离散数学（2）
—图的代数表示

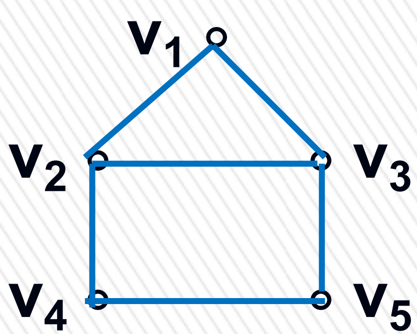
周旻
清华大学软件学院
软件工程与系统研究所

2024年4月7日
Sunday

图的代数表示

图的代数表示

- 为什么需要代数表示
  - 点线图 → 人容易理解的形式；
  - 计算机容易理解 → 清晰、明确的结构；
  - 建立图的代数表示，才可以
    - 利用代数方法解决图论问题；
    - 利用计算机的计算能力，完成图论算法的计算，解决实际问题。
- 什么是“图的代数表示”？
  - 简言之，将图表示成一种准确的、容易理解和使用的代数形式（如矩阵、表）；



$$A(G)=\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{1n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

G=(V, E)

- 如何表示结点和边？
- 如何使用？
- 有何优缺点？

图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

图的代数表示：邻接矩阵

邻接矩阵

表示结点与结点之间的邻接关系（不考虑边的权值）．

定义1.2.1: 设$G=<V,E>$是个简单图, $V = \{v_1, v_2, \ldots, v_n\}$. G的邻接矩阵是一个$n\times n$的矩阵$A=(a_{ij})$

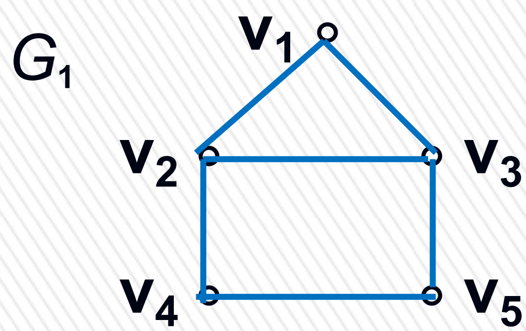
$$A(G) = \begin{array}{c} \\ v_1 \\ v_2 \\ \cdots \\ v_n \end{array} \begin{array}{c} \begin{array}{cccc} v_1, & v_2, & \ldots, & v_n \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{1n} \\ \ldots & \ldots & \ldots & \ldots \\ a_{n1} & a_{n2} & \ldots & a_{nn} \end{pmatrix} \end{array}$$

其中:

$$a_{ij}=\begin{cases} 1, & v_i\text{与}v_j\text{邻接，即}(v_i,v_j)\in E \\ 0, & \text{其他。} \end{cases}$$

邻接矩阵示例

例**1.2.1**：给定无向图G_1如图所示：



无向图

邻接矩阵是对称阵

每行**1**的个数=每列**1**的个数=对应结点的度

$A(G_1)=$

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	0
v_2	1	0	1	1	0
v_3	1	1	0	0	1
v_4	0	1	0	0	1
v_5	0	0	1	1	0

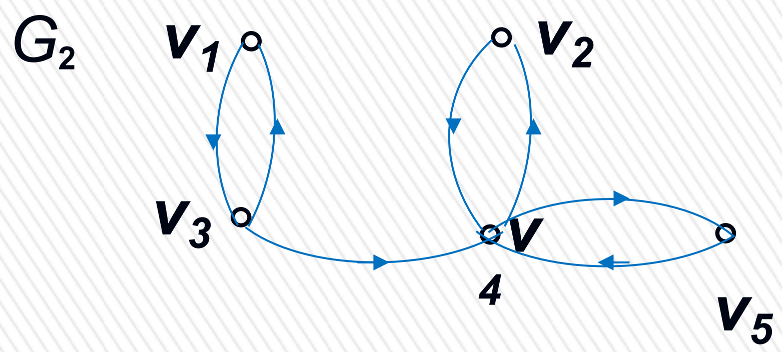
	v_1	v_2	v_3	v_4	v_5	
v_1	0	1	1	0	0	
v_2	1	0	1	1	0	
v_3	1	1	0	0	1	$sum=3$
v_4	0	1	0	0	1	
v_5	0	0	1	1	0	

$sum=3$

$d(v_3)=3$

邻接矩阵示例

例**1.2.1**：给定有向图G_2如图所示：



有向图

每行1的个数=对应结点的出度

每列1的个数=对应结点的入度

$A(G_2)=$

	v_1	v_2	v_3	v_4	v_5
v_1	0	0	1	0	0
v_2	0	0	0	1	0
v_3	1	0	0	1	0
v_4	0	1	0	0	1
v_5	0	0	0	1	0

	v_1	v_2	v_3	v_4	v_5	
v_1	0	0	1	0	0	
v_2	0	0	0	1	0	
v_3	1	0	0	1	0	$sum=2=d^+(v_3)$
v_4	0	1	0	0	1	
v_5	0	0	0	1	0	

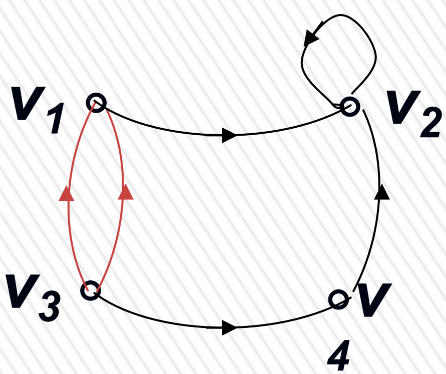
$sum=1=d^-(v_3)$

邻接矩阵的性质

如何表示自环: $a_{ii}=1$

如何表示重边：扩展 **0-1** 矩阵

$$a_{ij} = \text{从 } v_i \text{ 到 } v_j \text{ 的边的数量}$$



	v_1	v_2	v_3	v_4
v_1	0	1	0	0
v_2	0	1	0	0
v_3	2	0	0	1
v_4	0	1	0	0

能表示多重邻接关系，无法进一步表示"权值"等

邻接矩阵的性质

- 如何简单的实现邻接矩阵？
  - 二维数组？占用$O(n^2)$存储空间
- 如何检查两个结点是否邻接？
  - 很便捷：检查a_{ij}是否为0；
- 如何枚举出给定结点的所有直接后继/前驱？
  - 遍历整行得到继结点，遍历整列得到前驱点；
  - 复杂度$O(n)$；
- 思考：微信的朋友关系也是一个图，可以这么保存吗？

思考

如果A是图G的邻接矩阵，A^{T}（即A的转置）代表什么？

思考

如果A是图G的邻接矩阵，A^{T}（即A的转置）代表什么？

观察：$(v_i, v_j) \in E' \Leftrightarrow (A^{\mathrm{T}})_{ij}=1 \Leftrightarrow (A)_{ji}=1 \Leftrightarrow (v_j, v_i) \in E$

- 其中E'是A^{T}对应的图

- 转置图：
  - 给定 $G=(V,E)$，其转置图是$G'=(V,E')$，其中：

$$E' = \{(v, w) \mid (w, v) \in E\}$$

  - 转置图的邻接矩阵 = 原图邻接矩阵的转置

思考

如果A是图G的邻接矩阵，A^k代表什么？

思考

邻接矩阵的计算意义：$A, A^2, A^3, \ldots$

考虑 A^2：

$$(A^2)_{i,j} = \sum_k a_{i,k} \times a_{k,j}$$

对于每一个k：

- a_{ik}表示从v_i到v_k是否有一条边，a_{kj}表示从v_k到v_j是否有一条边；
- 当且仅当$a_{ik}=a_{kj}=1$时，$a_{ik}*a_{ik}=1$，理解为：从v_i经过v_k可以到达v_j。
- $(A^2)_{ij}$表示从v_i途径1个节点到v_j有多少种走法；

思考

$(A)_{ij}$ 表示从v_i直接到v_j有多少种走法（考虑重边）；

$(A^2)_{ij}$表示从v_i途径1个节点到v_j有多少种走法；

相应的：

$(A^3)_{ij}$表示从v_i途径2个节点到v_j有多少种走法；

$(A^k)_{ij}$表示从v_i途径$(k\text{-}1)$个节点到v_j有多少种走法；

后面课程详细了解

图的代数表示：权矩阵

- 图的权重通常具有重要意义：道路的长度、网络连接的带宽、延迟等。
- 赋权图用权矩阵表示：

$G = (V, E)$, 对节点编号 $V = \{v_1, v_2, \ldots, v_n\}$.

$$A(G) = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{1n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix}$$

$$a_{ij} = \begin{cases} w_{ij}, & (v_i, v_j) \in E \\ 0, & \text{其他} \end{cases}$$

权矩阵的性质

权矩阵（跟邻接矩阵相比）

- 用法跟邻接矩阵相似
  - 检查两个节点是否邻接：检查a_{ij}是否为**0**（注意不能与权值冲突）；
  - 枚举一个节点的所有直接后继/前驱：遍历行/列；
- 如何表示自环？对角线元素（a_{ii}）
- 如何表示重边？
  - 重边的例子：物流站点之间的多种运输方式；
  - 无法直接表示重边

图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

关联矩阵

- 无向图的关联矩阵

  表示结点与边之间的关联关系.

设无向图$G=<V,E>$, $V=\{v_1, v_2, \ldots, v_n\}$, $E=\{e_1, e_2, \ldots, e_m\}$.

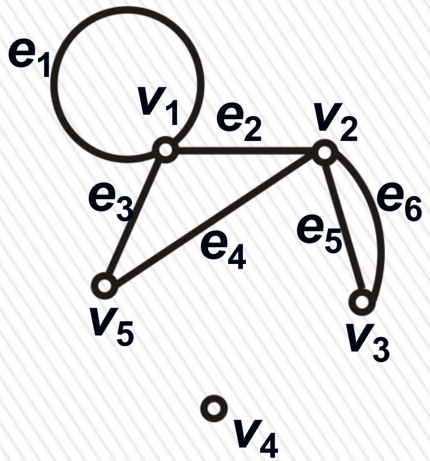
令b_{ij}为v_i与e_j的关联次数, 称$B_{n\times m}$为G的关联矩阵,

记为$B(G)$. b_{ij}的可能取值为:{0, 1, 2}

其中：<u>0表示无关联</u>，<u>1表示有关联</u>，<u>2表示自环</u>

$B(G)=$

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	2	1	1	0	0	0
v_2	0	1	0	1	1	1
v_3	0	0	0	0	1	1
v_4	0	0	0	0	0	0
v_5	0	0	1	1	0	0

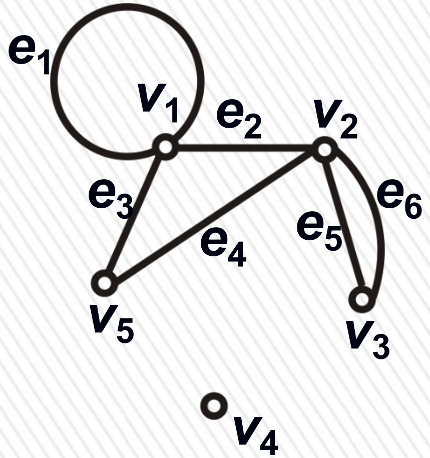


关联矩阵性质

- 无向图关联矩阵的性质

$B(G)=$

	e_1	e_2	e_3	e_4	e_5	e_6
v_1	2	1	1	0	0	0
v_2	0	1	0	1	1	1
v_3	0	0	0	0	1	1
v_4	0	0	0	0	0	0
v_5	0	0	1	1	0	0



从列来看→ (1) $\sum_{i=1}^{n} b_{ij} = 2, \quad j = 1,2,\ldots,m$

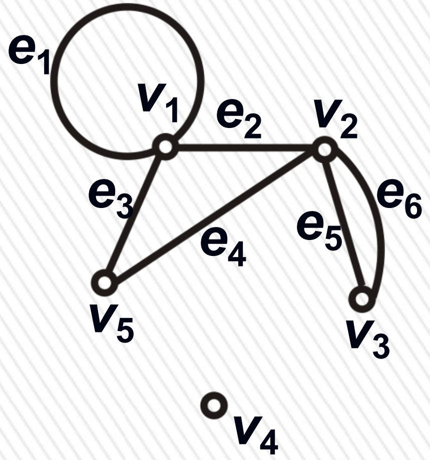
从行来看→ (2) $\sum_{j=1}^{m} b_{ij} = d(v_i), \quad i = 1,2,\ldots,n$

握手定理→ (3) $\sum_{i,j} b_{ij} = 2m$

思考

- 无向图关联矩阵的性质

$$B(G)=\begin{array}{c|cccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 \\ \hline v_1 & 2 & 1 & 1 & 0 & 0 & 0 \\ v_2 & 0 & 1 & 0 & 1 & 1 & 1 \\ v_3 & 0 & 0 & 0 & 0 & 1 & 1 \\ v_4 & 0 & 0 & 0 & 0 & 0 & 0 \\ v_5 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$



以下分别对应什么情况？

(1) 某列某元素为2, 其余为 → 自环（e_1）

(2) 相同的两列 → 重边（e_5 和 e_6）

(3) 某行全为0 → 孤立点（v_4）

关联矩阵

- 有向图（无环）的关联矩阵

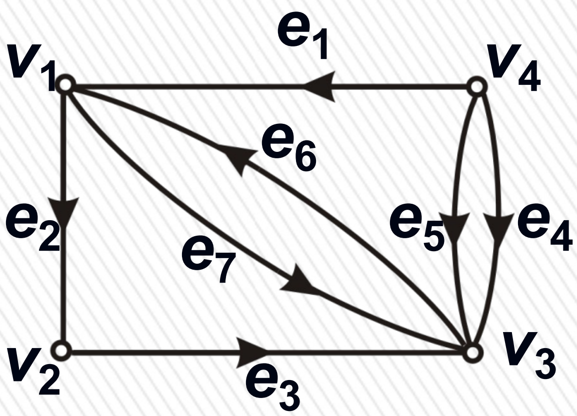
设无环有向图$D=<V,E>$, $V=\{v_1, v_2, \ldots, v_n\}$, $E=\{e_1, e_2, \ldots, e_m\}$.
令$B(D)=(b_{ij})_{n\times m}$满足：

$$b_{ij}=\begin{cases} 1, & v_i\text{为}e_j\text{的始点} \\ 0, & v_i\text{与}e_j\text{不关联} \\ -1, & v_i\text{为}e_j\text{的终点} \end{cases}$$

则称$B_{n\times m}$为D的关联矩阵

关联矩阵

- 有向图的关联矩阵



$B(D)=$

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1	-1	1	0	0	0	-1	1
v_2	0	-1	1	0	0	0	0
v_3	0	0	-1	-1	-1	1	-1
v_4	1	0	0	1	1	0	0

性质：(1) $\sum_{i=1}^{n} b_{ij} = 0, \qquad j = 1,2,\ldots,m$

(2) 第i行1的个数等于$d^+(v)$, 第i行-1的个数等于$d^-(v)$

(3) e_j与e_k是重边 $\Leftrightarrow$ 第j列与第k列相同

(4) 无法表达自环

思考

- 如何判断v_i与v_j是否邻接？
  - 查找是否存在某列，第 i 行为1，第 j 行为-1
  - 需要遍历所有边？O(m) v.s 邻接矩阵 O(1)
- 如何得到 v_i 的所有直接后继？
  - 先找到第 i 行的1，再找到对应的-1
  - 需要遍历所有边？O(m) v.s 邻接矩阵 O(n)
- 需要多少存储空间？
  - 关联矩阵大小$n*m$
  - 与邻接矩阵O(n^2)相比如何？
- 比邻接矩阵更复杂，为何要有关联矩阵？
  - 代数性质对于图论讨论很重要

各种矩阵表示的缺点

$$A(G)=\begin{array}{c|ccccc} & v_1 & v_2 & v_3 & v_4 & v_5 \\ \hline v_1 & 0 & 0 & 1 & 0 & 0 \\ v_2 & 0 & 0 & 0 & 1 & 0 \\ v_3 & 1 & 0 & 0 & 1 & 0 \\ v_4 & 0 & 1 & 1 & 0 & 1 \\ v_5 & 0 & 0 & 0 & 1 & 0 \end{array}$$

$$B(D)=\begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & -1 & 1 & 0 & 0 & 0 & -1 & 1 \\ v_2 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 0 & -1 & -1 & -1 & 1 & -1 \\ v_4 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$

- 空间需求大，无法利用图的稀疏性
  - 真实的图中，大量存在稀疏性：
    - 考虑社交APP中的好友关系构成的图；
    - 全球80亿人，微信好友上限5000？如果用邻接矩阵，非0元素比例不到10万分之一；如果用边列表，非0元素比例40亿分支一；
  - 使用效率低：查询后继节点需要遍历全图节点或者边
- 解决方法：矩阵 → 表
  - 边列表、正向表、逆向表、邻接表等

图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

边列表

- 边列表

$$B(D)=\begin{array}{c|ccccccc} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & -1 & 1 & 0 & 0 & 0 & -1 & 1 \\ v_2 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 0 & -1 & -1 & -1 & 1 & -1 \\ v_4 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \end{array}$$

观察：关联矩阵每列只有**2**个非**0**值,其余**(n-2)**个都是**0**

→ 对关联矩阵的列进行压缩

→ 直接用边的起始、终止节点编号来表示

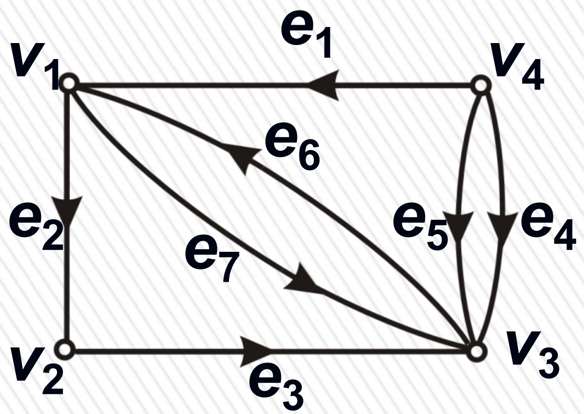
边列表有两个 m 维向量 A 和 B 组成，分别存放起始、终止节点的编号：

$$\text{若}\, e_k=(v_i,v_j)\text{，则}\, A(k)=i\text{，}\, B(k)=j$$

边列表

- 边列表

$$B(D)=\begin{array}{c|ccccccc|} & e_1 & e_2 & e_3 & e_4 & e_5 & e_6 & e_7 \\ \hline v_1 & -1 & 1 & 0 & 0 & 0 & -1 & 1 \\ v_2 & 0 & -1 & 1 & 0 & 0 & 0 & 0 \\ v_3 & 0 & 0 & -1 & -1 & -1 & 1 & -1 \\ v_4 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ \hline \end{array}$$



A:(4, 1, 2, 4, 4, 3, 1)

B:(1, 2, 3, 3, 3, 1, 3)

边列表的特点：

(1) 蓦然回首：图定义的直接表示，自环、重边均支持；

(2) 对赋权图，再用1个m维向量W存放权，$W(k)=w_k$

(3) 占用空间：O(n*m) 减少为 O(m)；

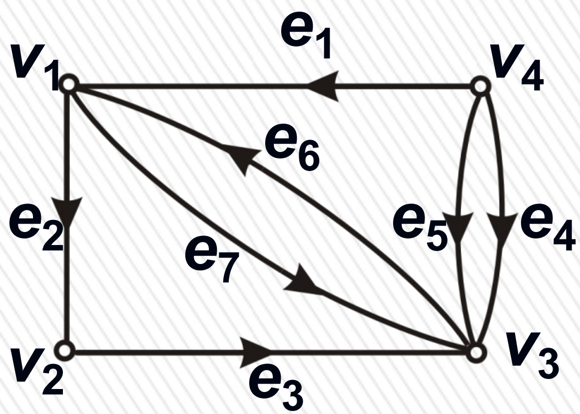
(4) 但是，怎么用呢？如何枚举直接“后继”和“前驱”？

边列表

- 边列表

$B(D)=$

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1	-1	1	0	0	0	-1	1
v_2	0	-1	1	0	0	0	0
v_3	0	0	-1	-1	-1	1	-1
v_4	1	0	0	1	1	0	0



A:(4, 1, 2, 4, 4, 3, 1) → A:(1, 1, 2, 3, 4, 4, 4)

B:(1, 2, 3, 3, 3, 1, 3) → B:(2, 3, 3, 1, 1, 3, 3)

以查询“直接后继”为例：

(1) 遍历所有的边 → O(m) → 低效

解决方法：排序

(1) 查询“直接后继”效率提升 → O(max(log$_2m$, d(v)))

相同起始节点的边，聚集在同一个连续区间 → 起始节点重复

同个起始节点只记录一次 → 在A上面建立“索引” → 正向表

图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

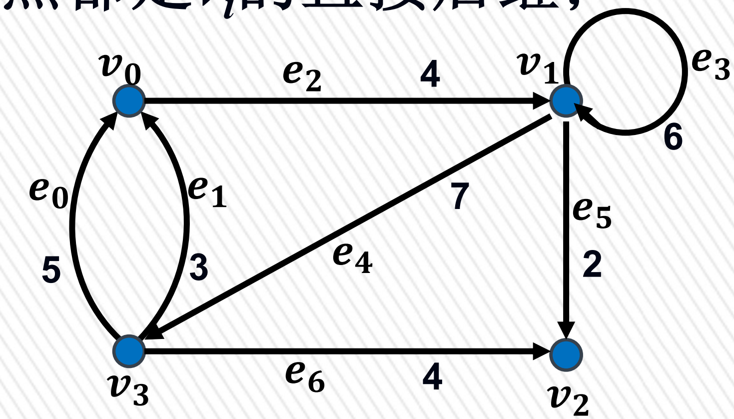
正向表

- 正向表

将节点从0开始编号，将边列表按照起始结点排序
每一个结点的直接后继对应B中的下标区间：

- A长度n+1，用A(i)表示结点v_i对应区间的起始下标；
- B中下标区间[$A(i)$, $A(i+1)$)的节点都是v_i的直接后继；

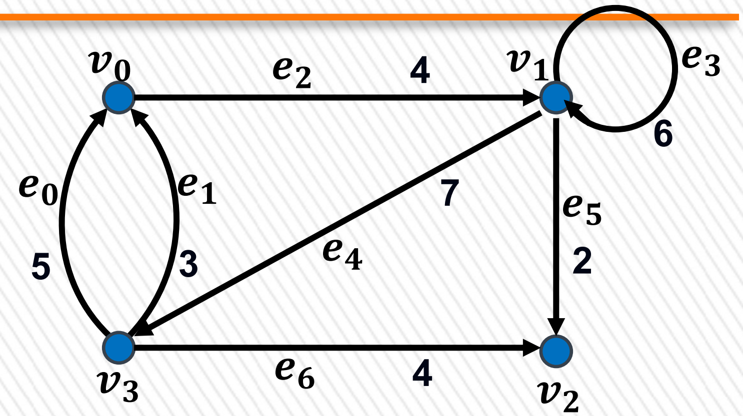
A	0	1	4	4	7		
A_0	0	1	1	1	3	3	3
B	1	1	2	3	0	0	2
W	4	6	2	7	3	5	4



边列表：A_0 和 B
正向表：A 和 B
如需权值：W

正向表

- 正向表



正向表中存在如下关系

1. $d^+(v_i) = \mathrm{A}(i+1) - \mathrm{A}(i)$
2. $\mathrm{A}(i) = \sum_{j=0}^{i-1} d^+(v_j)$
3. 占用空间从2m减少到n+m，是否有意义？
4. 查询“后继”很方便

A	0	1	4	4	7		
B	1	1	2	3	0	0	2
W	4	6	2	7	3	5	4

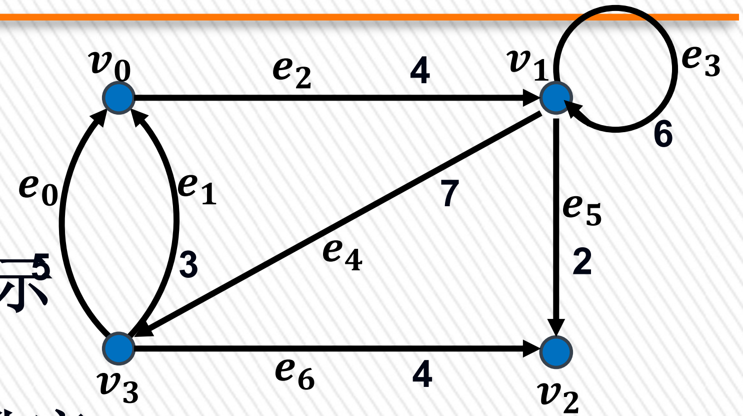
逆向表

- 逆向表

正向表上怎么查询“前驱”？

逆向表：正向表的“对偶”表示

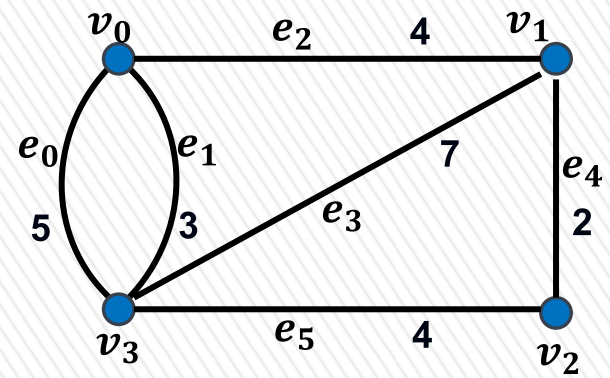
- 考虑正向边 → 考虑逆向边
- 起始节点排序 → 终止节点排序
- 方便查询“前驱”，不方便查“后继”，鱼和熊掌



A	0	2	4	6	7		
B	3	3	0	1	1	3	1
W	5	3	4	6	2	4	7

无向图的正向、逆向表

- 无向图正向表与逆向表相同
  - 思考题：想想为什么？



A	0	3	6	8	12

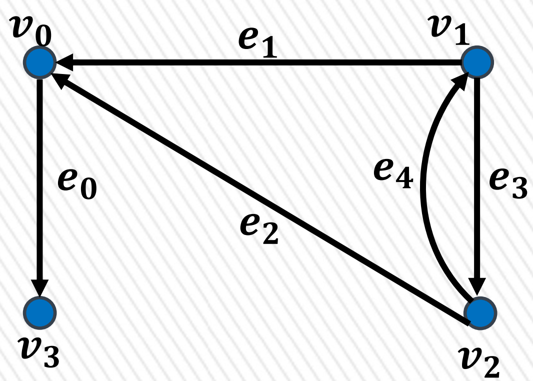
B	3	3	1	0	2	3	1	3	0	0	1	2
W	5	3	4	4	2	7	2	4	5	3	7	4

图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

邻接表



$A(G)=$

	v_0	v_1	v_2	v_3
v_0	0	0	0	1
v_1	1	0	1	0
v_2	1	1	0	0
v_3	0	0	0	0

	v_0	v_1	v_2	v_3
v_0	0	0	0	1
v_1	1	0	1	0
v_2	1	1	0	0
v_3	0	0	0	0

- 回顾邻接矩阵
  - a_{ij}=1 当且仅当存在边 (v_i, v_j)；
  - 表格中的每个1，对应了一条边；
- 典型场景：获取v_i的所有直接后继结点
  - 遍历第i行，检查其中的1；
  - 时间复杂度O(n)：尤其对于稀疏图而言，效率低；
- 解决思路：
  - 利用稀疏性，把每行中的1串接起来（跳过0）；
  - 怎么做？

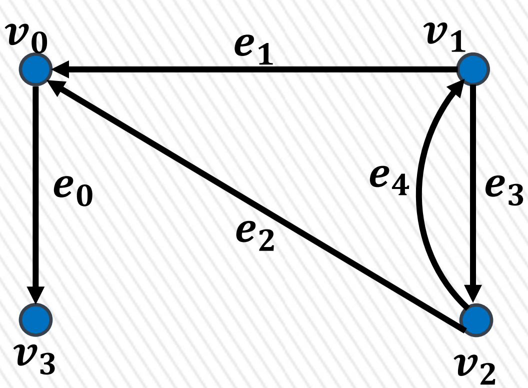
邻接表

■ 邻接表

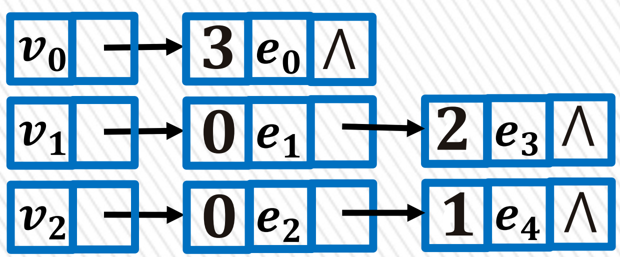
回顾：单链表

- 邻接矩阵重的每一行（结点v_i）对应一个单链表
- 表结点由三个域(*dst*, *w*, *link*)组成
  - *dst*: 邻接点域，存放邻接点的编号；
  - *w*:　数据域，存放边权值；
  - *link*:链域，下一个表结点的地址；

	v_0	v_1	v_2	v_3
v_0	0	0	0	1
v_1	1	0	1	0
v_2	1	1	0	0
v_3	0	0	0	0

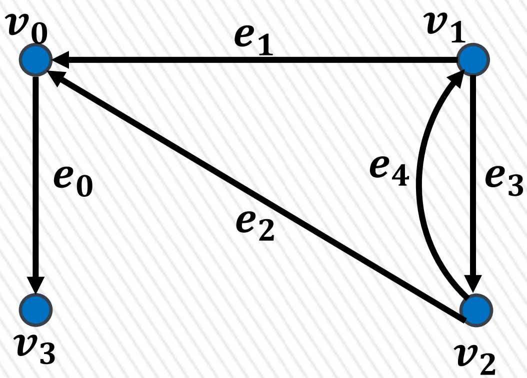


邻接表

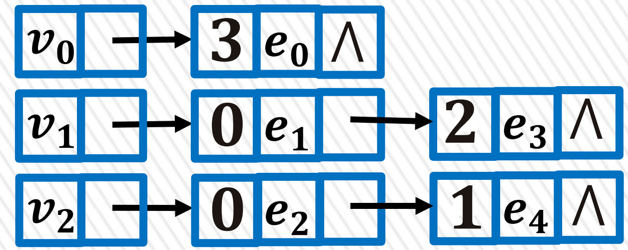


邻接表

邻接表



邻接表



- 典型场景：查询 v_i 的所有直接后继结点
  - 遍历第 v_i 对应的链表，时间复杂度O(d)
    - 其中 d 是 v_i 的出度，多数情况下，$d << n$;
- 思考几个问题：
  - 邻接表一定要使用链表吗？
    - 单链表的优势是长度可变，可以动态调整，缺点？
    - 图不需要修改时，长度为 d 的动态数组是否可行？
  - 直接“后继”好找了，直接“前驱”呢？

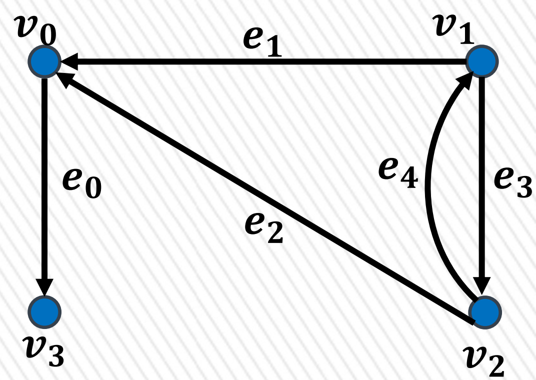
图的代数表示

常用表示方法

- 邻接矩阵
- 权矩阵
- 关联矩阵
- 边列表
- 正向、逆向表
- 邻接表
- 十字链表

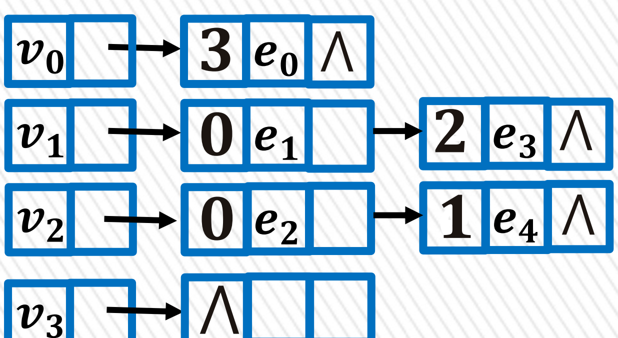
有向图的十字链表

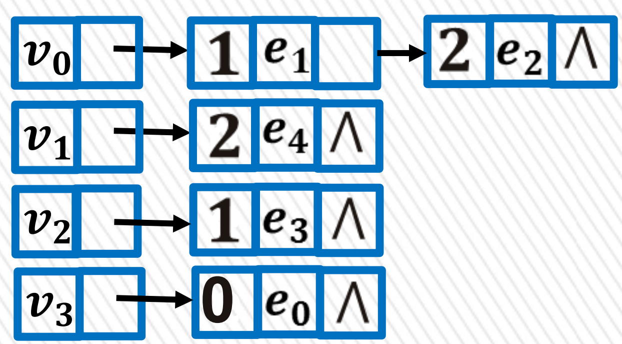
- 如何高效的表达后继到前驱的关系？



邻接表
结点 → [后继]

逆邻接表
结点 → [前驱]





有向图的十字链表

- 如何同时表示 前驱⇔后继 的双向关系？
  - 使用邻接表+逆邻接表，每条边需要保存2份
  - 更紧致的表示：十字链表（**Orthogonal List**）
- 思路：
  - 每行中的1串接 → 正向链表 → 后继的链表
  - 每列中的1串接 → 逆向链表 → 前驱的链表

	v_0	v_1	v_2	v_3
v_0	0	0	0	1
v_1	1	0	1	0
v_2	1	1	0	0
v_3	0	0	0	0

邻接表
(*dst*, *w*, *d_link*)

逆邻接表
(*src*, *w*, *s_link*)

十字链表的表结点
(*src*, *dst*, *w*, *d_link*, *s_link*)

- (*src*, *dst*) 是边的起始和终止结点
- *w* 是权值
- *d_link*： 正向链表
  - 下一个有相同起点的边
- *s_link*： 逆向链表
  - 下一个有相同终点的边

可行性：每个1（一条边）只出现在某1行、某1列，也就是说它只会出现在1个“正向链表”和1个“逆向链表”中

有向图的十字链表

- 有向图的十字链表

顶点

data	in	out
顶点信息数据		

指向该顶点的第一条入边

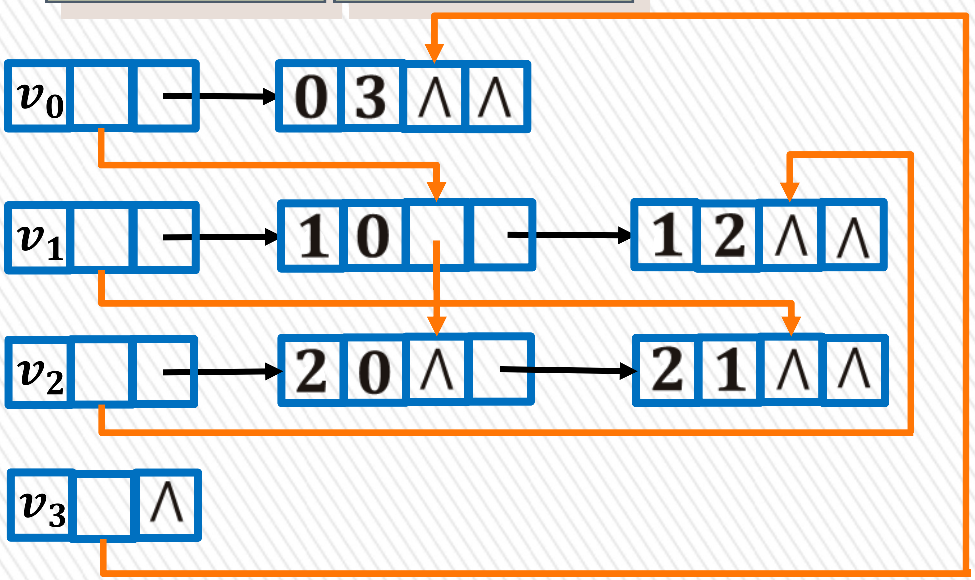
指向该顶点的第一条出边

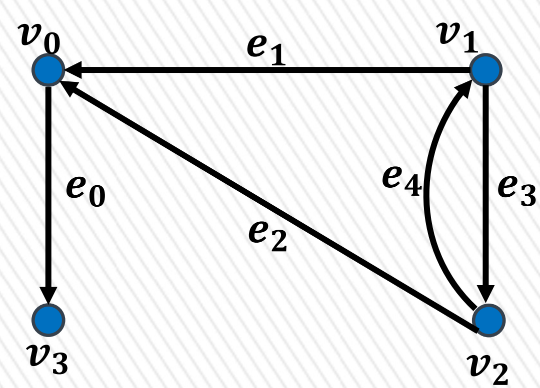
边

src	dst	info	d_link	s_link
边起点位置	边终点位置	边的相关信息		

指向下一个有相同终点的边

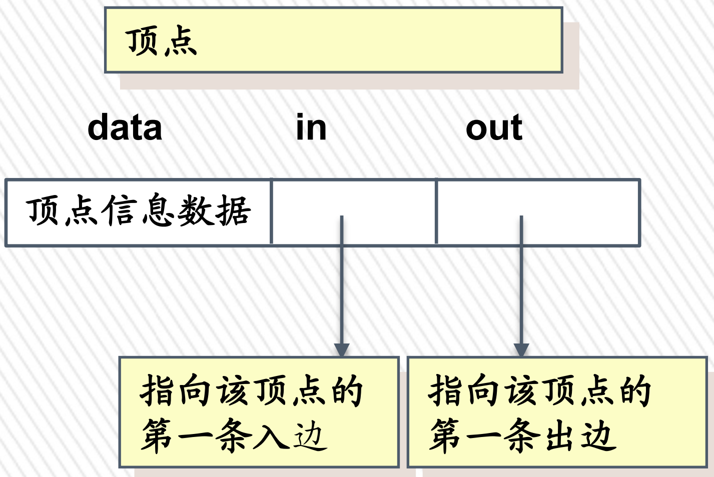
指向下一个有相同起点的边

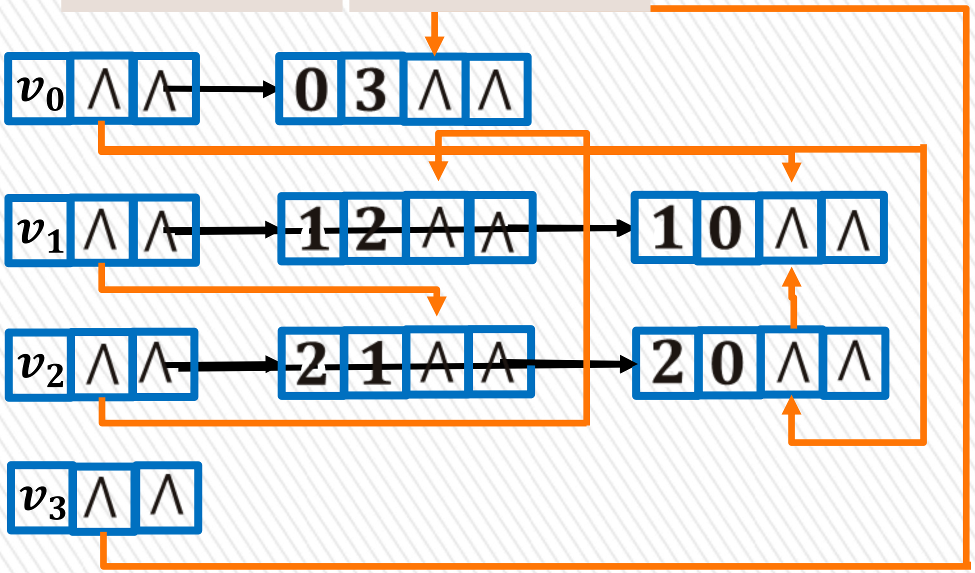


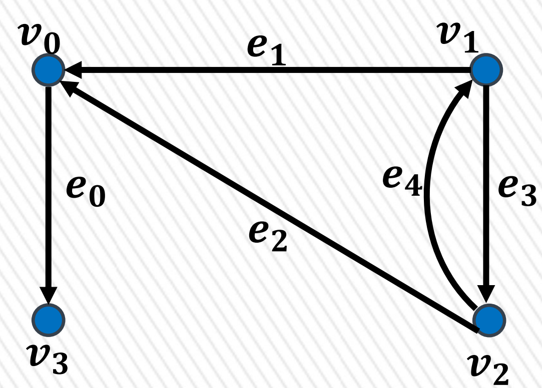


有向图的十字链表

- 有向图的十字链表







有向图的十字链表

- 有向图的十字链表

边

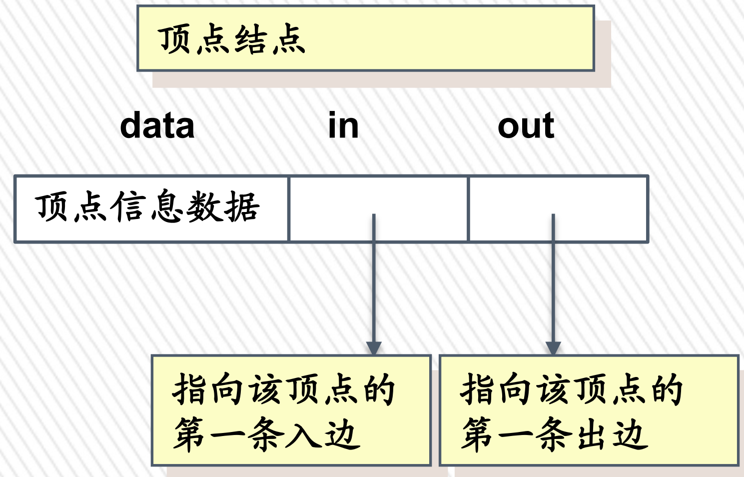
```
typedef struct edgestruct { // 边的结构表示
    int src, dst;   InfoType  *info;
    struct edgestruct  *d_link, *s_link;
    } Edgestruct;
```

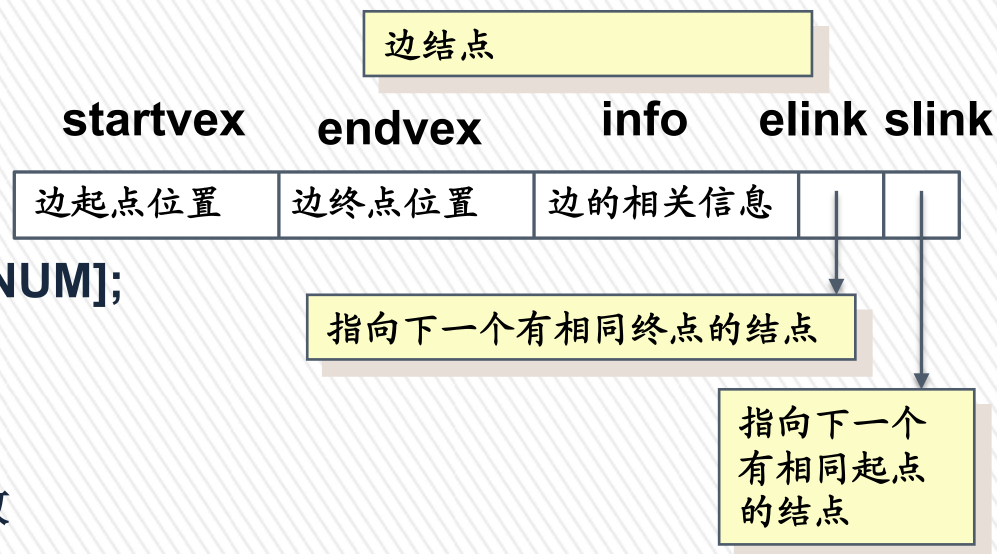
顶点

```
typedef struct vexNode { // 顶点的结构表示
    VertexType  data;
    Edgestruct  *in, *out;
} VexNode;
```

图结构

```
typedef struct oLGraph {
   VexNode  xlist[MAX_VERTEX_NUM];
          // 顶点结点(表头向量)
   int   vexnum, edgenum;
         //有向图的当前顶点数和边数
} OLGraph;
```

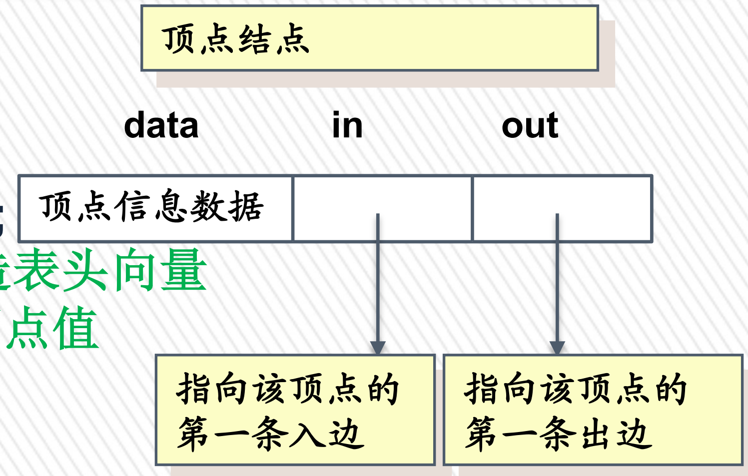


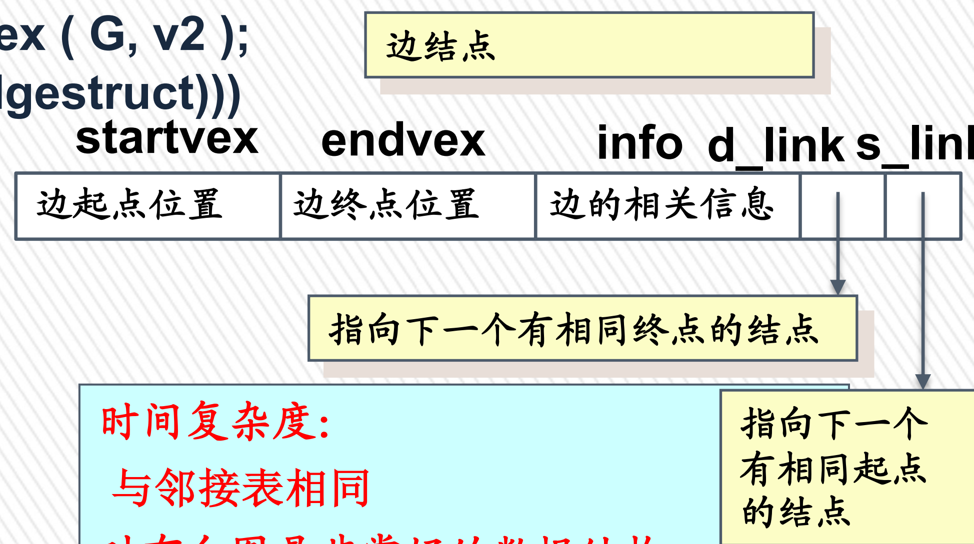


十字链表

构造图的十字链表算法

```
Status CreateDG ( OLGraph &G ) {
 scanf ( &G.vexnum, &G.edgenum, &IncInfo );
 for ( i = 0; i < G.vexnum; i++ ) {     //初始化构造表头向量
   scanf ( &G.xlist [i].data );               // 输入顶点值
   G.xlist[i].in = G.xlist[i].out = NULL;
 }
 for ( k = 0; k < G.edgenum; k++ ) {  // 构造十字链表
   scanf ( &v1, &v2 );                    // 输入一条边的始点和终点
   i = LocateVex ( G, v1 ); j = LocateVex ( G, v2 );
  if(!p=(Edgestruct *)malloc(sizeof(Edgestruct)))
      //产生新的边结点
      exit(OVERFLOW)
  p.src = i;  p.dst = j;   // 对边结点赋值
  p.d_link = G.xlist[j].in;      //插入
  p.s_link = G.xlist[i].out;
  G.xlist[j].in=G.xlist[i].out=p;
 }
} // CreateDG
```





时间复杂度:

与邻接表相同

对有向图是非常好的数据结构

谢谢！