第7章 图

提纲

CONTENTS

7.1 图的基本概念

7.2 图的存储结构

7.3 图的遍历

7.4 生成树和最小生成树

7.5 最短路径

7.6 拓扑排序

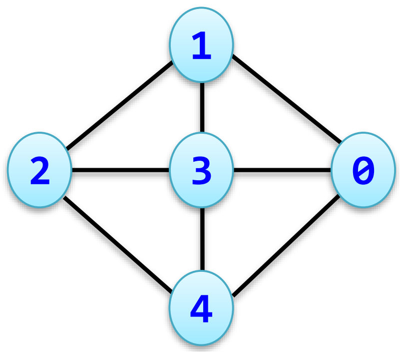
7.7 AOE网与关键路径

7.1 图的基本概念

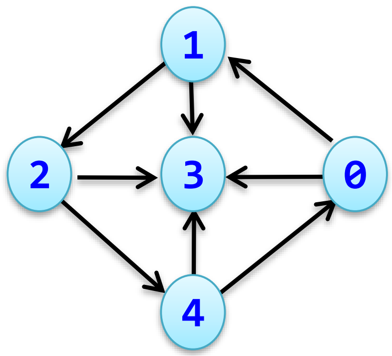
7.1.1 图的定义

- 图G（Graph）由两个集合V（Vertex）和E（Edge）组成，记为G=(V，E)。
- V是顶点的有限集合，记为V(G)。
- E是连接V中两个不同顶点（顶点对）的边的有限集合，记为E(G)。

无向图和有向图

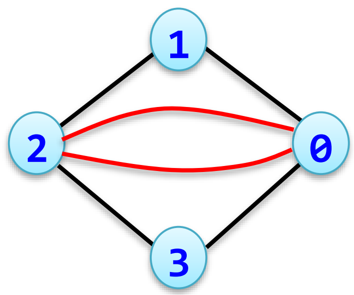


（a）一个无向图

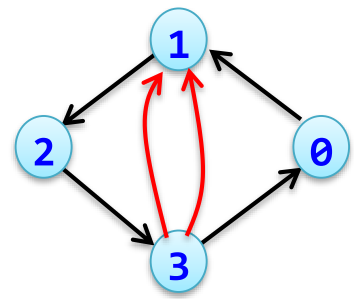


（b）一个有向图

- 在无向图G中，顶点i与顶点j的一条无向边用无序偶(i,j)或(j,i)表示。
- 在有向图G中，从顶点i到顶点j的一条有向边用序偶$<i,j>$ 表示。



（a）多重无向图

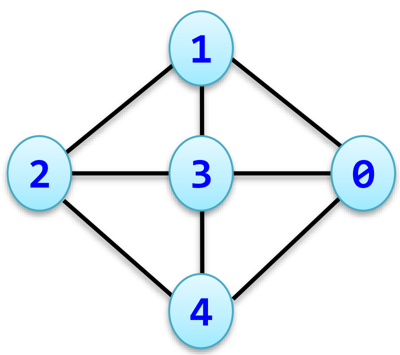


（b）多重有向图

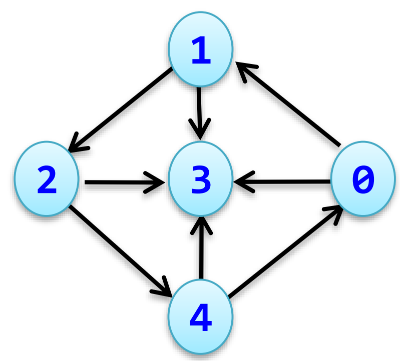
如果图中允许重复边出现，则称图为多重图，本书中讨论的图均指非多重图。

7.1.2 图的基本术语

- 在无向图中，若存在一条边(i,j)，则称边的端点i和j互为邻接点。
- 在有向图中，若存在一条边<i，j>，则称i为起点，j为终点。并称j是i的出边邻接点，i是j的入边邻接点。

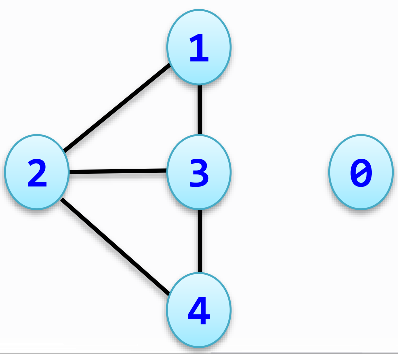


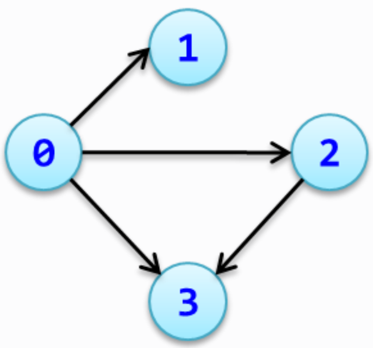
（a）一个无向图



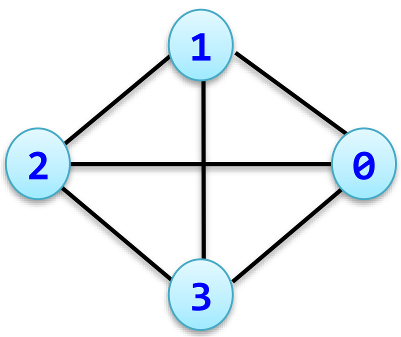
（b）一个有向图

- 在无向图中，顶点所关联的边的数目称为该顶点的度。
- 在有向图中，以顶点i为终点的边的数目，称为该顶点的入度。以顶点i为起点的边的数目，称为该顶点的出度。一个顶点的入度与出度的和为该顶点的度。

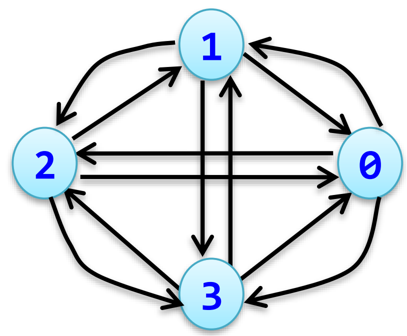




- 完全无向图中的每两个顶点之间都存在着一条边。含有n个顶点的完全无向图有$n(n-1)/2$条边。
- 完全有向图中的每两个顶点之间都存在着方向相反的两条边。含有n个顶点的完全有向图包含有$n(n-1)$条边。



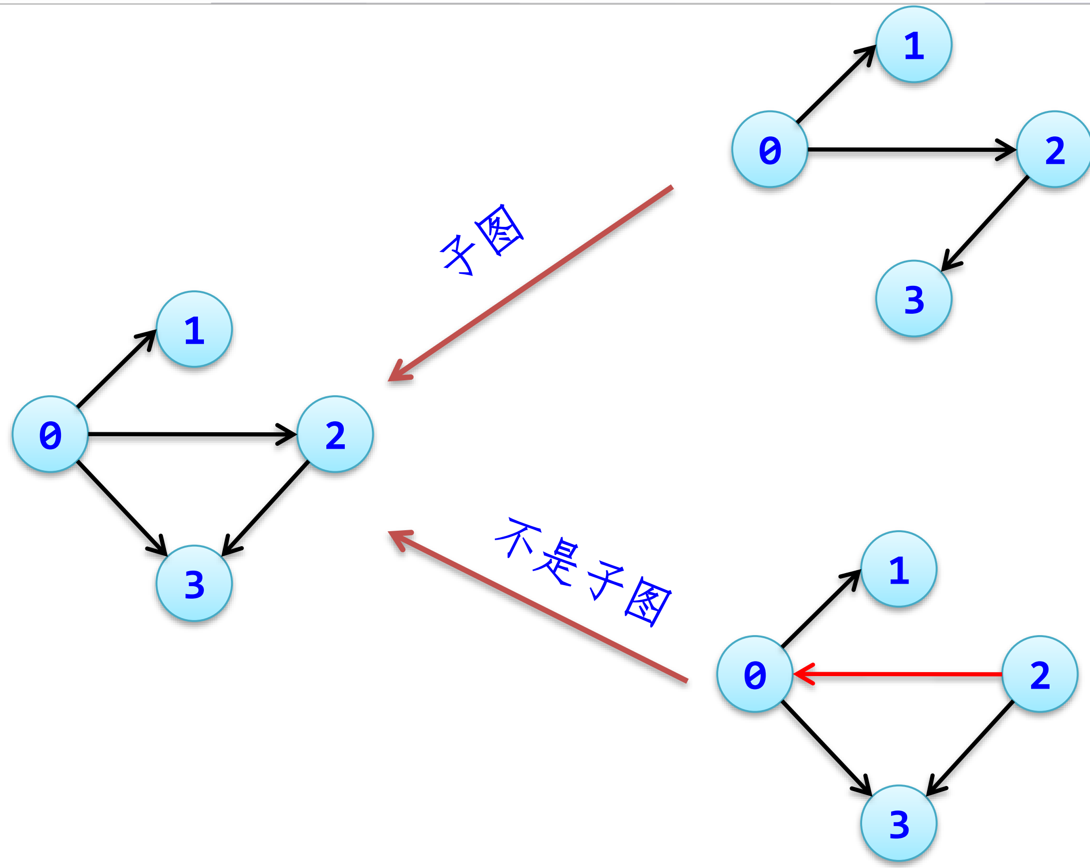
（a）一个完全无向图



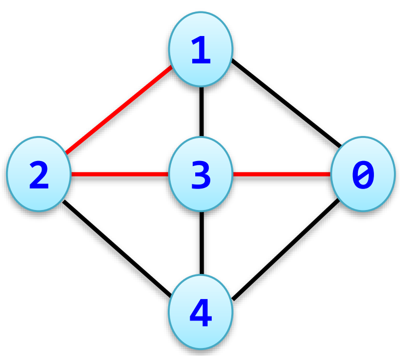
（b）一个完全有向图

- 当一个图接近完全图时，则称为稠密图。
- 当一个图含有较少的边数（即无向图有$e<<n(n-1)/2$，有向图有$e<<n(n-1)$）时，则称为稀疏图。

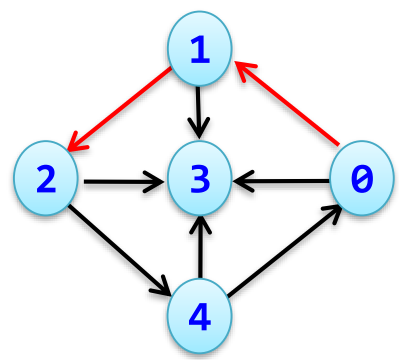
- 设有两个图G=(V，E)和G'=(V'，E')，若V'是V的子集，即V'⊆V，且E'是E的子集，即E'⊆E，则称G'是G的子图。



- 在一个图G=(V, E)中，从顶点i到顶点j的一条路径是一个顶点序列(i, i_1, i_2, …, i_m, j)，若此图G是无向图，则边(i, i_1)，(i_1, i_2)，…，(i_{m-1}, i_m)，(i_m, j)属于E(G)；若此图是有向图，则$<i, i_1>$，$<i_1, i_2>$，…，$<i_{m-1}, i_m>$，$<i_m, j>$属于E(G)。
- 路径长度是指一条路径上经过的边的数目。
- 若一条路径上除开始点和结束点可以相同外，其余顶点均不相同，则称此路径为简单路径。

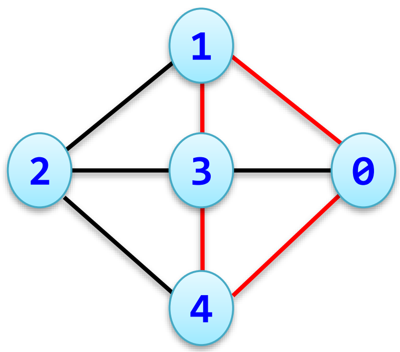


（0,3,2,1）的简单路径长度为3

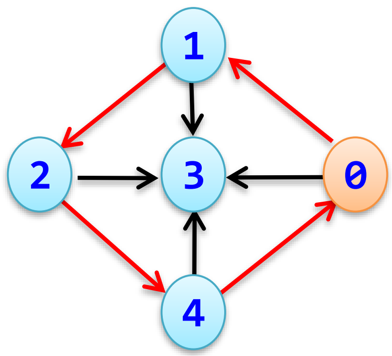


（0,1,2）的简单路径长度为2

- 若一条路径上的开始点与结束点为同一个顶点，则此路径被称为回路或环。
- 开始点与结束点相同的简单路径被称为简单回路或简单环。

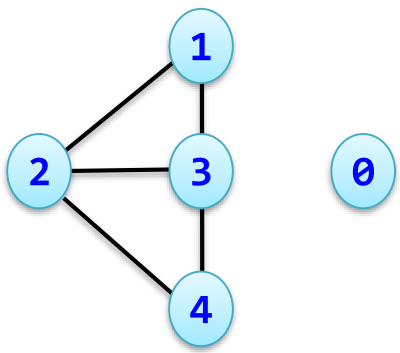


（0,1,3,4,0）的简单回路长度为4



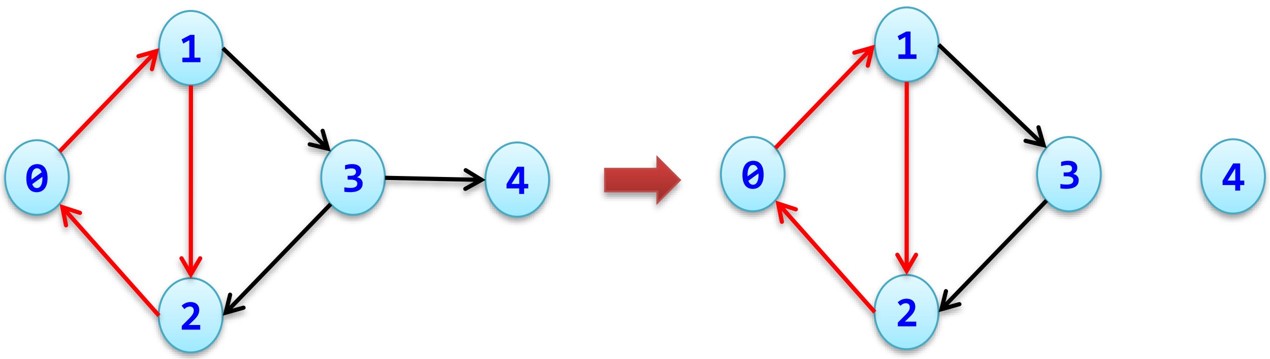
（0,1,2,4,0）的简单回路长度为4

- 在无向图G中，若从顶点i到顶点j有路径，则称顶点i和顶点j是连通的。
- 若图G中任意两个顶点都连通，则称G为连通图，否则称为非连通图。
- 无向图G中的极大连通子图称为G的连通分量。显然，任何连通图的连通分量只有一个即本身，而非连通图有多个连通分量。

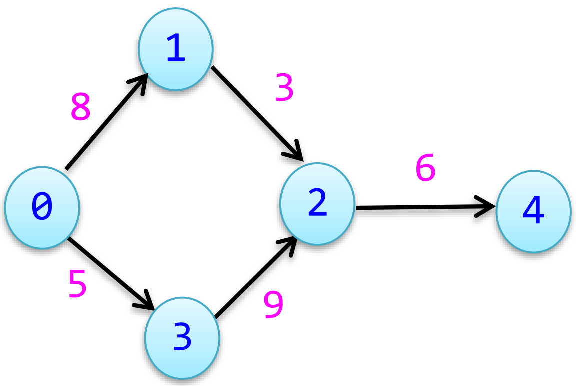


两个连通分量构成

- 在有向图G中，若从顶点i到顶点j有路径，则称从顶点i到顶点j是连通的。
- 若图G中的任意两个顶点i和j都连通，即从顶点i到顶点j和从顶点j到顶点i都存在路径，则称图G是强连通图。
- 有向图G中的极大强连通子图称为G的强连通分量。
- 显然，强连通图只有一个强连通分量即本身，非强连通图有多个强连通分量。一般地单个顶点自身就是一个强连通分量。



- 图中每一条边都可以附有一个对应的数值，这种与边相关的数值称为权。权可以表示从一个顶点到另一个顶点的距离或花费的代价。
- 边上带有权的图称为带权图，也称作网。



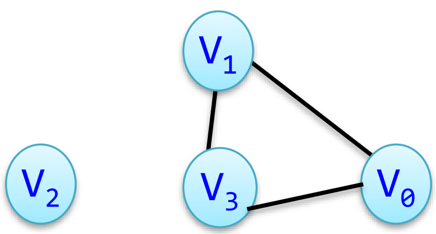
7.2 图的存储结构

7.2.1 邻接矩阵

数组表示法：用两个数组分别存储数据元素(顶点)的信息和数据元素之间的关系。

顶点数组：用一维数组存储顶点(元素)

邻接矩阵：用二维数组存储顶点(元素)之间的关系(边或弧)。



V_0	V_1	V_2	V_3
0	1	2	3

$$
\begin{array}{c c}
 & \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \end{array} & \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \end{bmatrix}
\end{array}
$$

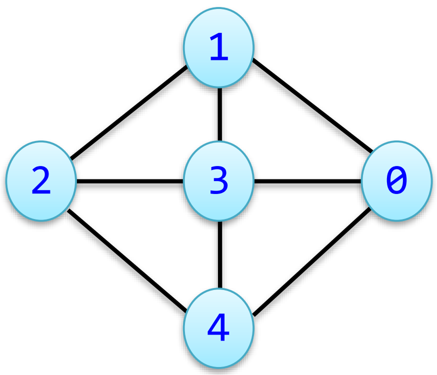
对称

（1）如果G是不带权图，则：

$$A[i][j]=\begin{cases}1 & 若(i,j)\in E(G)或者\langle i,j\rangle\in E(G)\\ 0 & 其他\end{cases}$$

（2）如果G是带权图，则：

$$A[i][j]=\begin{cases}w_{ij} & 若i\neq j并且(i,j)\in E(G)或者\langle i,j\rangle\in E(G)\\ 0 & 若i=j\\ \infty & 其他\end{cases}$$

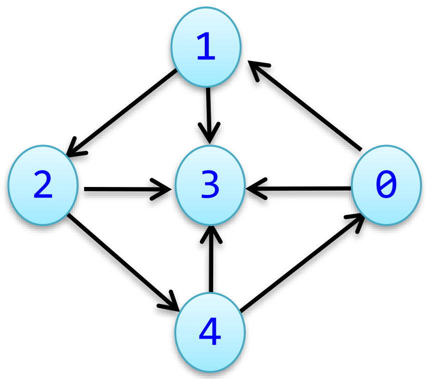


（a）一个无向图

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \end{bmatrix}$$

对称

说明 **无向图的邻接矩阵一定对称！**

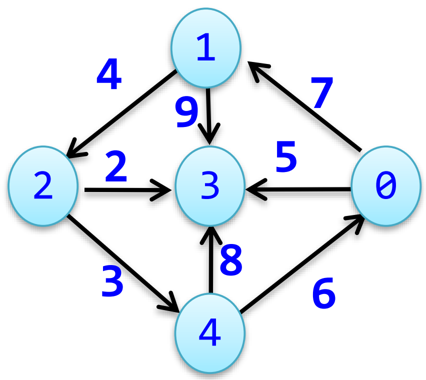


（b）一个有向图

$$A_2 = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix}$$

不对称

说明 **有向图的邻接矩阵不一定对称！**



（b）一个有向图

$$A_3 = \begin{bmatrix} 0 & 7 & \infty & 5 & \infty \\ \infty & 0 & 4 & 9 & \infty \\ \infty & \infty & 0 & 2 & 3 \\ \infty & \infty & \infty & 0 & \infty \\ 6 & \infty & \infty & 8 & 0 \end{bmatrix}$$

不对称

说明

有向带权图的邻接矩阵不一定对称！

邻接矩阵的特点

- 对于含有n个顶点的图，采用邻接矩阵存储时，其存储空间均为$O(n^2)$；邻接矩阵适合于存储边数较多的稠密图。
- 无向图的邻接矩阵一定是对称矩阵，在n很大时可采用压缩存储方法存储。
- 对无向图，其第i行(或第i列)非零元素(或非∞元素)个数是顶点i的度。
- 对有向图，其第i行(或第i列)非零元素(或非∞元素)的个数是顶点i的出度（或入度）。
- 用邻接矩阵存储图，确定任意两顶点之间是否有边相连的时间为O(1)。

抽象数据类型图的描述

```
ADT Graph
{
数据对象:
    D={a_i | 0≤i≤n-1, n≥0, a_i为int类型}  //a_i为每个顶点的唯一编号
数据关系:
    R={r}
    r={<a_i, a_j> | a_i, a_j∈D, 0≤i≤n-1, 0≤j≤n-1, 其中a_i可以有零个
          或多个前驱元素, 可以有零个或多个后继元素 }
基本运算:
    void CreateGraph(): 根据相关数据建立一个图。
    void DispGraph(): 输出一个图。
     …
}
```

说明 **约定用i（$0\leq i\leq n-1$）表示第i个顶点的编号。**

图的邻接矩阵类MatGraph

```
import copy
INF=0x3f3f3f3f                          #表示∞
class MatGraph:                         #图邻接矩阵类
  def __init__(self,n=0,e=0):           #构造方法
    self.edges=[]                       #邻接矩阵数组
    self.vexs=[]                        #vexs[i]存放顶点i的信息，暂时未用
    self.n=n                            #顶点数
    self.e=e                            #边数
  #图的基本运算算法
```

邻接矩阵数组

2. 图基本运算在邻接矩阵中的实现

（1）创建图的邻接矩阵

- 邻接矩阵数组*a*
- 顶点数*n*
- 边数*e*

→ 邻接矩阵g

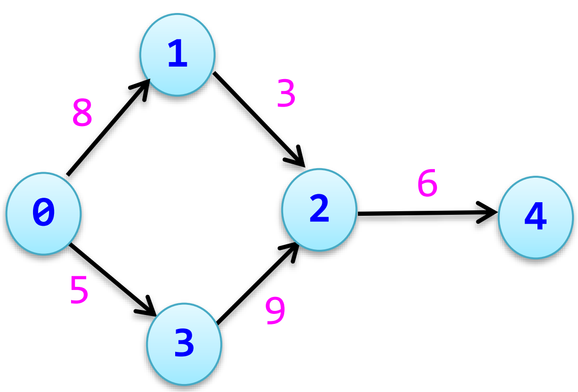
```
def CreateMatGraph(self,a,n,e):   #通过数组a、n和e建立图的邻接矩阵
  self.n=n                        #置顶点数和边数
  self.e=e
  self.edges=copy.deepcopy(a)     #深拷贝
```

（2）输出图

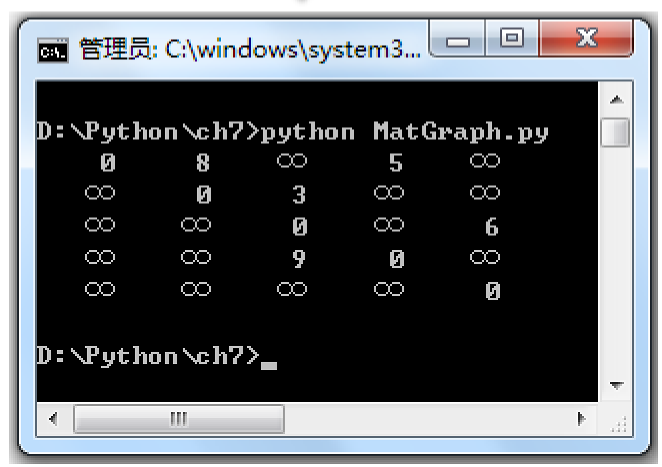
```
def DispMatGraph(self):                    #输出图的邻接矩阵
  for i in range(self.n):
    for j in range(self.n):
      if self.edges[i][j]==INF:
        print("%4s"%("∞"),end=' ')
      else:
        print("%5d" %(self.edges[i][j]),end=' ')
    print()
```

程序验证

```python
if __name__ == '__main__':
    g=MatGraph()
    n,e=5,5
    a=[ [0,8,INF,5,INF],
        [INF,0,3,INF,INF],
        [INF,INF,0,INF,6],
        [INF,INF,9,0,INF],
        [INF,INF,INF,INF,0]]
    g.CreateMatGraph(a,n,e)
    g.DispMatGraph()
```



一个带权有向图



【例7.2】一个含有n个顶点e条边的图采用邻接矩阵g存储，设计以下算法：

（1）该图为无向图，求其中顶点v的度。

（2）该图为有向图，求该图中顶点v的出度和入度。

- 无向图，求其中顶点v的度

```
def Degree1(g,v):                    #无向图邻接矩阵g中求顶点v的度
  d=0
  for j in range(g.n):               #统计第v行的非0非∞元素个数
     if g.edges[v][j]!=0 and g.edges[v][j]!=INF:
        d+=1
  return d
```

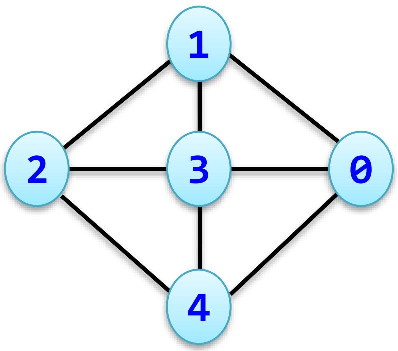
有向图，求该图中顶点v的出度和入度

```
def Degree2(g,v):                  #有向图邻接矩阵g中求顶点v的出度和入度
  ans=[0,0]                        #ans[0]累计出度,ans[1]累计入度
  for j in range(g.n):             #统计第v行的非0非∞元素个数为出度
    if g.edges[v][j]!=0 and g.edges[v][j]!=INF:
       ans[0]+=1
  for i in range(g.n):             #统计第v列的非0非∞元素个数为入度
    if g.edges[i][v]!=0 and g.edges[i][v]!=INF:
       ans[1]+=1
  return ans                       #返回出度和入度
```

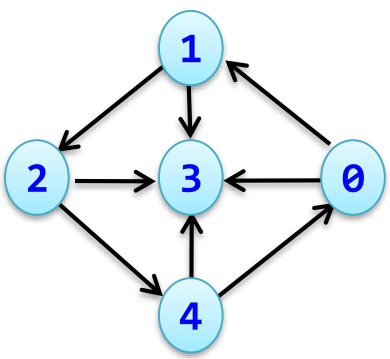
程序验证

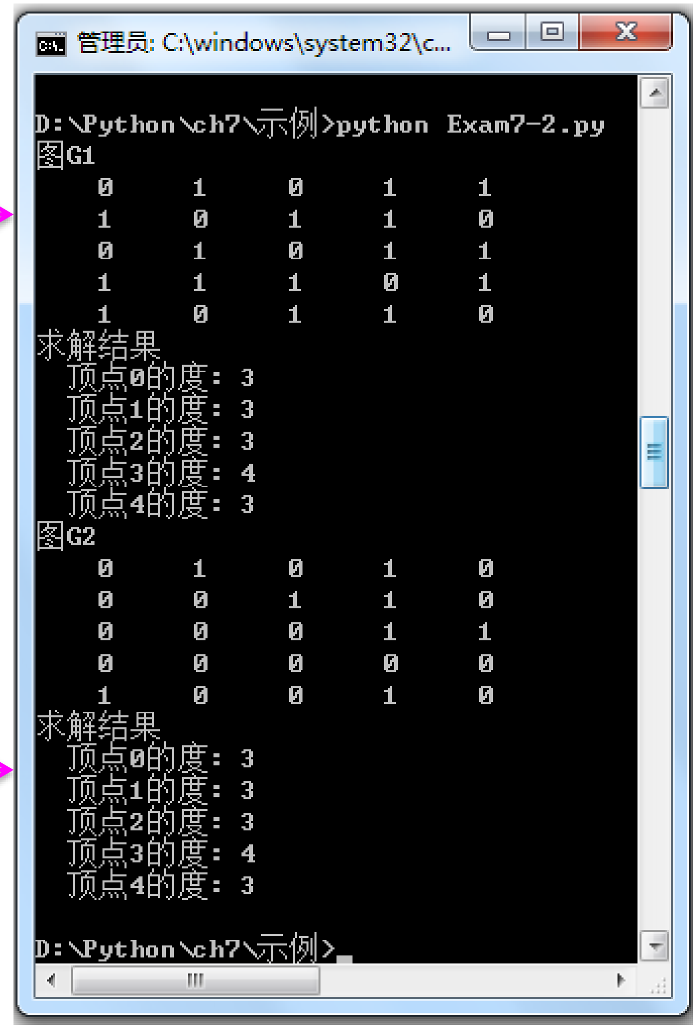
```
#主程序
g=MatGraph()
n,e=5,8
a=[[0,1,0,1,1],[1,0,1,1,0],[0,1,0,1,1],[1,1,1,0,1],[1,0,1,1,0]]
g.CreateMatGraph(a,n,e)
print("图G1")
g.DispMatGraph()
print("求解结果");
for i in range(g.n):
  print("  顶点%d的度: %d" %(i,Degree1(g,i)))

g1=MatGraph()
n,e=5,8
b=[[0,1,0,1,0],[0,0,1,1,0],[0,0,0,1,1],[0,0,0,0,0],[1,0,0,1,0]]
g1.CreateMatGraph(b,n,e)
print("图G2")
g1.DispMatGraph()
print("求解结果");
for i in range(g1.n):
   ans=Degree2(g1,i)
   print("  顶点%d的度: %d" %(i,ans[0]+ans[1]))
```

（a）一个无向图



（b）一个有向图



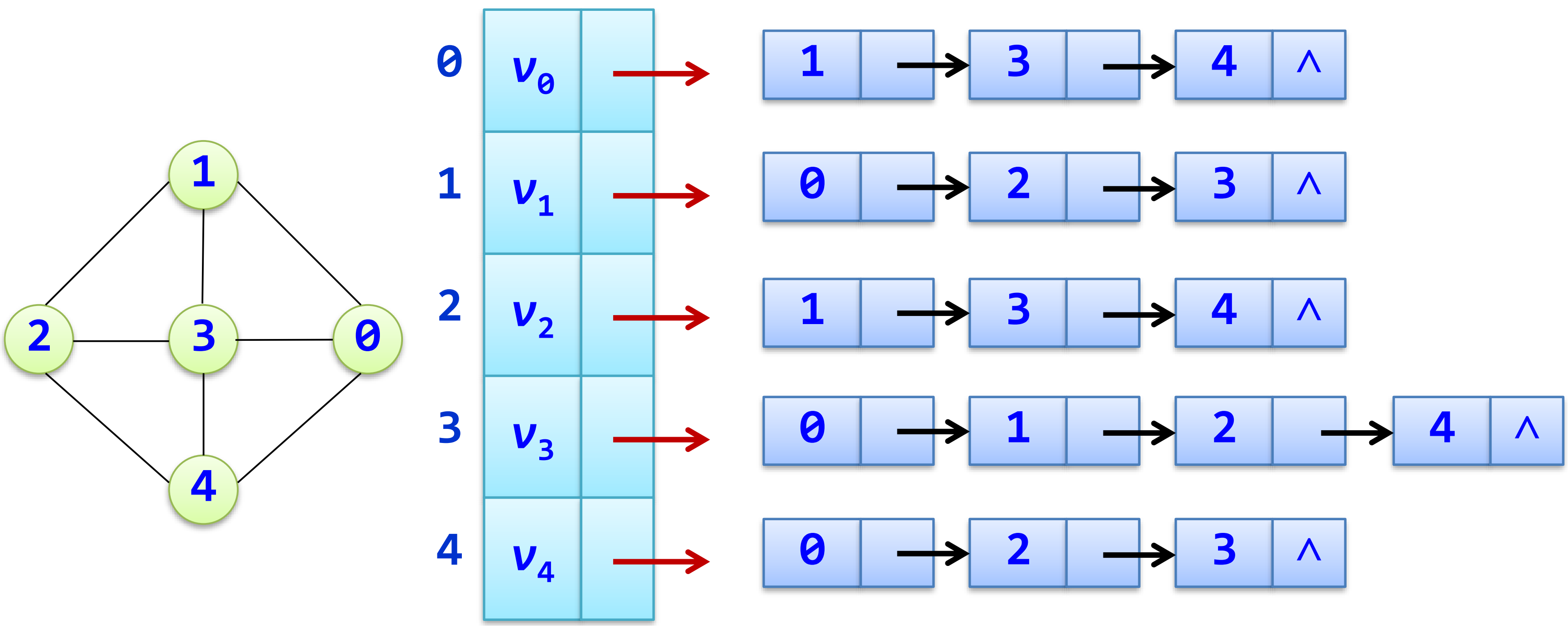


```
管理员: C:\windows\system32\c...

D:\Python\ch7\示例>python Exam7-2.py
图G1
    0     1     0     1     1
    1     0     1     1     0
    0     1     0     1     1
    1     1     1     0     1
    1     0     1     1     0
求解结果
  顶点0的度: 3
  顶点1的度: 3
  顶点2的度: 3
  顶点3的度: 4
  顶点4的度: 3
图G2
    0     1     0     1     0
    0     0     1     1     0
    0     0     0     1     1
    0     0     0     0     0
    1     0     0     1     0
求解结果
  顶点0的度: 3
  顶点1的度: 3
  顶点2的度: 3
  顶点3的度: 4
  顶点4的度: 3

D:\Python\ch7\示例>_
```

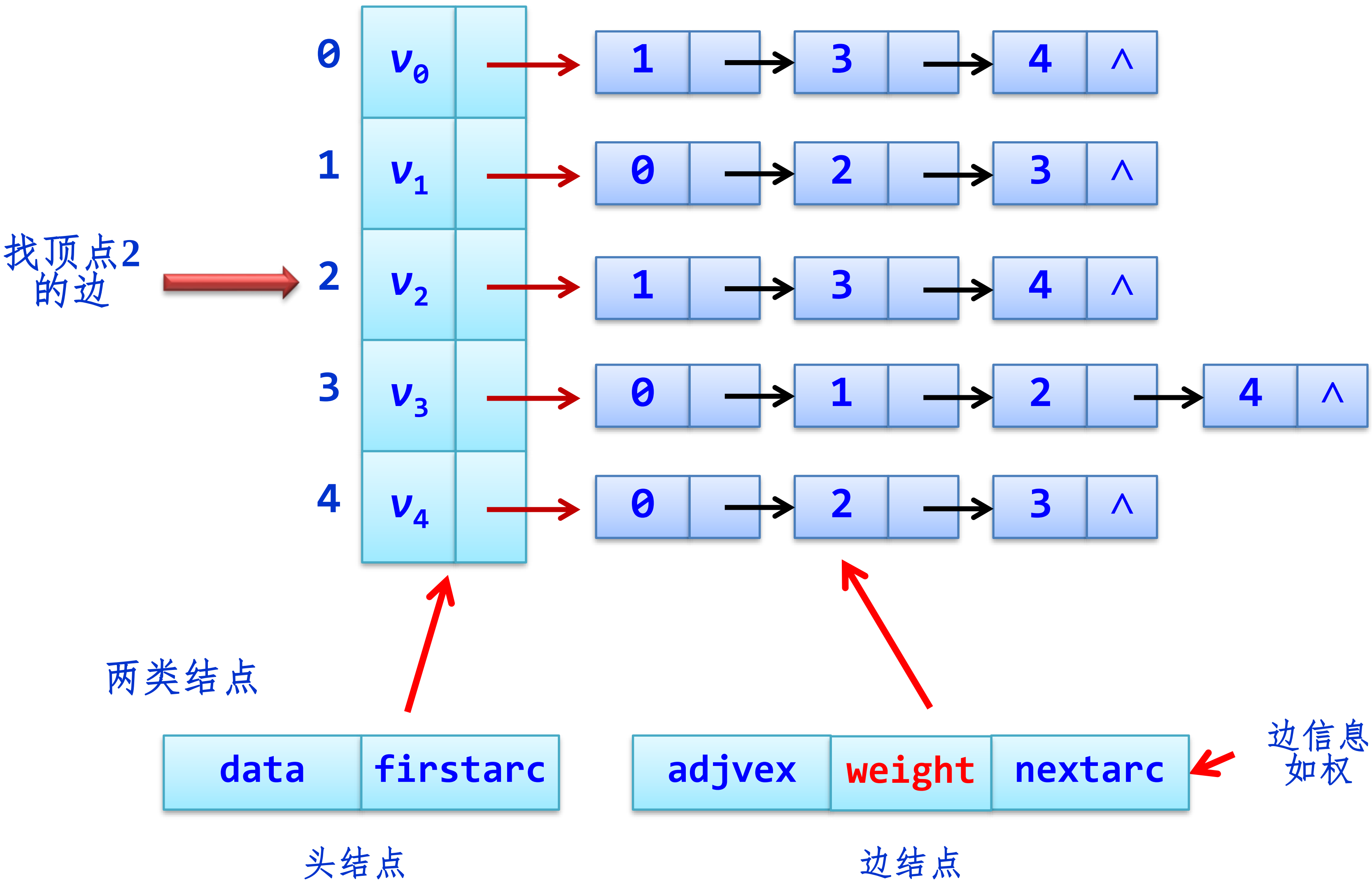
7.2.2 邻接表

- 对图中每个顶点*i*建立一个单链表，将顶点*i*的所有邻接点链起来。

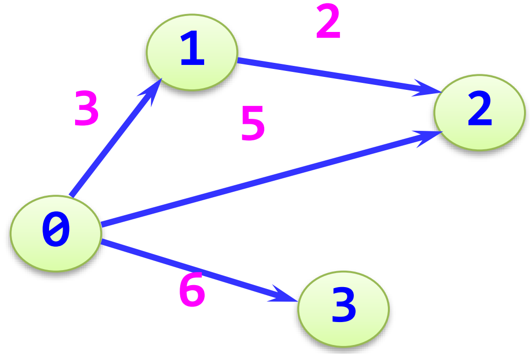
- 每个单链表上添加一个表头结点（表示顶点信息）。并将所有表头结点构成一个数组，下标为*i*的元素表示顶点*i*的表头结点。

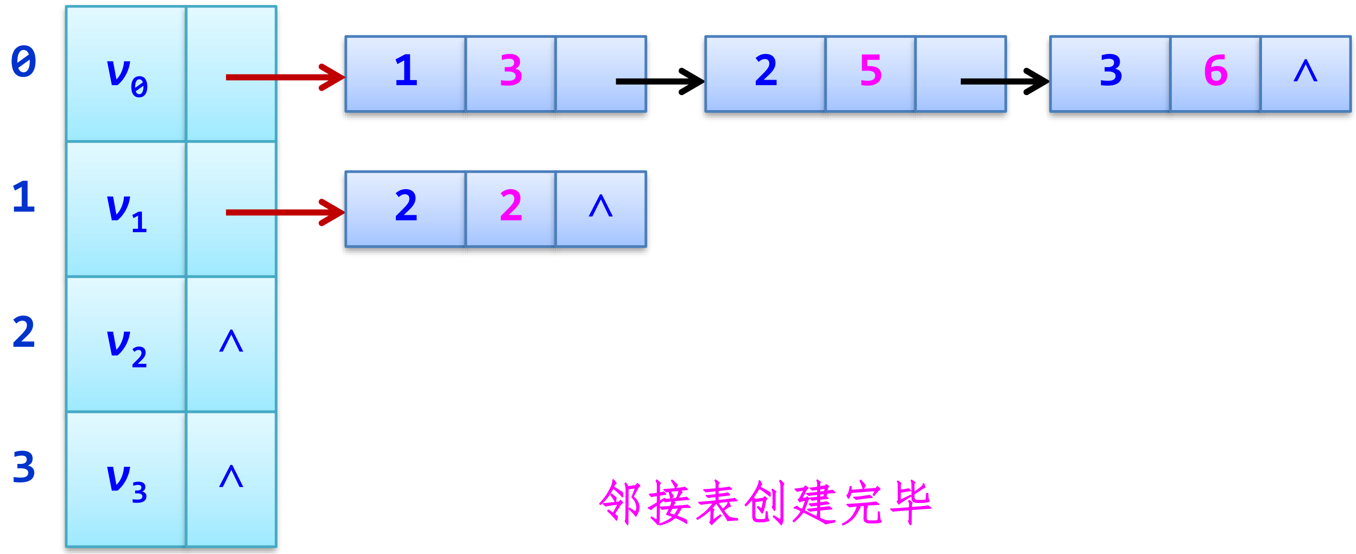


图的邻接表存储方法是一种顺序分配与链式分配相结合的存储方法。



一个网络

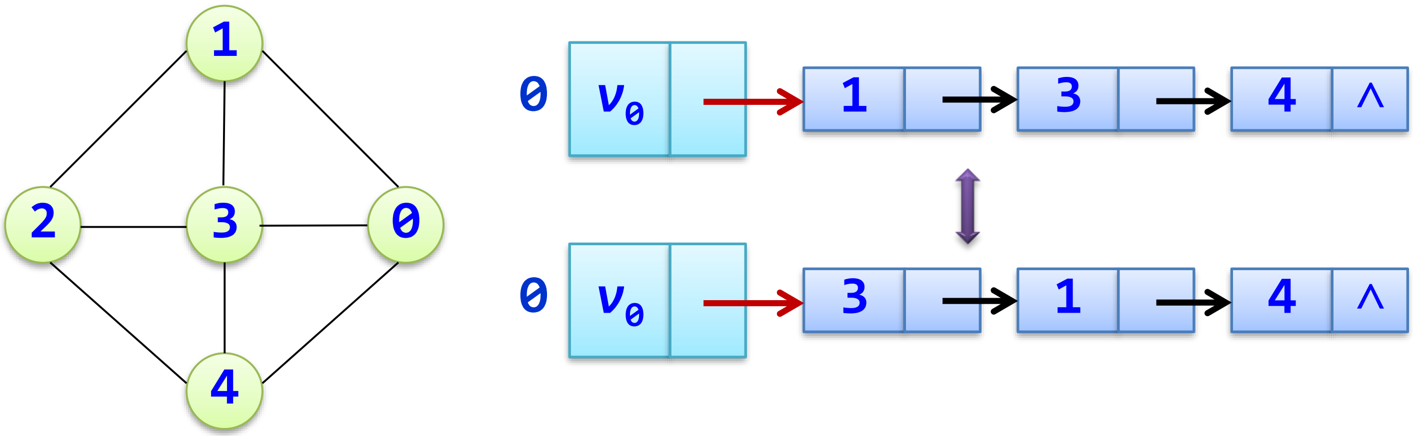




邻接表创建完毕

邻接表的特点：

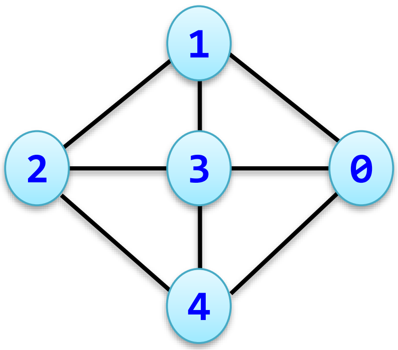
- 邻接表表示不唯一；



- 特别适合于稀疏图存储；

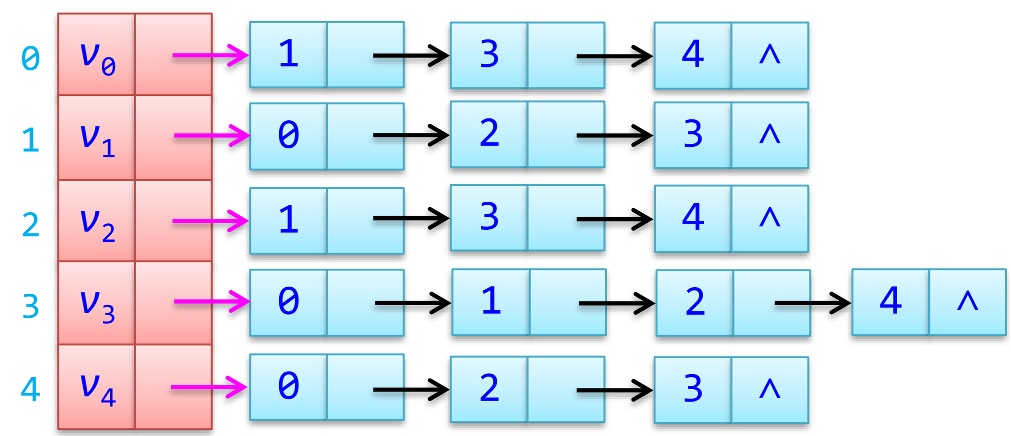
邻接表的存储空间为**O(*n*+*e*)**

- 无向图中顶点的度：每个顶点的邻接列表中的结点个数；
- 有向图中顶点的出度和入度：出度即为每个顶点的出边列表中的结点个数，而入度是所有出边列表中含顶点i的出边列表的个数。




```
[
    [[1, 1], [3, 1], [4, 1]]             #顶点0
    [[0, 1], [2, 1], [3, 1]]             #顶点1
    [[1, 1], [3, 1], [4, 1]]             #顶点2
    [[0, 1], [1, 1], [2, 1], [4, 1]]     #顶点3
    [[0, 1], [2, 1], [3, 1]]             #顶点4
]
```

Python中的简化表示



每个边结点的类型ArcNode定义如下

```
class ArcNode:                              #边结点
    def __init__(self,adjv,w):              #构造方法
        self.adjvex=adjv                    #邻接点
        self.weight=w                       #边的权值
```

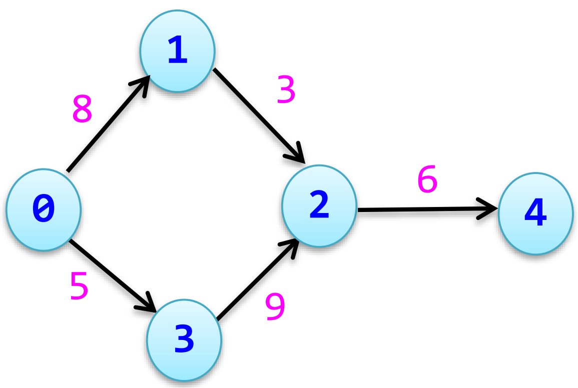
```
[
   [[1, 1], [3, 1], [4, 1]]              #顶点0
   [[0, 1], [2, 1], [3, 1]]              #顶点1
   [[1, 1], [3, 1], [4, 1]]              #顶点2
   [[0, 1], [1, 1], [2, 1], [4, 1]]      #顶点3
   [[0, 1], [2, 1], [3, 1]]              #顶点4
]
```

图的邻接表存储类AdjGraph

```
class AdjGraph:                                  #图邻接表类
  def __init__(self,n=0,e=0):                    #构造方法
     self.adjlist=[]                             #邻接表数组
     self.vexs=[]                                #vexs[i]存放顶点i的信息，暂未用
     self.n=n                                    #顶点数
     self.e=e                                    #边数
   #图的基本运算算法
```

扩展

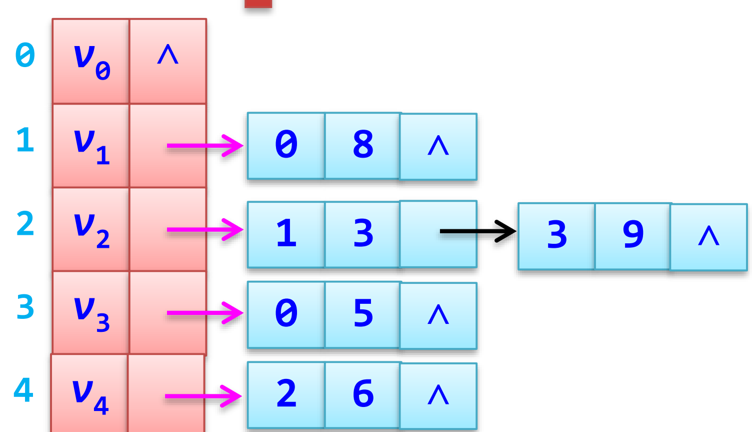
逆邻接表



一个带权有向图

```
[
    []                      #顶点0的入边列表
    [[0, 8]]                #顶点1的入边列表
    [[1, 3], [3, 9]]        #顶点2的入边列表
    [[0, 5] ]               #顶点3的入边列表
    [2, 6]                  #顶点4的入边列表
]
```

用列表简化表示



☎ 方便查找每个顶点的入边

☎ 方便计算入度

2．图基本运算在邻接表中的实现

（1）创建图的邻接表

- 邻接矩阵数组*a*
- 顶点数*n*
- 边数*e*

邻接表G

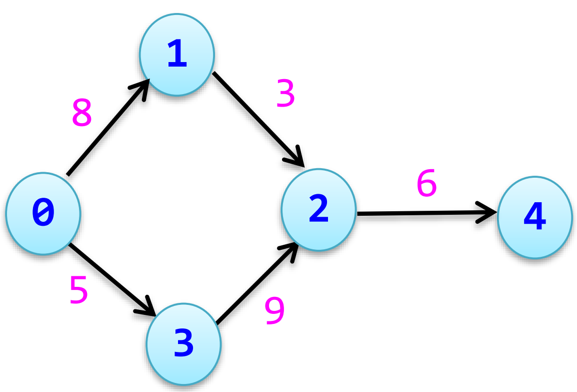
```
def CreateAdjGraph(self,a,n,e):            #通过数组a、n和e建立图的邻接表
  self.n=n                                 #置顶点数和边数
  self.e=e
  for i in range(n):                       #检查边数组a中每个元素
     adi=[]                                #存放顶点i的邻接点,初始为空
     for j in range(n):
       if a[i][j]!=0 and a[i][j]!=INF:     #存在一条边
          p=ArcNode(j,a[i][j])             #创建<j,a[i][j]>出边的结点p
          adi.append(p)                    #将结点p添加到adi中
     self.adjlist.append(adi)
```

（2）输出图

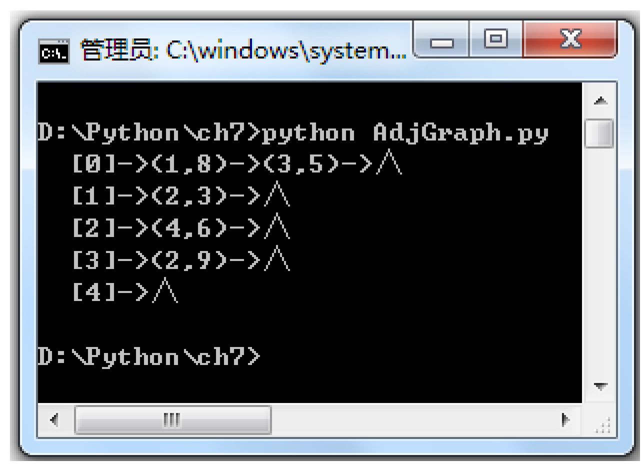
```
def DispAdjGraph(self):                         #输出图的邻接表
  for i in range(self.n):                       #遍历每一个顶点i
    print("  [%d]" %(i),end='')
    for p in self.adjlist[i]:
       print("->(%d,%d)" %(p.adjvex,p.weight),end='')
    print("->∧")
```

程序验证

```
if __name__ == '__main__':
    G=AdjGraph()
    n,e=5,5
    a=[ [0,8,INF,5,INF],
        [INF,0,3,INF,INF],
        [INF,INF,0,INF,6],
        [INF,INF,9,0,INF],
        [INF,INF,INF,INF,0]]
    G.CreateAdjGraph(a,n,e)
    G.DispAdjGraph()
```



一个带权有向图



【例7.3】一个含有*n*个顶点*e*条边的图采用邻接表存储，设计以下算法：

（1）该图为无向图，求其中顶点*v*的度。

（2）该图为有向图，求该图中顶点*v*的出度和入度。

- 无向图，求其中顶点*v*的度

```
def DeGree1(G,v):                    #无向图邻接表G中求顶点v的度
  return len(G.adjlist[v])           #顶点v的度为G.adjlist[v]的长度
```

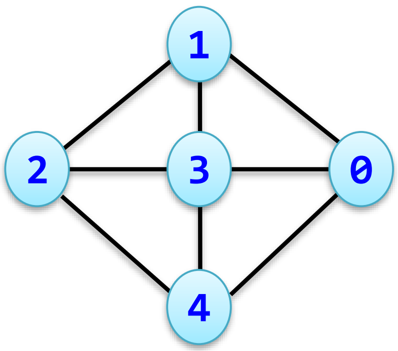
■ 有向图，求该图中顶点v的出度和入度

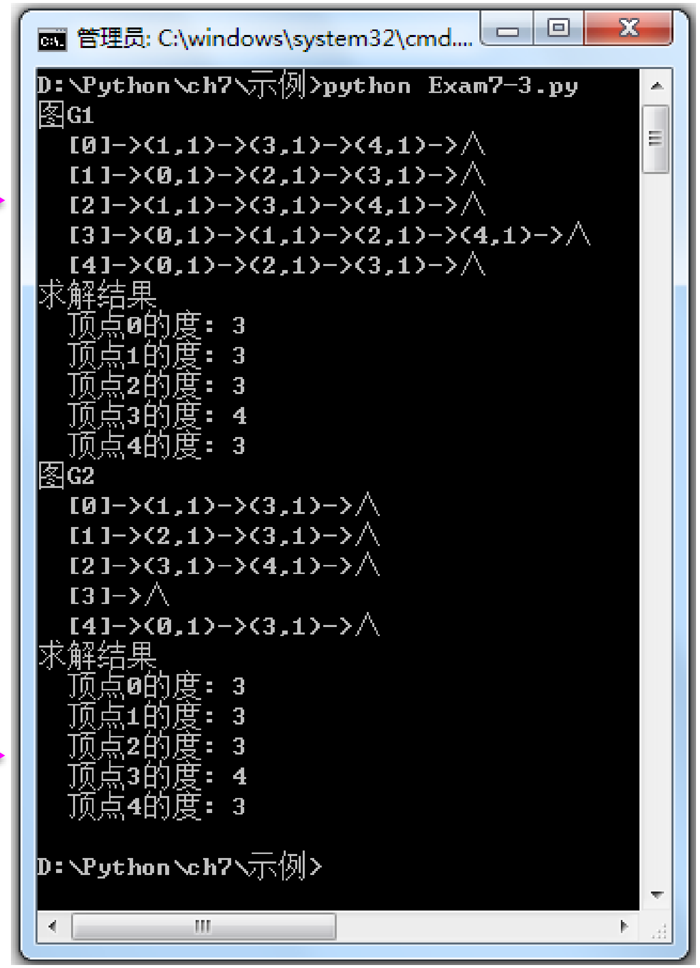
```
def DeGree2(G,v):                     #有向图邻接表G中求顶点v的出度和入度
  ans=[0,0]                           #ans[0]累计出度,ans[1]累计入度
  ans[0]=len(G.adjlist[v])            #顶点v的出度为G.adjlist[v]的长度
  for i in range(G.n):                #遍历所有的头结点
     for p in G.adjlist[i]:
        if p.adjvex==v:               #存在<i,v>的边
           ans[1]+=1                  #顶点v的入度增加1
           break
   return ans                         #返回出度和入度
```

程序验证

```
#主程序
G=AdjGraph()
n,e=5,8
a=[[0,1,0,1,1],[1,0,1,1,0],[0,1,0,1,1],[1,1,1,0,1],[1,0,1,1,0]]
G.CreateAdjGraph(a,n,e)
print("图G1")
G.DispAdjGraph()
print("求解结果");
for i in range(G.n):
  print("  顶点%d的度: %d" %(i,DeGree1(G,i)))

G1=AdjGraph()
n,e=5,8
b=[[0,1,0,1,0],[0,0,1,1,0],[0,0,0,1,1],[0,0,0,0,0],[1,0,0,1,0]]
G1.CreateAdjGraph(b,n,e)
print("图G2")
G1.DispAdjGraph()
print("求解结果");
for i in range(G1.n):
  ans=DeGree2(G1,i)
  print("  顶点%d的度: %d" %(i,ans[0],ans[0]+ans[1]))
```

（a）一个无向图




管理员: C:\windows\system32\cmd....

```
D:\Python\ch7\示例>python Exam7-3.py
图G1
  [0]->(1,1)->(3,1)->(4,1)->∧
  [1]->(0,1)->(2,1)->(3,1)->∧
  [2]->(1,1)->(3,1)->(4,1)->∧
  [3]->(0,1)->(1,1)->(2,1)->(4,1)->∧
  [4]->(0,1)->(2,1)->(3,1)->∧
求解结果
  顶点0的度: 3
  顶点1的度: 3
  顶点2的度: 3
  顶点3的度: 4
  顶点4的度: 3
图G2
  [0]->(1,1)->(3,1)->∧
  [1]->(2,1)->(3,1)->∧
  [2]->(3,1)->(4,1)->∧
  [3]->∧
  [4]->(0,1)->(3,1)->∧
求解结果
  顶点0的度: 3
  顶点1的度: 3
  顶点2的度: 3
  顶点3的度: 4
  顶点4的度: 3

D:\Python\ch7\示例>
```

（b）一个有向图