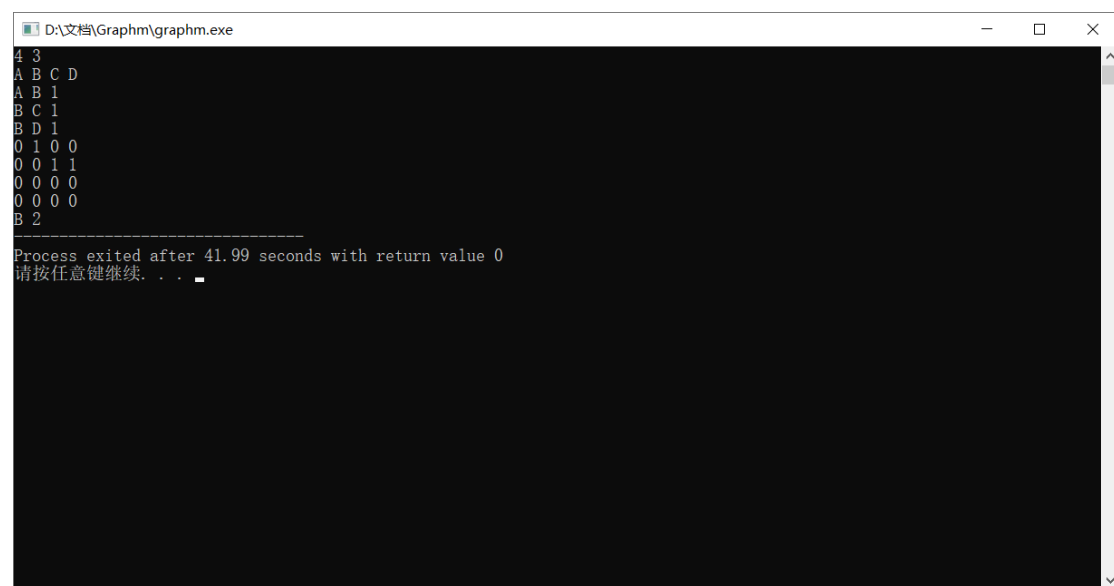


正确运行结果：



```
D:\文档\Graphm\graphm.exe
4 4
A B C D
A B 1
A C 2
B C 1
B D 2
0 1 2 0
0 0 1 2
0 0 0 0
0 0 0 0
B 2
-----
Process exited after 1.939 seconds with return value 0
请按任意键继续...
```

## 六、测试用例



```
D:\文档\Graphm\graphm.exe
4 3
A B C D
A B 1
B C 1
B D 1
0 1 0 0
0 0 1 1
0 0 0 0
0 0 0 0
B 2
-----
Process exited after 41.99 seconds with return value 0
请按任意键继续...
```

实验五测试用例

```
D:\文档\Graphm\graphm.exe
5 4
A B C D E
A B 3
A C 5
C D 2
D E 1
0 3 5 0 0
0 0 0 0 0
0 0 0 2 0
0 0 0 0 1
0 0 0 0 0
A 2
-----
Process exited after 20.83 seconds with return value 0
请按任意键继续. . .
```

实验五测试用例

```
D:\文档\Graphm\graphm.exe
4 5
A B C D
A B 1
A C 2
B A 3
B C 1
C D 4
0 1 2 0
3 0 1 0
0 0 0 4
0 0 0 0
B 2
-----
Process exited after 5.638 seconds with return value 0
请按任意键继续. . .
```

实验五测试用例

```
D:\文档\Graphm\graphm.exe
4 3
A B C D
A B -1
Assertion Failed: Illegal weight value

-----
Process exited after 12.12 seconds with return value 4294967295
请按任意键继续. . .
```

实验五测试用例

```
D:\文档\Graphm\manager.exe
顶点数: 5
边 数: 7
图类型: 无向图
顶点信息:
A B C D E
边信息:
A <--> B:10
A <--> C:3
A <--> D:20
B <--> C:2
B <--> D:5
C <--> E:15
D <--> E:11
邻接矩阵为:
0 10 3 20 0
10 0 2 5 0
3 2 0 0 15
20 5 0 0 11
0 0 15 11 0

-----
Process exited after 0.05472 seconds with return value 0
请按任意键继续. . .
```

用邻接矩阵实现图的基本操作

```
D:\文档\Graph\manager.exe
顶点数: 4
边数: 4
图类型: 有向图
顶点信息:
shanghai B C D
边信息:
shanghai --> B:6
shanghai --> D:5
B --> C:7
C --> D:1
邻接表为:
shanghai --> B --> D
B --> C
C --> D
D

-----
Process exited after 0.03406 seconds with return value 0
请按任意键继续. . .
```

用邻接表实现图的基本操作

## 七、总结与心得

邻接矩阵和邻接表都属于图的存储结构，各有各的优缺点。

经过这次实验，我对于图的存储结构的相关代码已基本熟悉，算法知识得到了复习与巩固。在写代码的过程与调试中，在解决问题过程中，丰富了个人编程的经历和经验，提高了个人解决问题的能力。

数据结构中的图的实验很多知识都是以前 C++ 语言学过的，只是将这些知识进行了一些结构化，使程序的结构看起来更加合理，方便代码管理。另外，在使用链表时指针的使用是最值得我们注意的地方，它需要我们在逻辑顺序上不能出错，否则就会造成错误，很多时候编译器识别不到，只有在运行时才会出现内存错误的提示，调试起来也不是那么明显。所以指针是我应该重点掌握的地方，只有学好了指针，才可能学好 C++。总之今后在实验过程中要更加注重这些基本的操作。

通过本次基础实验，我对图的概念有了一个新的认识，在学习离散数学的时候，总觉得图是很抽象的东西，但是在学习了《数据结构》这门课程之后，我慢慢地体会到了其中的奥妙，图能够在计算机中存在，首先要捕捉他有哪些具体化、数字化的信息，比如说权值、顶点个数等，这也就说明了想要把生活中的信息转化到计算机中必须用数字来完整的构成一个信息库，而图的存在，又涉及到了顶点之间的联系。

图分为有向图和无向图，而无向图又是有向图在权值双向相等下的一种特例，如何能在计算机中表示一个双向权值不同的图，这就是一件很巧妙的事情，经过了思考和老师同学的帮助，我用 `edges[i][j]=a` 和 `edges[j][i]=b` 就能实现了一个双向图信息的存储。由于这段时间事情比较多，故在实验上未花很多的精力，用了三天抽空把图的两种实现方法做了，有些不尽人意的地方也没有加以修正，但是实验的过程之中我又对图的知识重新温习了一遍，获益匪浅。我希望能够做更多这样的实验，这样我才能发现自己在实际操作中的不足并加以改正。

## 八、实验说明：

开发语言：C++

开发平台：Dev-C++

2020.12.5

杨杰