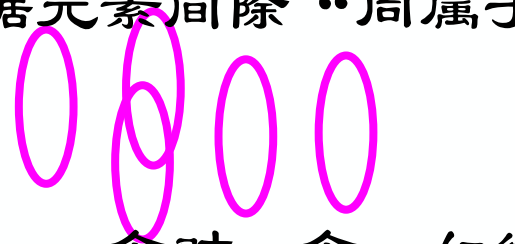
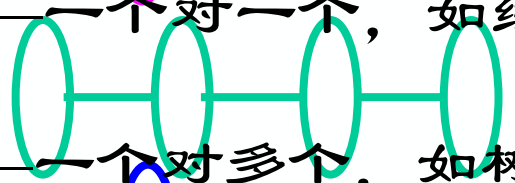
第6章 图

逻辑结构

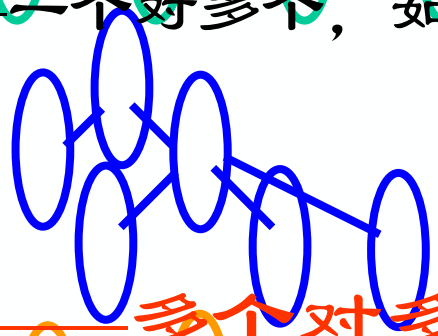
集合——数据元素间除“同属于一个集合”外，无其它关系



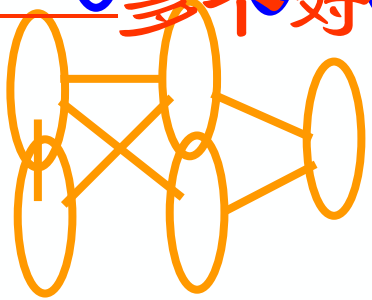
线性结构——一个对一个，如线性表、栈、队列



树形结构——一个对多个，如树

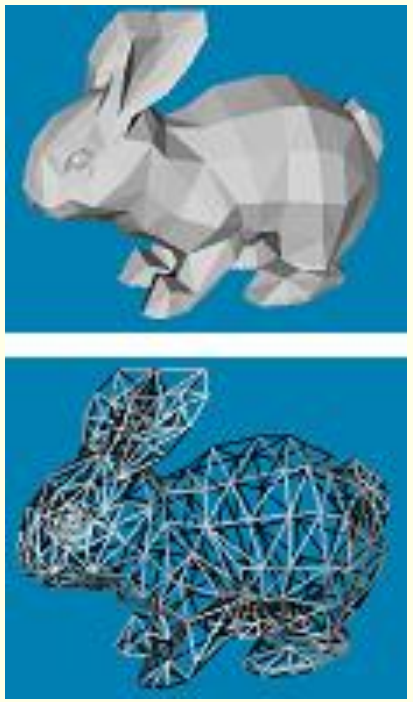


图形结构——多个对多个，如图



教学内容

6.1　图的定义和基本术语
6.2　案例引入
6.3　图的类型定义
6.4　图的存储结构
6.5　图的遍历
6.6　图的应用
6.7　案例分析与实现



教学目标

1.掌握：图的基本概念及相关术语和性质

2.熟练掌握：图的邻接矩阵和邻接表两种存储表示方法

3.熟练掌握：图的两种遍历方法DFS和BFS

4.熟练掌握：最短路算法（Dijkstra算法）

5.掌握：最小生成树的两种算法及拓扑排序算法的思想

6.1 图的定义和术语

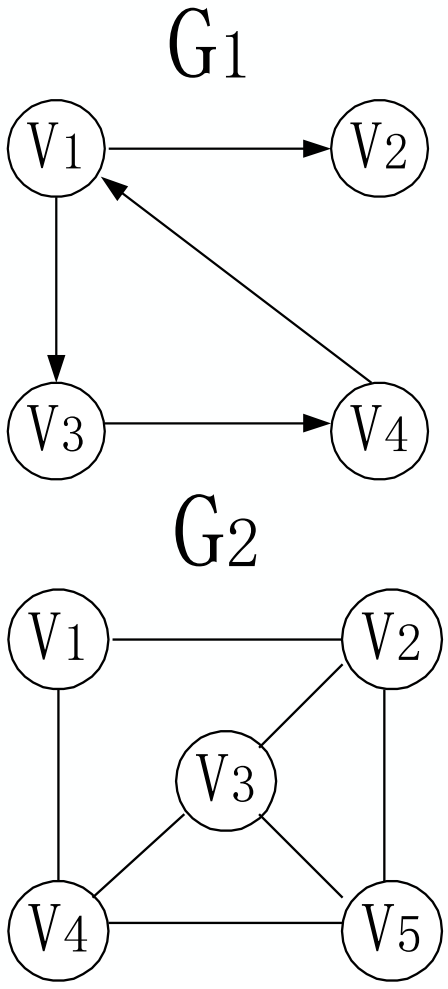
图： Graph=(V,E)

V：顶点(数据元素)的**有穷非空**集合；

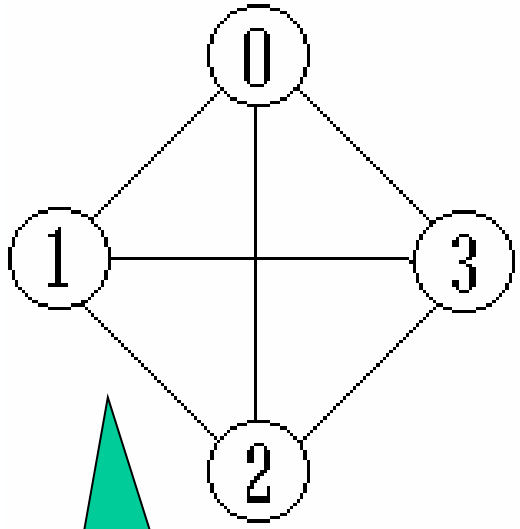
E：边的**有穷**集合。

无向图：每条边都是无方向的

有向图：每条边都是有方向的

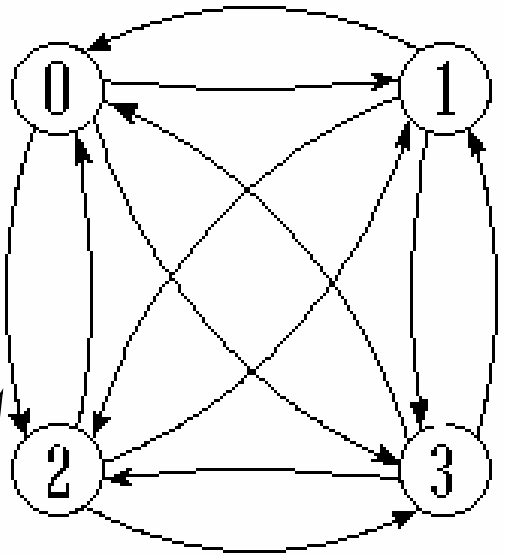


完全图：任意两个点都有一条边相连



无向完全图

$n(n-1)/2$ 条边



有向完全图

$n(n-1)$ 条边

稀疏图：有很少边或弧的图。

稠密图：有较多边或弧的图。

网：边/弧带权的图。

邻接：有边/弧相连的两个顶点之间的关系。

存在(v_i, v_j)，则称v_i和v_j互为**邻接点**；

存在$<v_i, v_j>$，则称v_i**邻接到**v_j，v_j**邻接于**v_i

关联(依附)：边/弧与顶点之间的关系。

存在(v_i, v_j)/ $<v_i, v_j>$，则称该边/弧关联于v_i和v_j

顶点的度：与该顶点相关联的边的数目，记为TD(v)

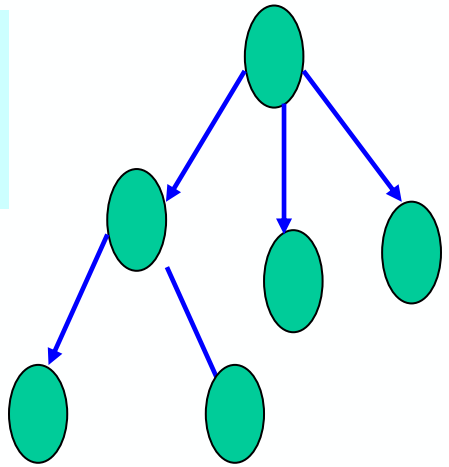
在**有向图**中，顶点的度等于该顶点的**入度**与**出度**之和。

顶点 v 的**入度**是以 v 为终点的有向边的条数，记作 ID(v)

顶点 v 的**出度**是以 v 为始点的有向边的条数，记作OD(v)

问：当有向图中仅1个顶点的入度为0,其余顶点的入度均为1，此时是何形状?

答：是树！而且是一棵有向树！



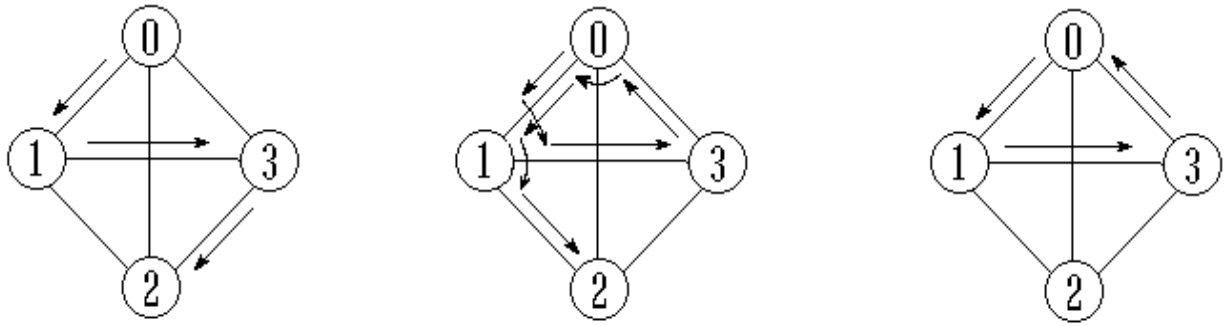
路径：接续的边构成的顶点序列。

路径长度：路径上边或弧的数目/权值之和。

回路(环)：第一个顶点和最后一个顶点相同的路径。

简单路径：除路径起点和终点可以相同外，其余顶点均不相同的路径。

简单回路(简单环)：除路径起点和终点相同外，其余顶点均不相同的路径。

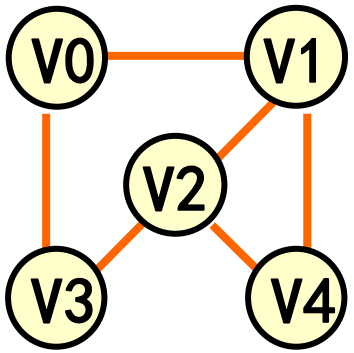


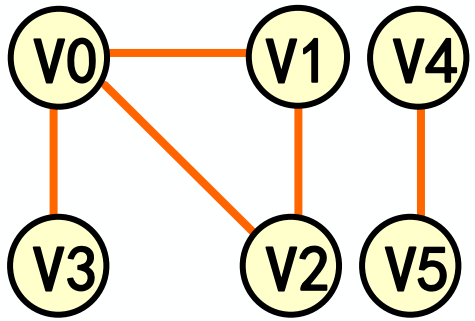
（a）简单路径　　（b）非简单路径　　（c） 回路

连通图（强连通图）

在无（有）向图G=(V, {E})中，若对任何两个顶点v、u 都存在从v 到 u 的路径，则称G是连通图（强连通图）。

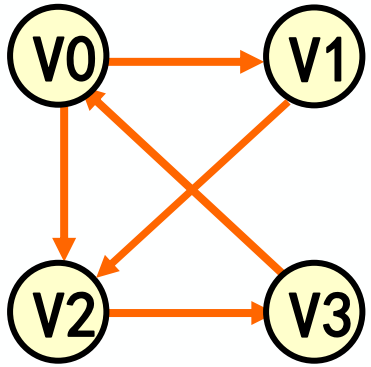
连通图

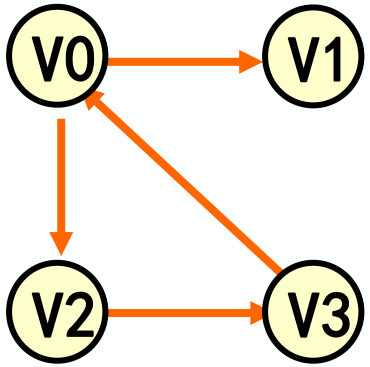




非连通图

强连通图





非强连通图

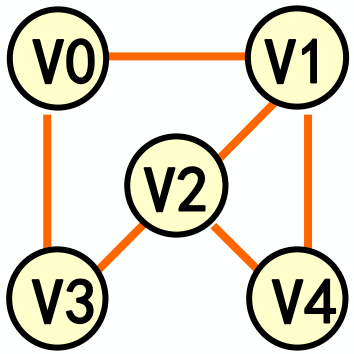
权与网

图中边或弧所具有的相关数称为权。表明从一个顶点到另一个顶点的距离或耗费。带权的图称为网。

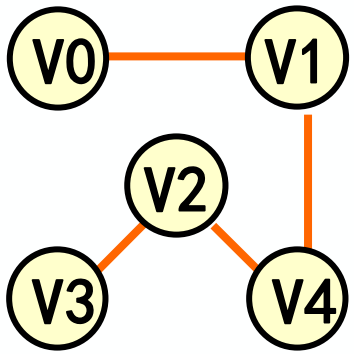
子图

设有两个图G=（V，{E}）、G1=（V1，{E1}），若V1⊆ V，E1 ⊆ E，则称 G1是G的子图。

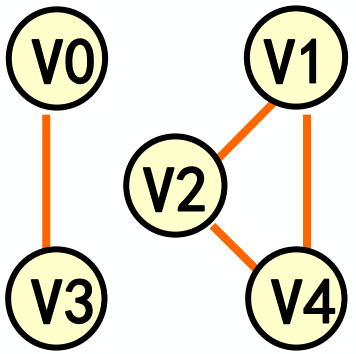
例:(b)、(c) 是 (a) 的子图



(a)



(b)

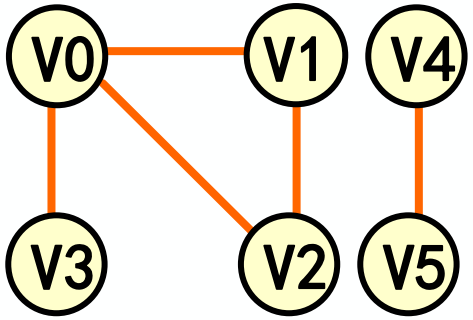


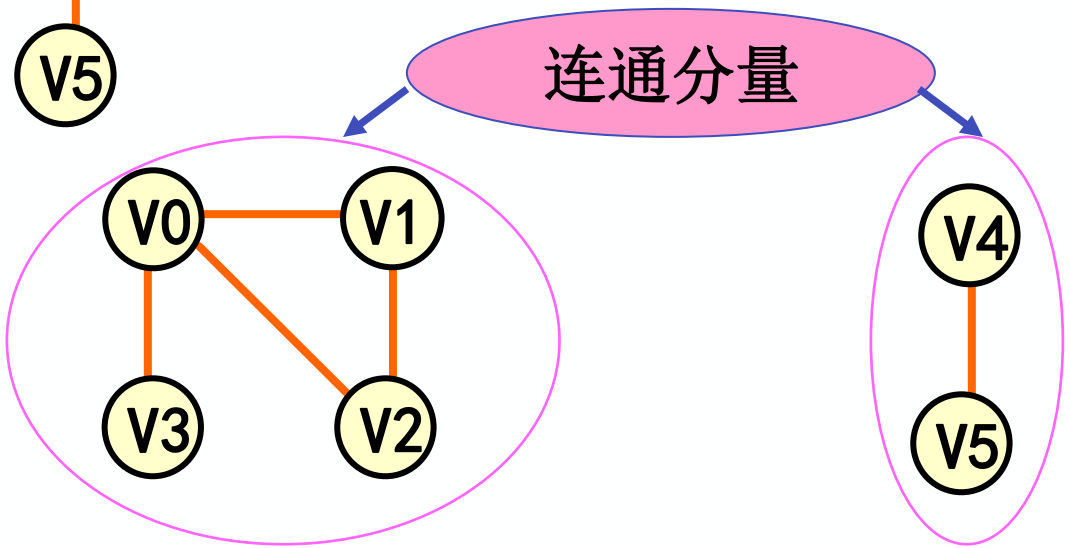
(c)

连通分量（强连通分量）

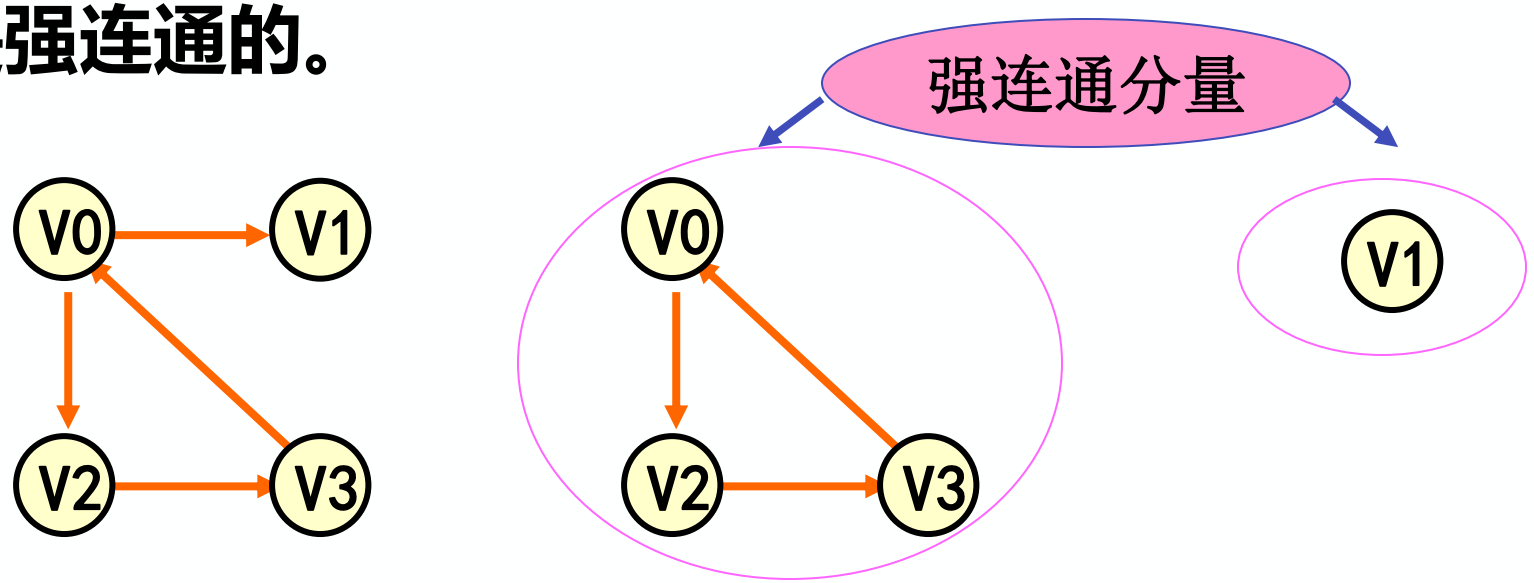
无向图G 的极大连通子图称为G的连通分量。
极大连通子图意思是：该子图是 G 连通子图，将G的任何不在该子图中的顶点加入，子图不再连通。

非连通图





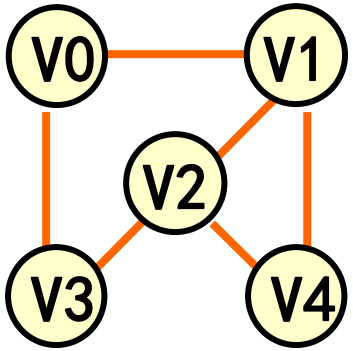
有向图G 的极大强连通子图称为G的强连通分量。极大强连通子图意思是：该子图是G的强连通子图，将D的任何不在该子图中的顶点加入，子图不再是强连通的。



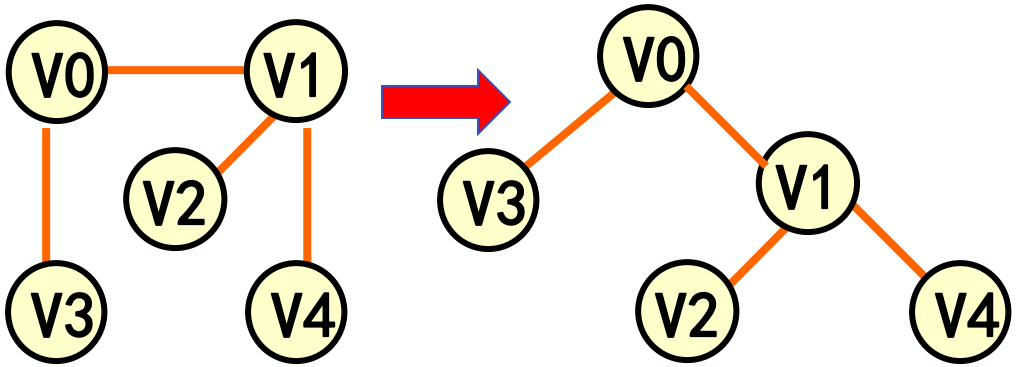
极小连通子图：该子图是G 的连通子图，在该子图中删除任何一条边，子图不再连通。

生成树：包含无向图G 所有顶点的极小连通子图。

生成森林：对非连通图，由各个连通分量的生成树的集合。



连通图 G1

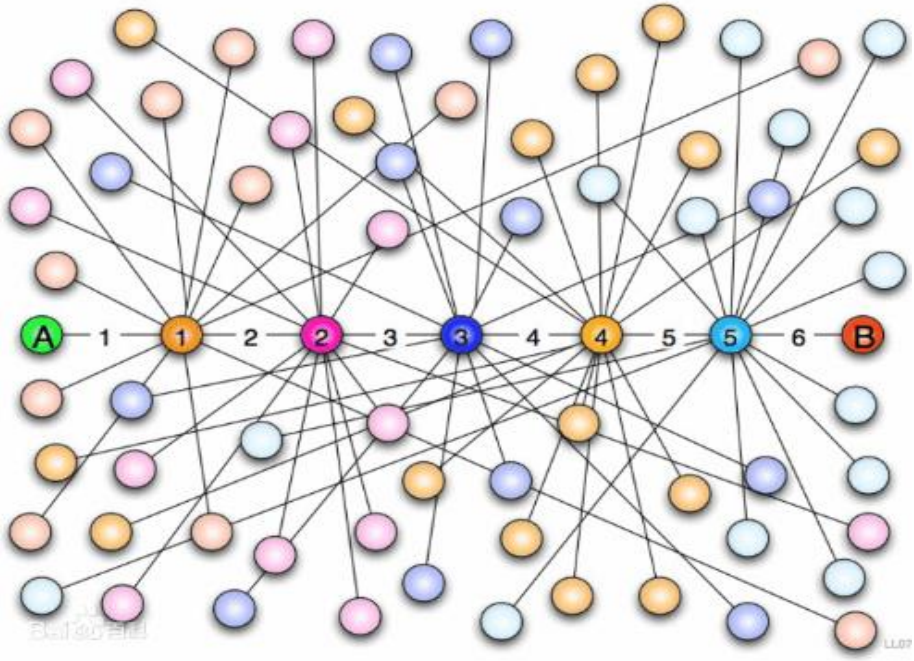


G1的生成树

6.2 案例引入

案例6.1： 六度空间理论

你和任何一个陌生人之间所间隔的人不会超过6个，也就是说，最多通过6个中间人你就能够认识任何一个陌生人。



6.3 图的类型定义

CreateGraph(&G,V,VR)

初始条件：V是图的顶点集，VR是图中弧的集合。

操作结果：按V和VR的定义构造图G。

DFSTraverse(G)

初始条件：图G存在。

操作结果：对图进行深度优先遍历。

BFSTraverse(G)

初始条件：图G存在。

操作结果：对图进行广度优先遍历。

6.4 图的存储结构

顺序存储结构： 数组表示法（邻接矩阵）

链式存储结构： 多重链表

- 邻接表
- 邻接多重表
- 十字链表

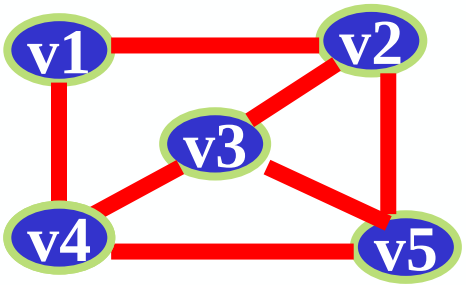
重点介绍： 邻接矩阵(数组)表示法
邻接表(链式)表示法

数组（邻接矩阵）表示法

- 建立一个顶点表（记录各个顶点信息）和一个邻接矩阵（表示各个顶点之间关系）。
- 设图 A = (*V*, *E*) 有 n 个顶点，则图的邻接矩阵是一个二维数组 A.Edge[n][n]，定义为：

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

无向图的邻接矩阵表示法



顶点表：（ v1 v2 v3 v4 v5 ）

邻接矩阵：

$$A.Edge = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

v1
v2
v3
v4
v5

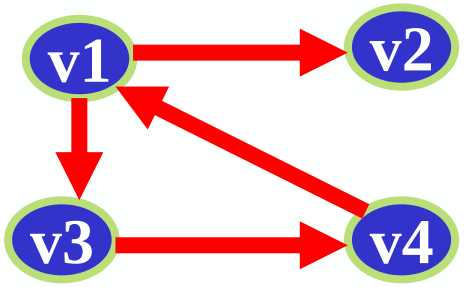
分析1：无向图的邻接矩阵是对称的；

分析2：顶点*i*的度＝第 *i*行 (列) 中1 的个数；

特别：完全图的邻接矩阵中，对角元素为0，其余1。

有向图的邻接矩阵表示法

A



顶点表：(v1 v2 v3 v4)

邻接矩阵：

$$A.Edge = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

v1
v2
v3
v4

注：在有向图的邻接矩阵中，
第i行含义：以结点v_i为尾的弧(即出度边)；
第i列含义：以结点v_i为头的弧(即入度边)。

分析1：有向图的邻接矩阵可能是不对称的。

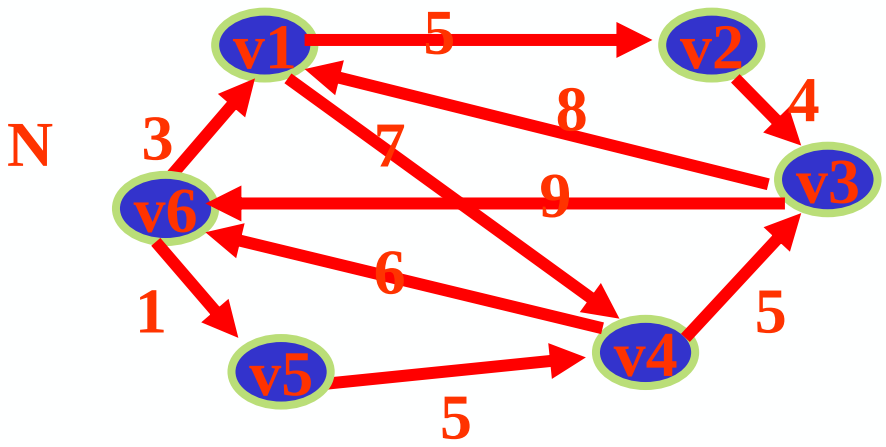
分析2：顶点的出度=第i行元素之和

顶点的入度=第i列元素之和

顶点的度=第i行元素之和+第i列元素之和

网（即有权图）的邻接矩阵表示法

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



顶点表：（ v1 v2 v3 v4 v5 v6 ）

邻接矩阵：

$$N.Edge=\begin{pmatrix} \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 5 & \infty & \infty & 6 \\ \infty & \infty & \infty & 5 & \infty & \infty \\ 3 & \infty & \infty & \infty & 1 & \infty \end{pmatrix}$$

邻接矩阵表示法的特点

优点： 容易实现图的操作，如：求某顶点的度、判断顶点之间是否有边、找顶点的邻接点等等。

缺点： n个顶点需要*n*n*个单元存储边;空间效率为$O(n^2)$。 对稀疏图而言尤其浪费空间。

邻接矩阵的存储表示

```
//用两个数组分别存储顶点表和邻接矩阵
#define MaxInt 32767                    //表示极大值，即∞
#define MVNum 100                       //最大顶点数
typedef char VerTexType;                //假设顶点的数据类型为字符型
typedef int ArcType;                    //假设边的权值类型为整型
typedef struct{
  VerTexType vexs[MVNum];                       //顶点表
  ArcType arcs[MVNum][MVNum];                   //邻接矩阵
  int vexnum,arcnum;                          //图的当前点数和边数
}AMGraph;
```

采用邻接矩阵表示法创建无向网

```
4   5
A B C D
A B 500
A C 200
A D 150
B C 400
C D 600
```

【算法思想】

(1) 输入总顶点数和总边数。

(2) 依次输入点的信息存入顶点表中。

(3) 初始化邻接矩阵，使每个权值初始化为极大值。

(4) 构造邻接矩阵。

【算法描述】

```
4  5
A B C D
A B 500
A C 200
A D 150
B C 400
C D 600
```

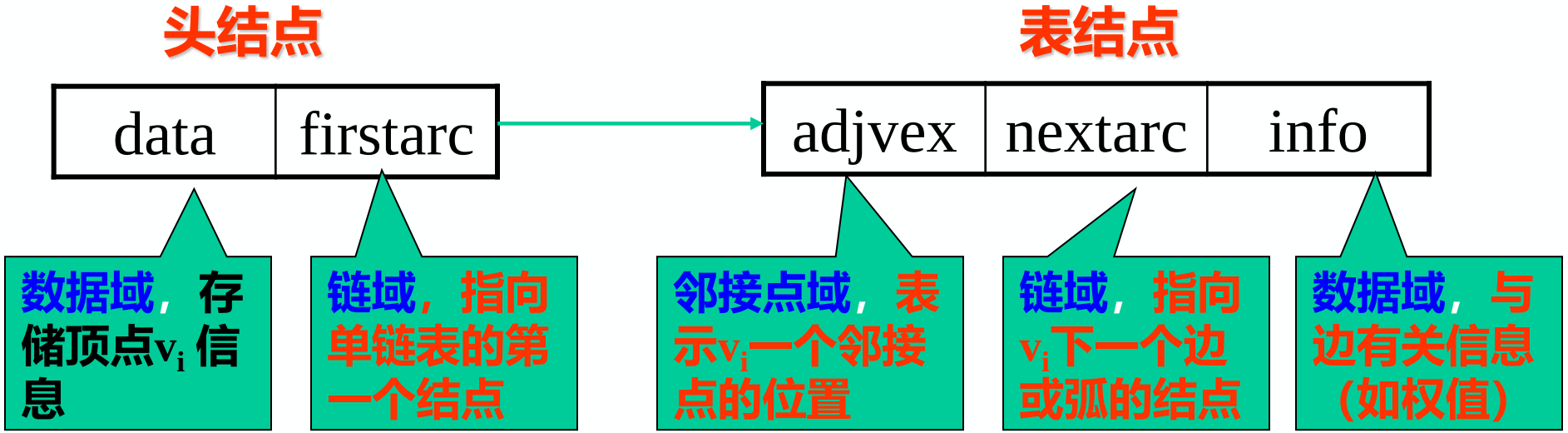
```
Status CreateUDN(AMGraph &G){
  //采用邻接矩阵表示法，创建无向网G
  cin>>G.vexnum>>G.arcnum;  //输入总顶点数，总边数
  for(i = 0; i<G.vexnum; ++i)
    cin>>G.vexs[i];                    //依次输入点的信息
  for(i = 0; i<G.vexnum;++i)           //初始化邻接矩阵，边的权值均置为极大值
    for(j = 0; j<G.vexnum;++j)
      G.arcs[i][j] = MaxInt;
  for(k = 0; k<G.arcnum;++k){                    //构造邻接矩阵
    cin>>v1>>v2>>w;                              //输入一条边依附的顶点及权值
    i = LocateVex(G, v1);  j = LocateVex(G, v2);  //确定v1和v2在G中的位置
    G.arcs[i][j] = w; //边<v1, v2>的权值置为w
    G.arcs[j][i] = G.arcs[i][j];                 //置<v1, v2>的对称边<v2, v1>的权值为w
  }//for
  return OK;
}//CreateUDN
```

```
4    5
A B C D
A B 500
A C 200
A D 150
B C 400
C D 600
```

```
int LocateVex(MGraph G,VertexType u)
{//存在则返回u在顶点表中的下标;否则返回-1
  int i;
  for(i=0;i<G.vexnum;++i)
    if(u==G.vexs[i])
      return i;
  return -1;
}
```

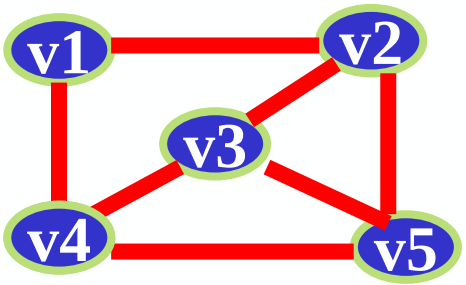
邻接表（链式）表示法

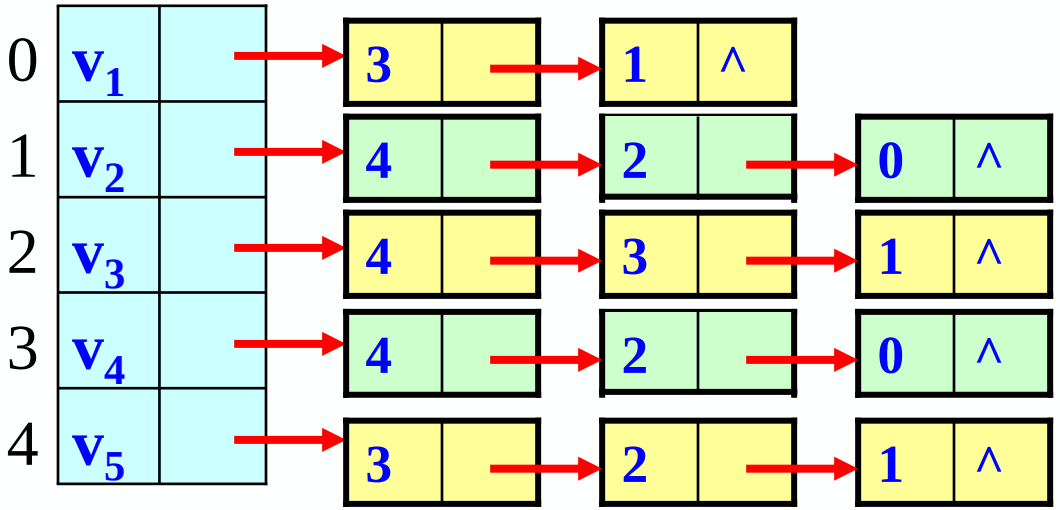
- 对每个顶点v_i建立一个**单链表**，把与v_i有关联的**边的信息链接**起来，每个结点设为3个域；



- 每个单链表有一个**头结点**（设为2个域），存v_i信息；
- 每个单链表的**头结点另外用顺序存储**结构存储。

无向图的邻接表表示



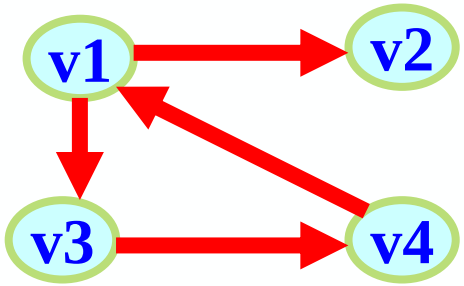


注：邻接表不唯一，因各个边结点的链入顺序是任意的

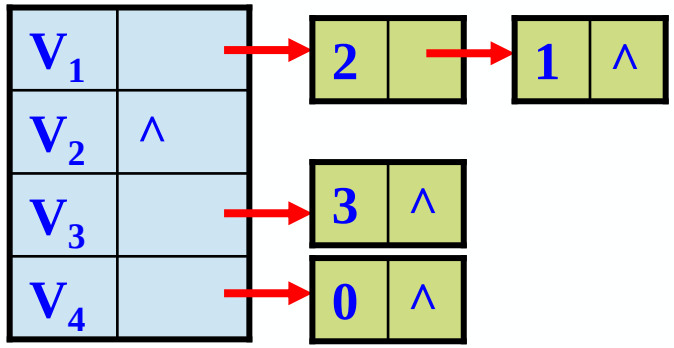
空间效率为O(n+2e)。
若是稀疏图($e<<n^2$)，比邻接矩阵表示法$O(n^2)$省空间。

TD(Vi)=单链表中链接的结点个数

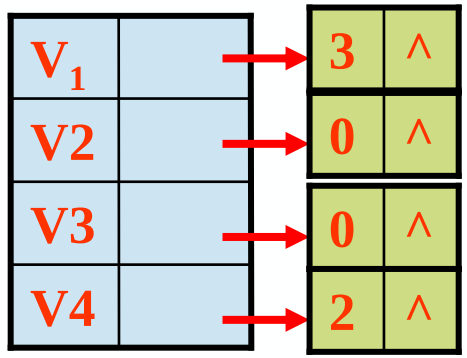
有向图的邻接表表示



邻接表(出边)



逆邻接表(入边)



空间效率为O(n+e)

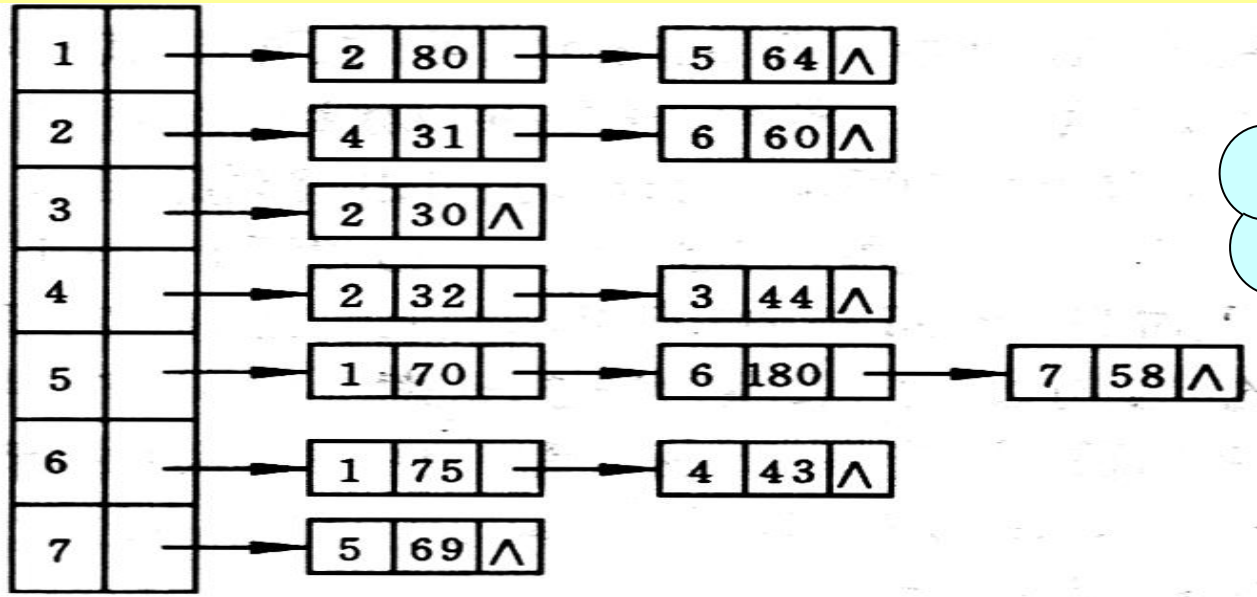
出度
入度
度:

OD(Vi) = 单链出边表中链接的结点数

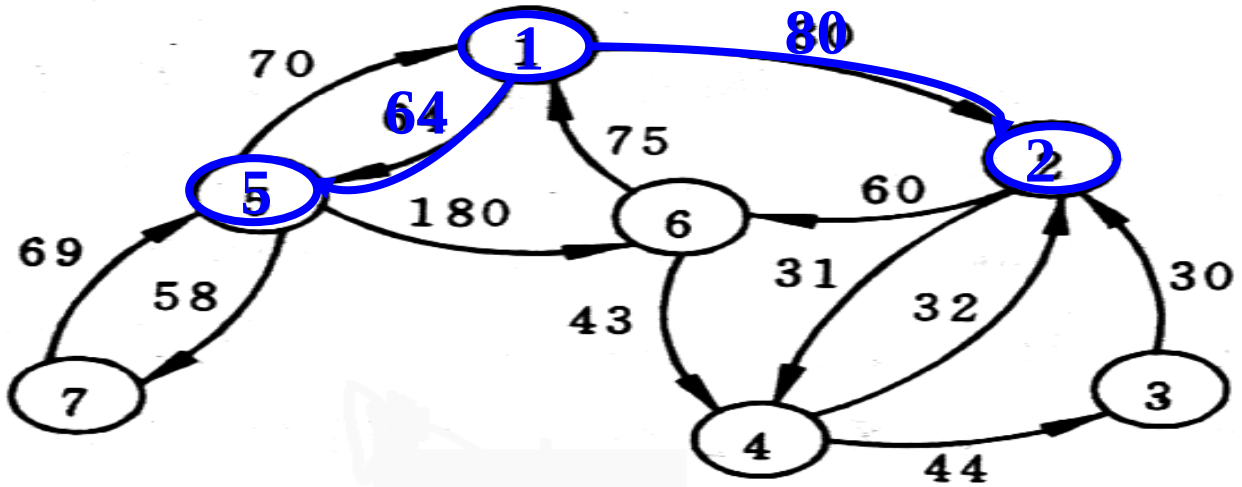
ID(Vi) = 邻接点域为Vi的弧个数

TD(Vi) = OD(Vi) + I D(Vi)

已知某网的邻接（出边）表，请画出该网络。



当邻接表的存储结构形成后，图便唯一确定！



邻接表的存储表示

```
#define MVNum 100                          //最大顶点数
typedef struct ArcNode{                    //边结点
    int adjvex;                            //该边所指向的顶点的位置
    struct ArcNode * nextarc;              //指向下一条边的指针
    OtherInfo info;                        //和边相关的信息
}ArcNode;
typedef struct VNode{
    VerTexType data;                //顶点信息
    ArcNode * firstarc;             //指向第一条依附该顶点的边的指针
}VNode, AdjList[MVNum];                    //AdjList表示邻接表类型
typedef struct{
    AdjList vertices;                      //邻接表
    int vexnum, arcnum;                    //图的当前顶点数和边数
}ALGraph;
```