数据结构与算法

疫情面前，让我们一起努力！

1

图状结构？

2

填空题 5分

图G是由一个 [填空1] 和一个 [填空2] 构成的有序对。记作: G= (V, E)

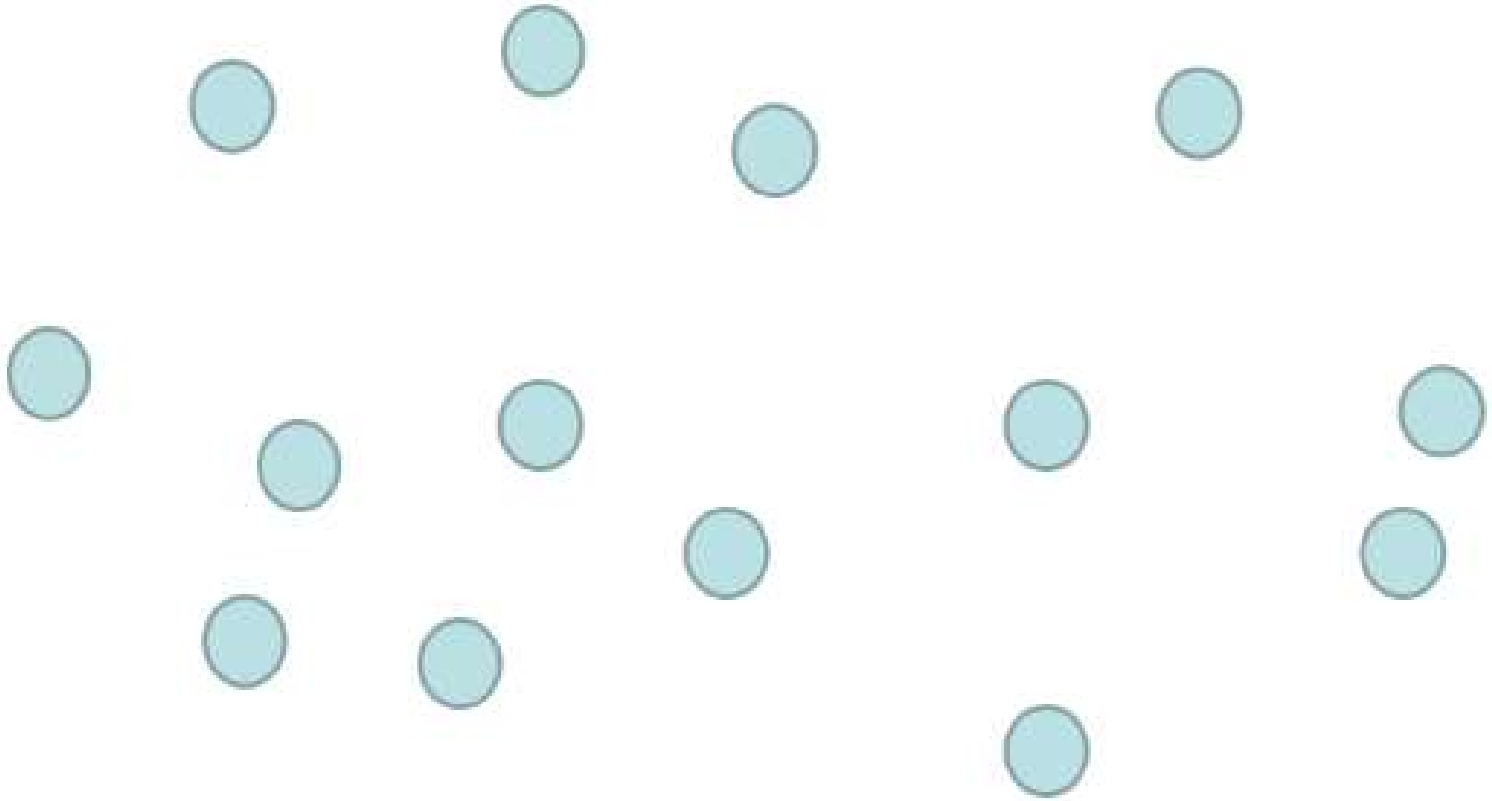
其中， <v,w> ∈ E 表示从 v 到 w 的一条弧，并称 v 为弧头，w 为弧尾。

每条边都是无向边的图称为 [填空3] 。

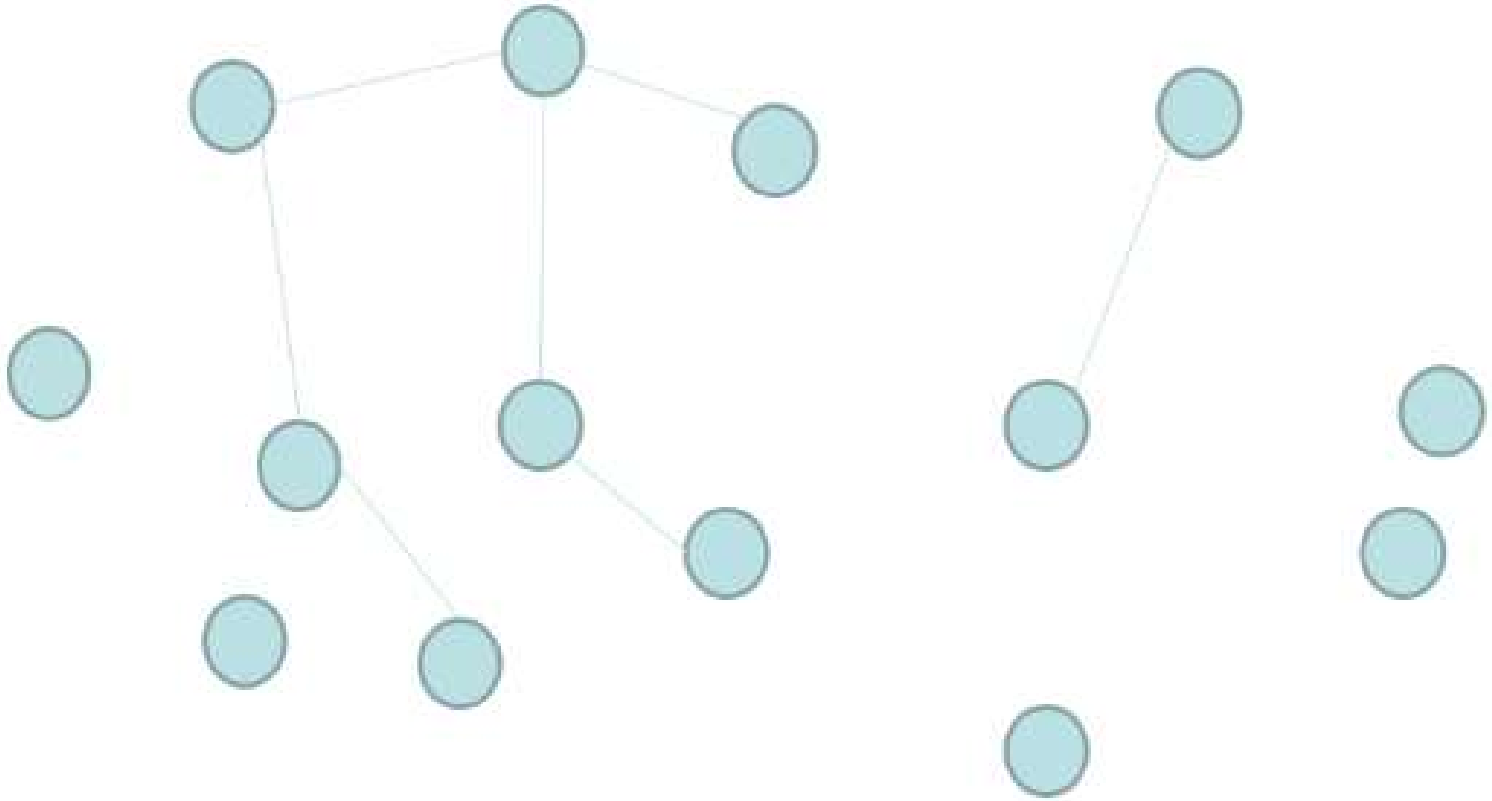
每条边都是有向边的图称为 [填空4] 。

假若顶点v 和顶点w 之间存在一条边，则称顶点v 和w 互为 [填空5] 。

图的基本概念



图的基本概念



5

第六章　图

- 6.1 图的定义和术语
- 6.2 图的存储结构
- 6.3 图的遍历
- 6.4 图的连通性问题
- 6.5 有向无环图及其应用
- 6.6 最短路径

6.1 图的定义和术语

- 图的定义
- 图的术语
- 图的抽象数据类型

6.1图的定义与术语

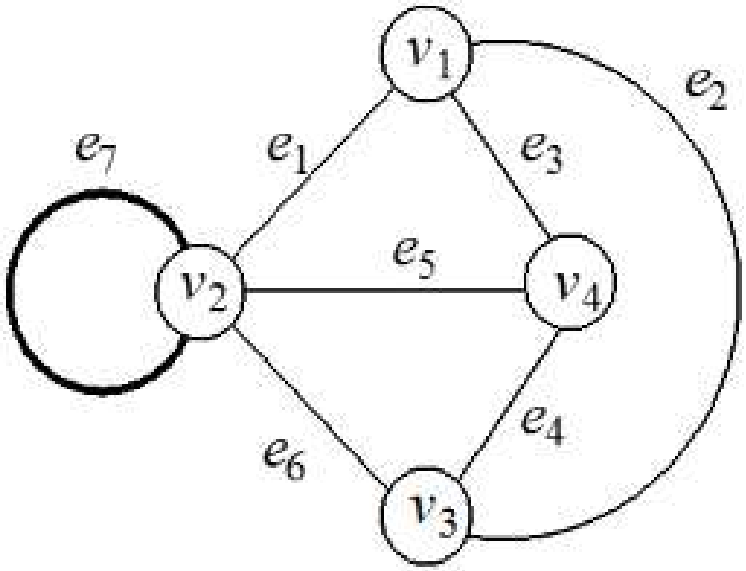
6.1.1图的相关术语

图G是由一个顶点集 V 和一个边集（或弧集） E 构成的有序对。记作: G= (V, E)

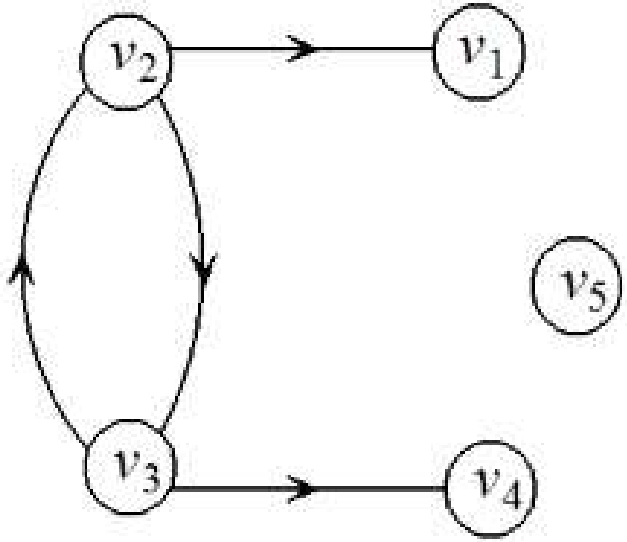
其中， <v,w> ∈ E 表示从 v 到 w 的一条弧，并称 v 为弧头， w 为弧尾。

由每条边都是无向边的图称为无向图。

每条边都是有向边的图称为有向图。



（1）无向图



（2）有向图

图7-1 无向图和有向图

名词和术语

网、子图、生成子图➡

完全图、稀疏图、稠密图 ➡

邻接点、度、入度、出度 ➡

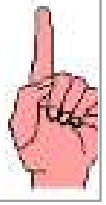
路径、路径长度、简单路径、简单回路➡

连通图、连通分量、
强连通图、强连通分量 ➡

生成树、生成森林 ➡

弧或边带权的图分别称作有向网或无向网。

设图G=(V, R) 和图 G′=(V′, R′), 若 V′⊆V, R′⊆R,则称 G′ 为 G 的子图。若 V′=V, R′⊆R,则称 G′ 为 G 的生成子图。



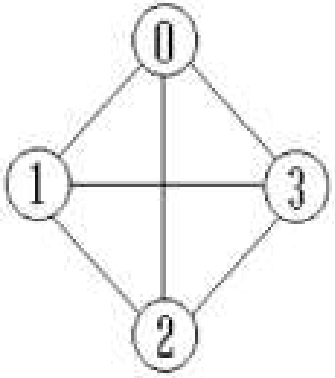
假设图中有n 个顶点e 条边，则

含有 e=n(n-1)/2 条边的无向图称作完全图；

含有 e=n(n-1) 条弧的有向图称作有向完全图；

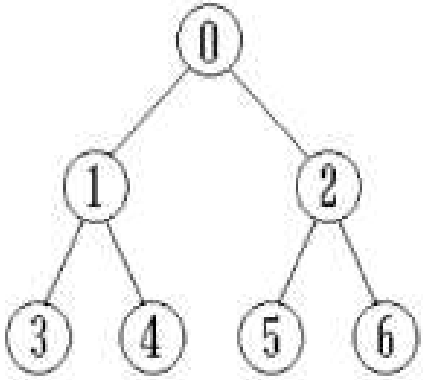
若边或弧的个数 e<nlogn，则称作稀疏图，否则称作稠密图。

6.1 图的定义和术语



(a) G1

无向 完全图

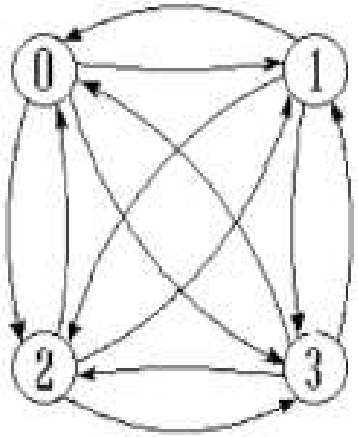


(b) G2

无向图(树)

(c) G3

有向图



(d) G4

有向完全图

13

假若顶点v 和顶点w 之间存在一条边，则称顶点v 和w 互为邻接点，边(v,w) 和顶点v 和w 相关联。

和顶点v 关联的边的数目定义为边的度。

对有向图来说：

顶点的入度：以顶点v为弧头的弧的数目。

顶点的出度：以顶点v为弧尾的弧的数目；

顶点的度=出度+入度

14

设图G=(V,R)中的一个顶点序列

$\{ u=v_{i,0},v_{i,1}, \cdots, v_{i,m}=w\}$中，$(v_{i,j-1},v_{i,j})\in R$ $1\le j\le m$,

则称从顶点u 到顶点w 之间存在一条路径。

若起点与终点相同，则称其为回路。

路径上边的数目称作路径长度。

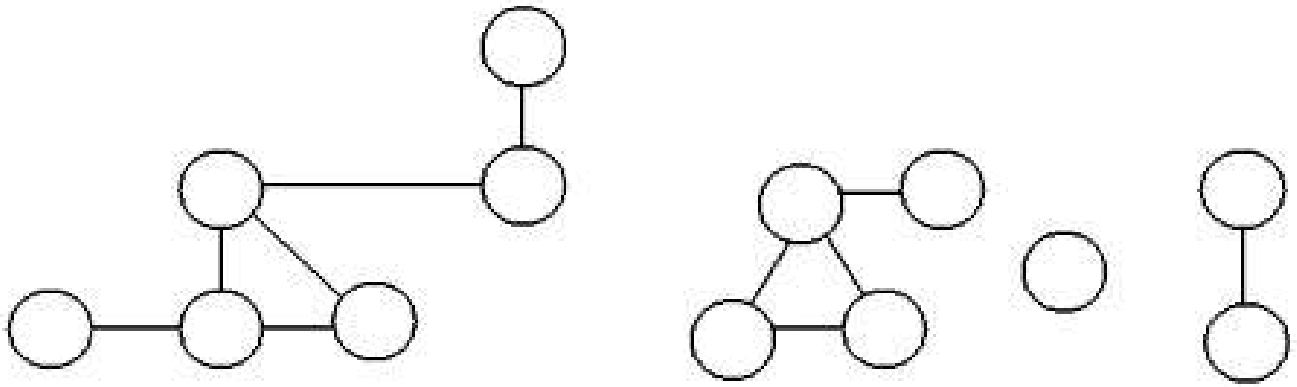
简单路径：顶点不重复的路径。

简单回路：顶点不重复的回路。

15

若无向图G中任意两个顶点之间都有路径相通，则称此图为连通图；

若无向图为非连通图，则图中各个极大连通子图称作此图的连通分量。



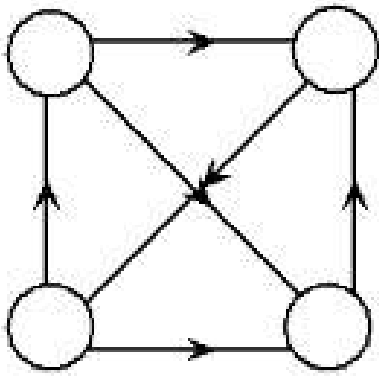
（1）连通图　　（2）连通分量

图7-4 连通图与连通分量

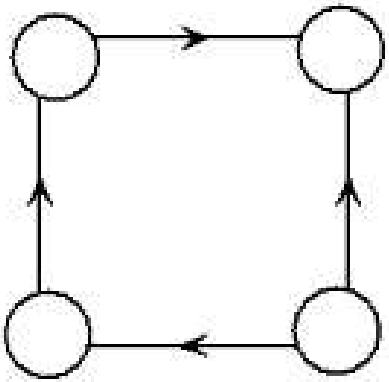
16

对有向图，若任意两个顶点之间都存在一条有向路径，则称此有向图为强连通图。

否则其各个强连通子图称作它的强连通分量。



（1）强连通图



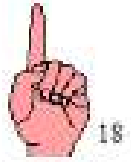
（2）非强连通图

图7-5 强连通图与非强连通图

17

假设一个连通图有 **n** 个顶点和 **e** 条边，其中 **n-1** 条边和 **n** 个顶点构成一个极小连通子图，称该极小连通子图为此连通图的生成树。

对非连通图，则称由各个连通分量的生成树的集合为此非连通图的生成森林。



6.1.2图的抽象数据类型定义

ADT Graph {

数据对象V：

具有相同特性的元素的集合。

数据关系：

R={VR}，VR={<v,w>|v,w∈V且P(v,w)，<v,w>表示从v到w的弧或边，谓词P(v,w)定义了边或弧<v,w>的意义或信息}

19

基本操作：

结构的建立和销毁

对顶点的访问操作

插入或删除顶点

插入和删除弧

对邻接点的操作

遍历

} ADT List

对邻接点的操作

FirstAdjVex(G, v);

// 返回 v 的“第一个邻接点”。若该顶点
//在 G 中没有邻接点，则返回“空”。

NextAdjVex(G, v, w);

// 返回 v 的（相对于 w 的）“下一个邻接
// 点”。若 w 是 v 的最后一个邻接点，则
// 返回“空”。

25

遍 历

DFSTraverse(G, v);

//从顶点v起深度优先遍历图G，并对每

//个顶点遍历一次且仅一次。

BFSTraverse(G, v);

//从顶点v起广度优先遍历图G，并对每

//个顶点遍历一次且仅一次。

6.2 图的存储结构

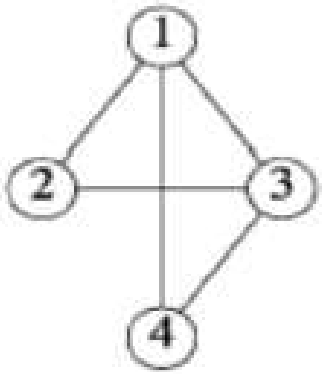
图的四种常用的存储形式：

- 邻接矩阵和加权邻接矩阵（labeled adjacency matrix）
- 邻接表
- 十字链表
- 邻接多重表

一、(加权)邻接矩阵（labeled adjacency matrix）

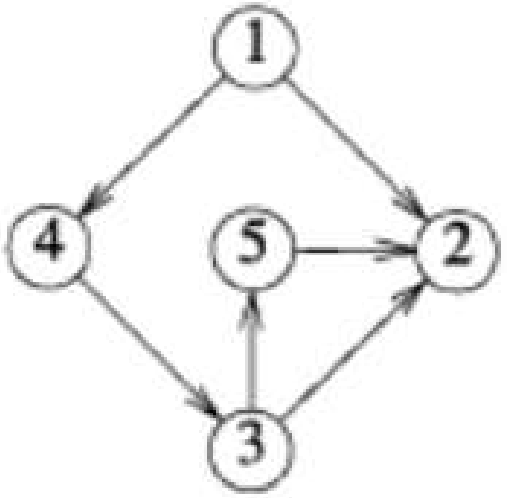
设图 **G= <V, E>**是一个有 ***n*** 个顶点的图，则图的邻接矩阵是一个二维数组 **G.arcs[n][n]**，定义：

$$G.arcs[i][j]=\begin{cases}1, & \text{如果} <i,j>\in E \text{ 或者 } (i,j)\in E\\ 0, & \text{否则}\end{cases}$$



无向图G_1

$$G_1.arcs = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix}$$



有向图G_2

$$G_2.arcs = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

一、(加权)邻接矩阵（continue）

从邻接矩阵中可以反映图的许多特征:

- 无向图

(1) 对称矩阵(可采用压缩存储);

(2) 每一行(或列)1的个数是该顶点的度;

(3) 主对角线全为0(简单图);

一、(加权)邻接矩阵（continue）

- 有向图

(1) 每一行1的个数是该顶点的出度; $OD(v_i) = \sum_{j=1}^{n} a_{ij}$

(2) 每一列1的个数是该顶点的入度; $ID(v_i) = \sum_{j=1}^{n} a_{ij}$

(3) 主对角线全为0(简单图);

- *有向图的邻接矩阵不一定对称。*

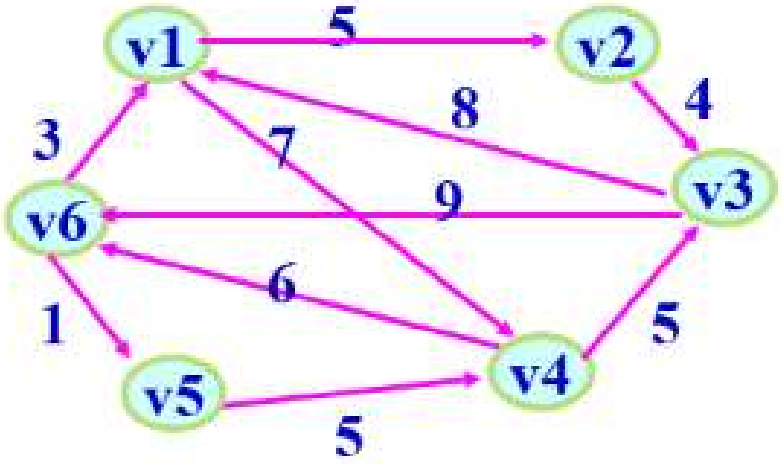
一、(加权)邻接矩阵（continue）

网络的邻接矩阵

定义为：$G.arcs[i][j]=\begin{cases} W_{ij} & <vi, vj> 或 (vi, vj) \in VR \\ \infty & 无边（弧） \end{cases}$

以有向网为例：

N



顶点表：（ v1 v2 v3 v4 v5 v6 ）

邻接矩阵：

N.*Edge* =

∞	5	∞	7	∞	∞
∞	∞	4	∞	∞	∞
8	∞	∞	∞	∞	9
∞	∞	5	∞	∞	6
∞	∞	∞	5	∞	∞
3	∞	∞	∞	1	∞

用邻接矩阵表示的图的存储表示

```
#define INFINITY INT_MAX
#define MAX_VERTEX_NUM 20
typedef enum{DG, DN,UDG,UDN} GraghKind;
typedef struct ArcCell { // 弧的定义
    VRType  adj;    // VRType是顶点关系类型。
          // 对无权图，用1或0表示相邻否；
          // 对带权图，则为权值类型。
  InfoType  *info; // 该弧相关信息的指针
} ArcCell,
AdjMatrix[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
```

用邻接矩阵表示的图的存储表示(continue)

```
typedef struct { // 图的定义
 VertexType    vexs[MAX_VERTEX_NUM];// 顶点信息
 AdjMatrix     arcs;      // 弧的信息
 int    vexnum, arcnum;  // 顶点数，弧数
 GraphKind   kind;     // 图的种类标志
} MGraph;
```

例：用邻接矩阵生成无向网的算法

```
Status CreateUDN(Mgraph &G){//用邻接矩阵表示
 1.scanf(&G.vexnum, &G.arcnum, &IncInfo);
 2.for(i=0;i<G.vexnum,;++i) scanf(&G.vexs[i] );
 3.for(i=0; i<G.vexnum; ++i)
 4.for(j=0;j<G.vexnum;++j)
 5. G.arcs[i][j]={INFINITY, NULL};
 6.for(k=0;k<G.arcnum;++k)
      {scanf(&v1, &v2, &w);        i=LocateVex(G,v1);
       j=LocateVex(G,v2);    G.arcs[i][j].adj=w;
       If(IncInfo)   Input(*G.arcs[i][j].info);
       G.arcs[j][i] = G.arcs [i] [j];  }
```

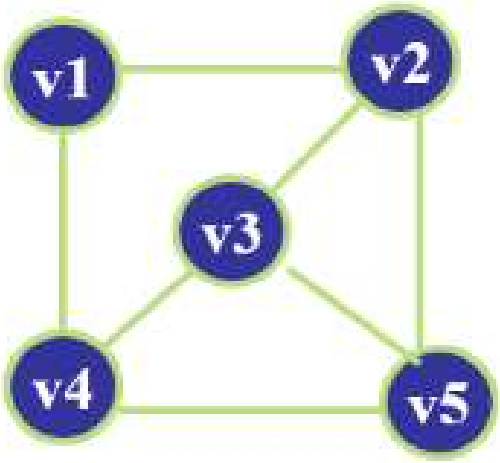
对于n个顶点e条弧的网，
建网时间效率 = $O(n^2+n+e*n)$

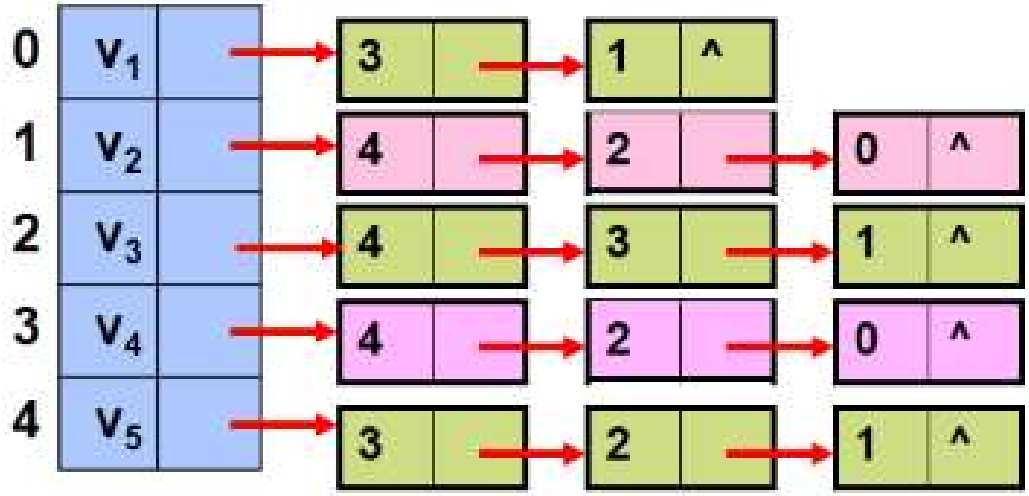
```
return OK;
}
```

二、 邻接表 (Adjacency List)

❑无向图的邻接表

为每个顶点建立一个单链表，第i个单链表中的结点表示与顶点v_i相邻的顶点





注：邻接表不唯一，因各个边结点的链入顺序是任意的。

二、 邻接表 (Adjacency List)

头结点:

data	firstarc

data: 结点的数据域，保存结点的数据值。

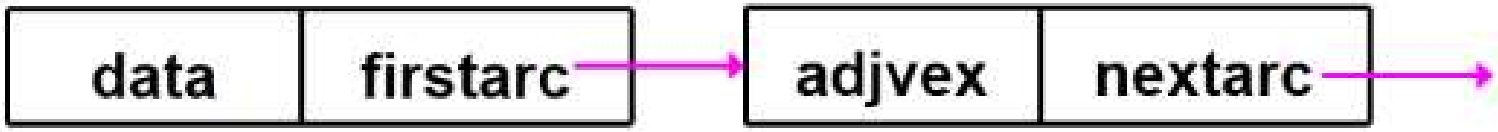
firstarc: 结点的指针域，给出自该结点出发的的第一条边的边结点的地址。

表结点:

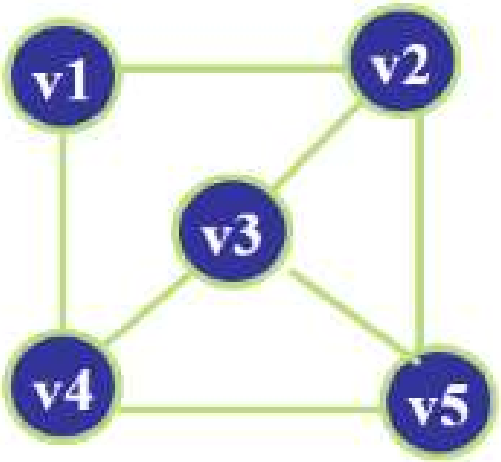
adjvex	nextarc

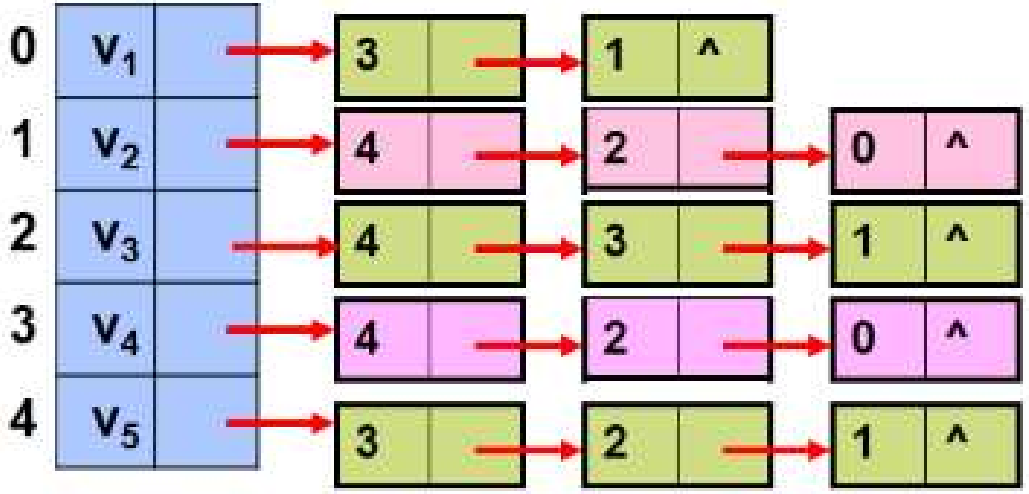
adjvex: 该边或弧所指向的顶点的位置.

nextarc: 指向下一条边或弧的指针.



二、 邻接表 (Adjacency List)





- 在无向图的邻接表中，如何求每个顶点的度？

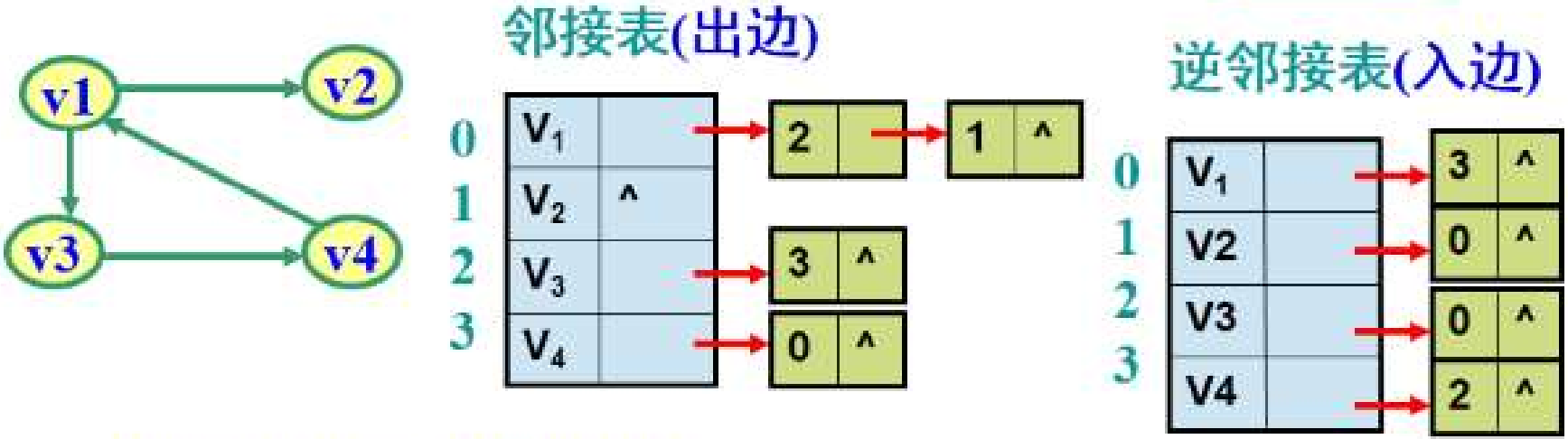
  第 i 个链表中的结点个数是顶点 i 的度

- 若顶点数为n，边数为e时，则在无向图中共需多少个结点？

  n个头结点，2e个表结点

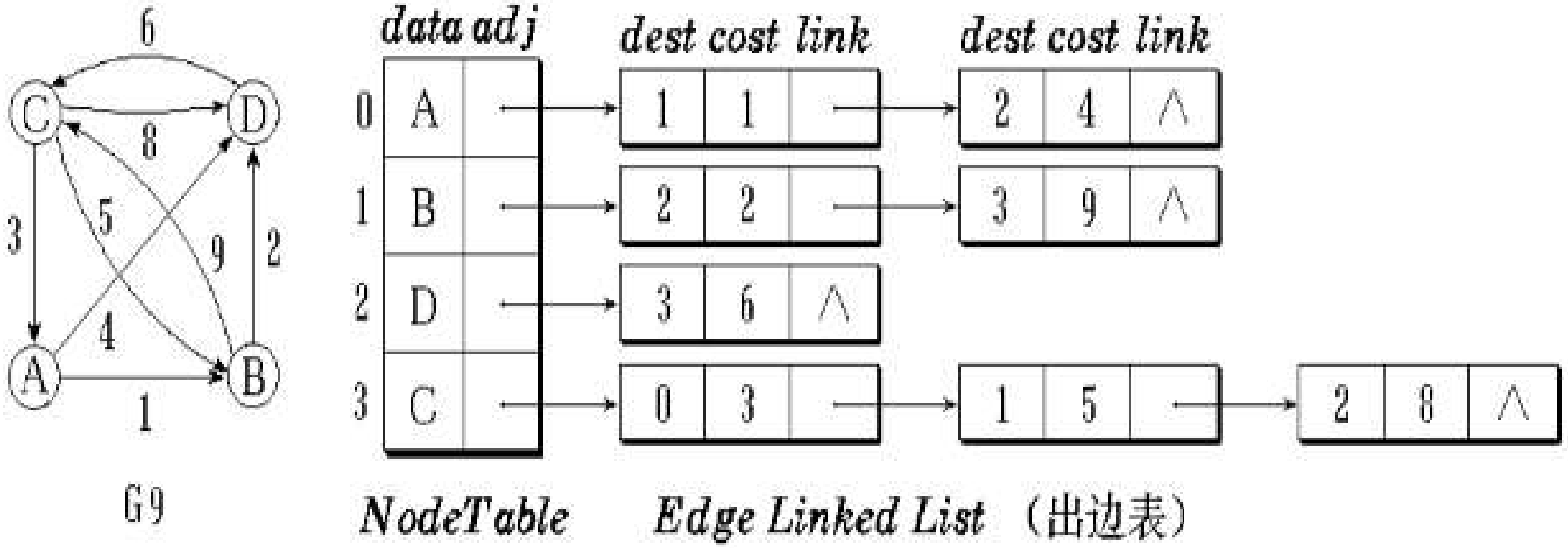
❑有向图的邻接表和逆邻接表

- 在有向图的邻接表中，第 i 个单链表链接的边都是顶点 i 发出的边。也叫做出边表。
- 在有向图的逆邻接表中，第 i 个单链表链接的边都是进入顶点 i 的边。也叫做入边表。



用邻接表表示有向图时，若不考虑逆邻接表，只需 n 个顶点结点，e 个边结点。

网络 (带权图) 的邻接表



带权图的边结点中保存该边上的权值 *cost*

邻接表表示的图的存储结构

```
//边结点定义
typedef struct ArcNode {
  int    adjvex;
// 该弧所指向的顶点的位置
  struct ArcNode  *nextarc;  // 指向下一条弧的指针
  InfoType   *info;   // 该弧相关信息的指针
} ArcNode;
```

邻接表表示的图的存储结构

```
//头结点定义
typedef struct VNode {
 VertexType  data;   // 顶点信息
 ArcNode    *firstarc; // 指向第一条依附该顶点的弧
 } VNode, AdjList[MAX_VERTEX_NUM];
```

邻接表表示的图的存储结构

```
//图的定义
typedef struct {
   AdjList  vertices;
  int      vexnum, arcnum;
  int      kind; // 图的种类标志
} ALGraph;
```

如何建立邻接表（以有向图为例）

- 算法思想：
  - 首先将邻接表表头数组初始化，vertex域初始化为i，firstarc初始化为NULL；
  - 然后对读入的每一条弧<i，j>，产生一个表结点，adjver域置为j，并将结点链接到邻接表的第i个链表中。

•算法如下：

```
Void GreatAdjList(ALGraph &G) {
  1.ArcNode   *p;
2.sacnf("%d%d",&n,&e); G.vexnum=n;  G.arcnum=e;
3.for(i=0; i<n; i++) {
 4.G.vertices[i].vertex=i;  G.vertices[i].firstarc=NULL;}
5.for(k=0; k<e; k++){
  6.scanf(i,j);
  7.p=(ArcNode *)malloc(sizeof(ArcNode));
  8.p->adjvex=j;  p->nextarc=G.vertices[i].firstarc;
  9.G.vertices[i].firstarc=p;}
}// GreatAdjList
```

讨论：邻接表与邻接矩阵的比较

1. **联系：**都可以用来存储有向图和无向图；邻接表中每个链表对应于邻接矩阵中的一行，链表中结点个数等于一行中非零元素的个数。
2. **区别：**

①邻接矩阵是顺序结构，邻接表是链式结构

②对于任一确定的无向图，邻接矩阵是唯一的（行列号与顶点编号一致），但邻接表不唯一（链接次序与顶点编号无关）。

③邻接矩阵的空间复杂度为$O(n^2)$，而邻接表的空间复杂度为$O(n+e)$。

④邻接矩阵多用于稠密图的存储；
而邻接表多用于稀疏图的存储