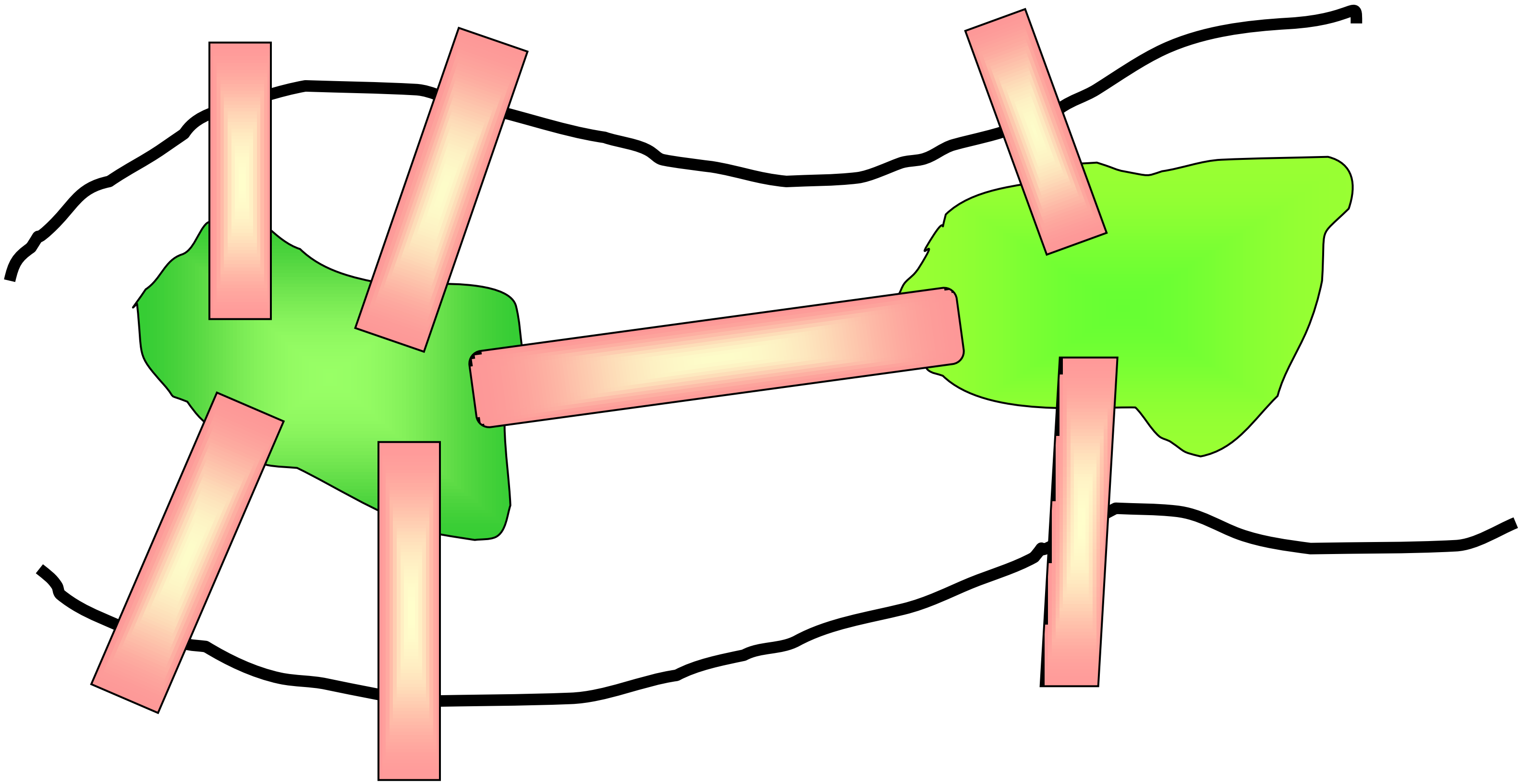
第4章 图结构及其应用算法

图论——欧拉

欧拉1707年出生在瑞士的巴塞尔城，19岁开始发表论文，直到76岁。几乎每一个数学领域都可以看到欧拉的名字，从初等几何的欧拉线，多面体的欧拉定理，立体解析几何的欧拉变换公式，四次方程的欧拉解法到数论中的欧拉函数，微分方程的欧拉方程，级数论的欧拉常数，变分学的欧拉方程，复变函数的欧拉公式等等。据统计他那不倦的一生，共写下了886本书籍和论文，其中分析、代数、数论占40%，几何占18%，物理和力学占28%，天文学占11%，弹道学、航海学、建筑学等占3%。1733年，年仅26岁的欧拉担任了彼得堡科学院数学教授。1741年到柏林担任科学院物理数学所所长，直到1766年，重回彼得堡，没有多久，完全失明。欧拉在数学上的建树很多，对著名的哥尼斯堡七桥问题的解答开创了图论的研究。

哥尼斯堡七桥问题

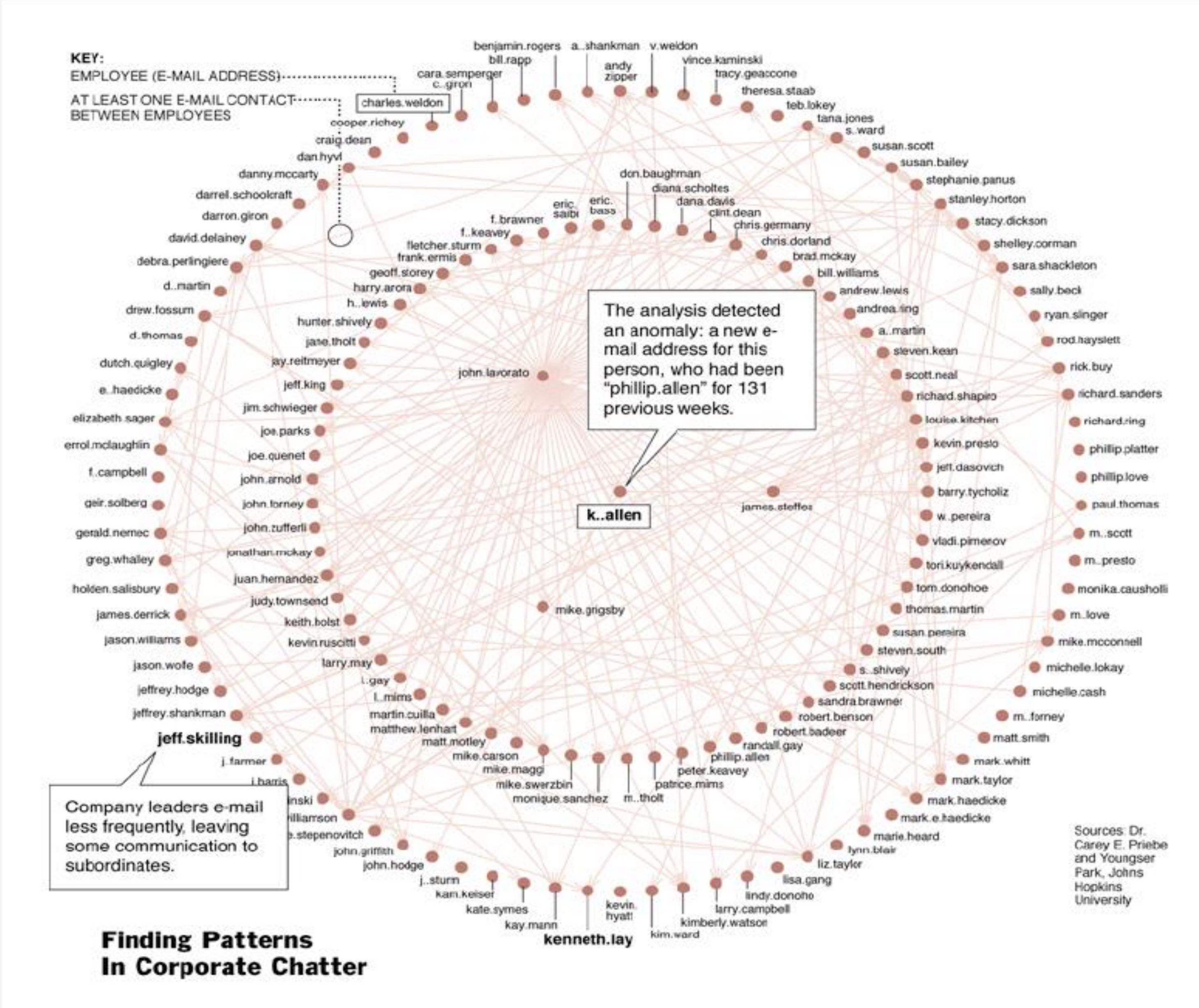


- 从某个陆地区域出发，是否存在一条能够经过每座桥一次且仅一次，最后回到出发地?

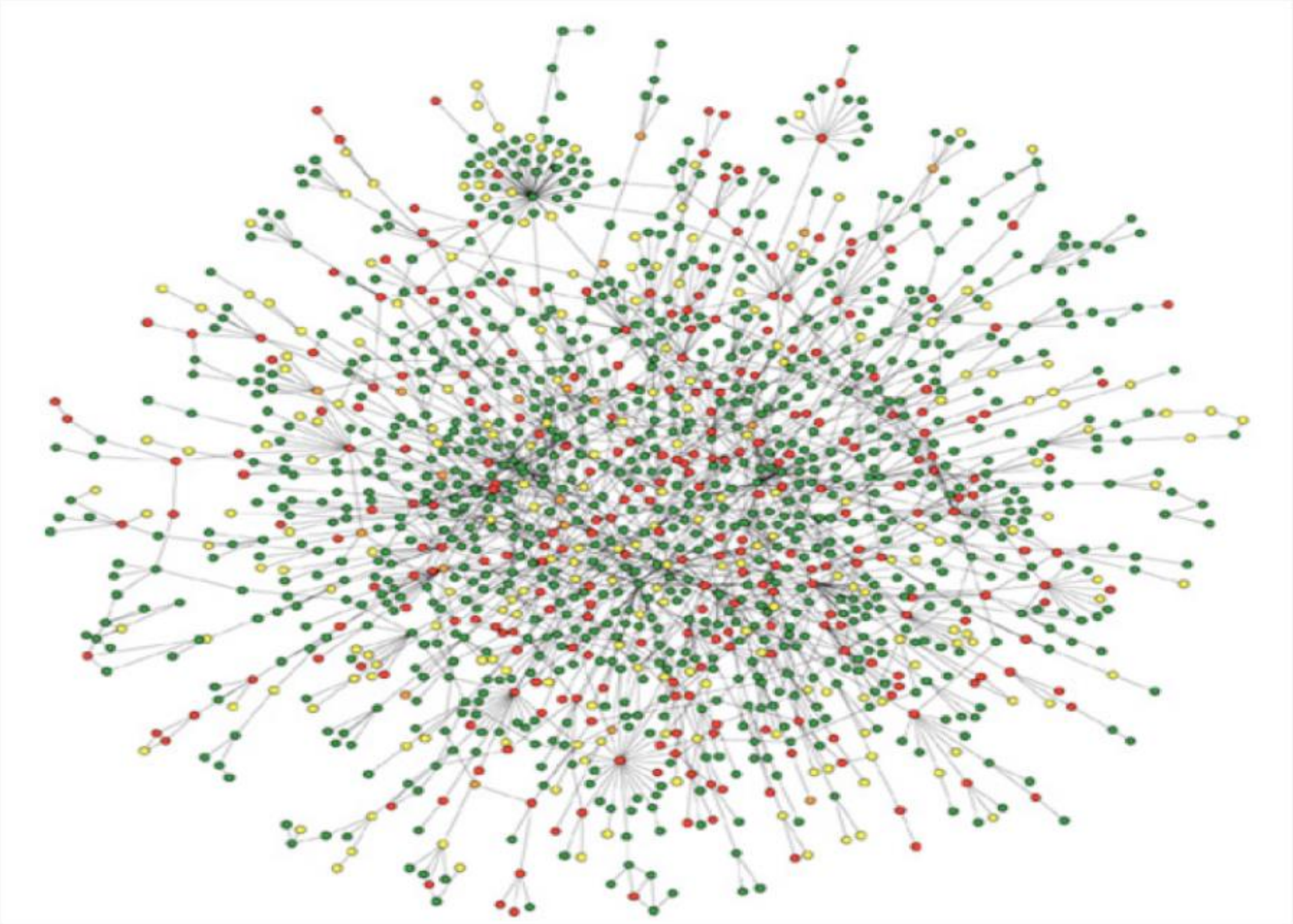
10 million Facebook friends

"Visualizing Friendships" by Paul Butler

One week of Enron emails



Protein-protein interaction network



Reference: Jeong et al, Nature Review | Genetics

学习目标

- 图结构是一种非线性结构，反映了数据对象之间的任意关系，在计算机科学、数学和工程中有着非常广泛的应用。
- 了解图的定义及相关的术语，掌握图的逻辑结构及其特点 ；
- 了解图的存储方法，重点掌握图的邻接矩阵和邻接表存储结构；
- 掌握图的遍历方法，重点掌握图的遍历算法的实现；
- 了解图的应用，重点掌握最小生成树算法、最短路径算法、拓扑排序和关键路径算法的基本思想、算法原理和实现过程。

主要内容

4.1 基本定义

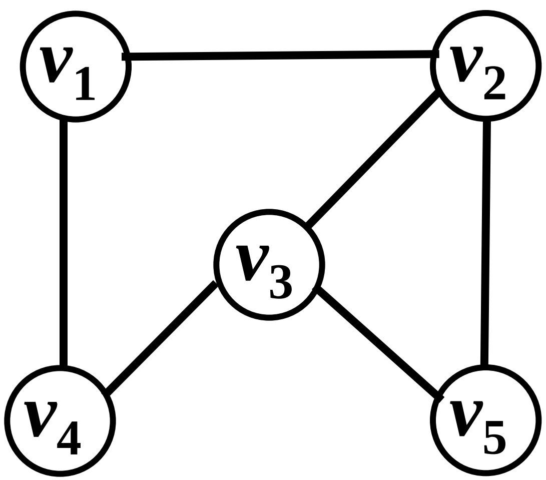
定义1 图(Graph)

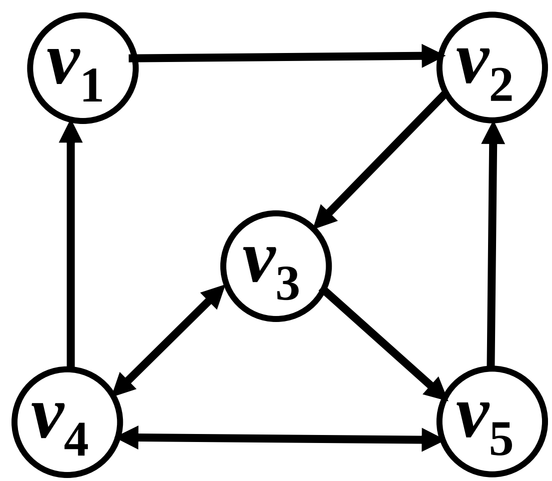
- 图是由顶点（vertex）的有穷非空集合和顶点之间边（edge）的集合组成的一种数据结构，通常表示为：

$$G = (V, E)$$

其中：*G*表示一个图，*V*是图*G*中顶点的集合，*E*是图*G*中顶点之间边的集合。

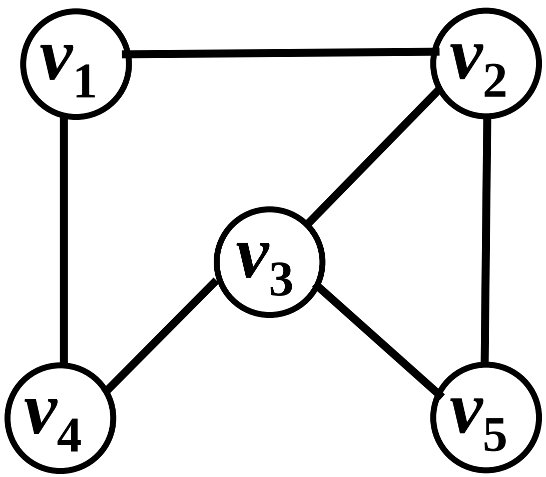
顶点表示数据对象；边表示数据对象之间的关系。



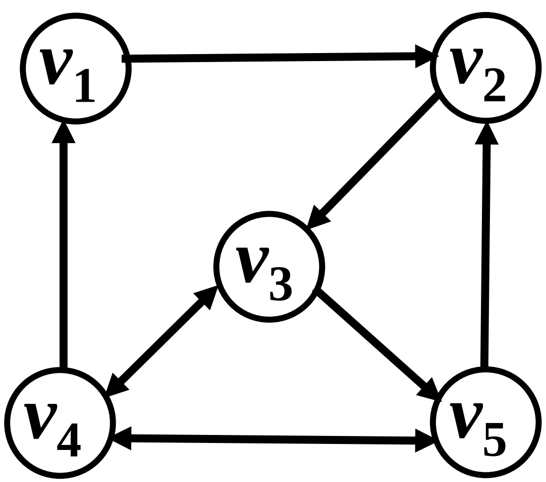


4.1 基本定义(cont.)

定义1 图



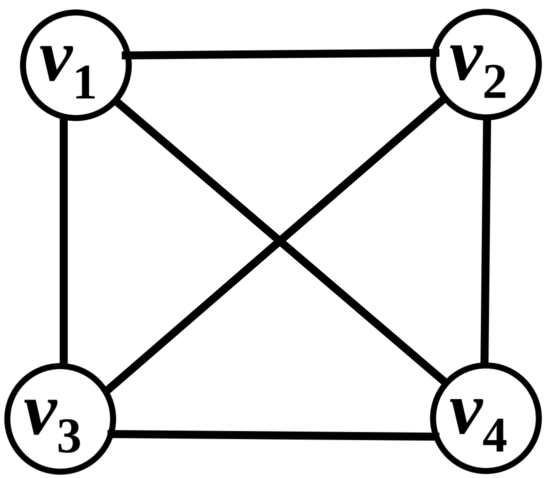
- 无向图：
  - 若顶点v_i和v_j之间的边没有方向，则称这条边为无向边，表示为(v_i,v_j)。
  - 如果图的任意两个顶点之间的边都是无向边，则称该图为无向图。

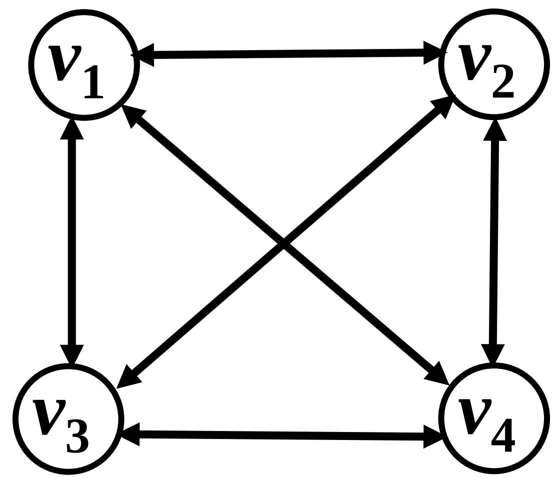


- 有向图：
  - 若顶点v_i和v_j之间的边都有方向，则称这条边为有向边(弧)，表示为$<v_i,v_j>$。
  - 如果图的任意两个顶点之间的边都是有向边，则称该图为有向图。

4.1 基本定义(cont.)

- **无向完全图**：在无向图中，如果任意两个顶点之间都存在边，则称该图为无向完全图。
- **有向完全图**：在有向图中，如果任意两个顶点之间都存在方向相反的两条弧，则称该图为有向完全图。

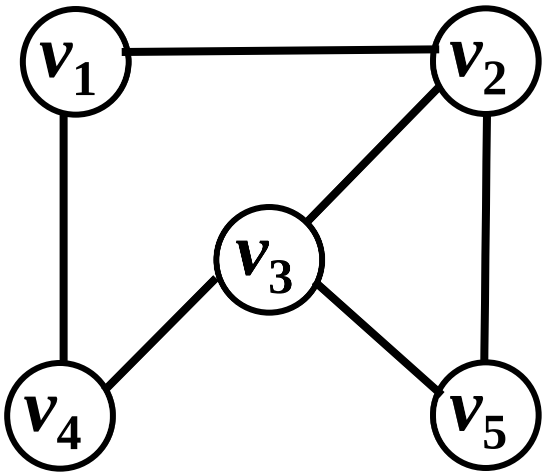


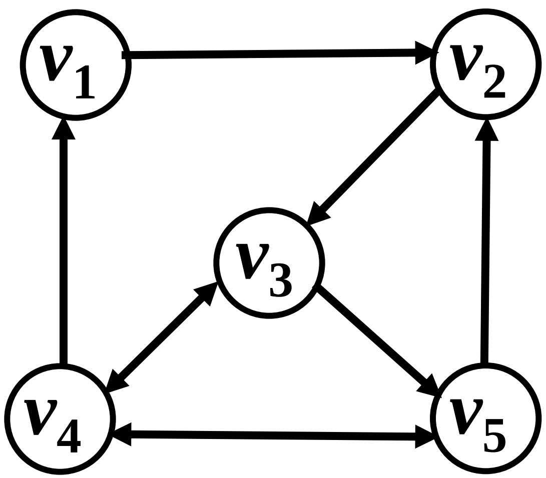


- 含有*n*个顶点的无向完全图有多少条边？
- 含有*n*个顶点的有向完全图有多少条弧？

4.1 基本定义(cont.)

定义1 图

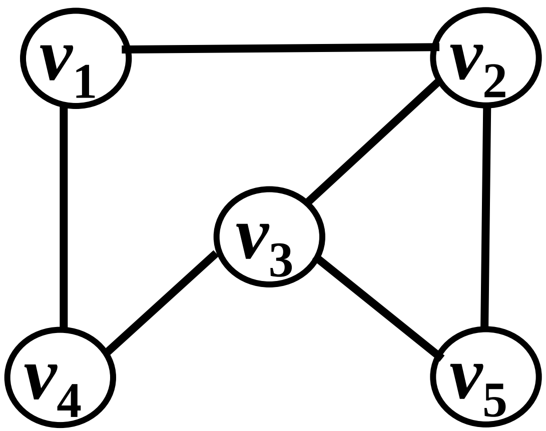


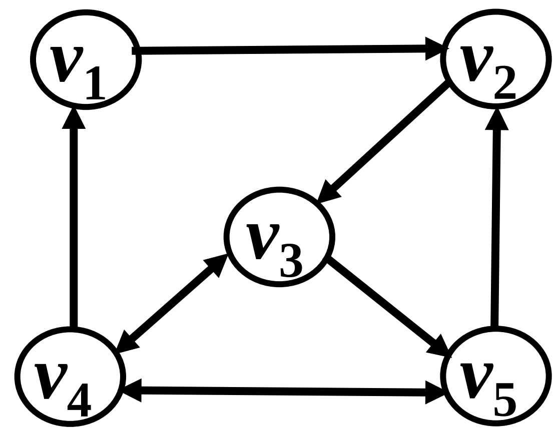


- 邻接、依附
  - 在无向图中，对于任意两个顶点v_i和顶点v_j，若存在边(v_i，v_j)，则称顶点v_i和顶点v_j相邻，互为邻接点，同时称边(v_i，v_j)依附于顶点v_i和顶点v_j。
  - 如：v_2的邻接点： v_1, v_3 ,v_5
  - 在有向图中，对于任意两个顶点v_i和顶点v_j，若存在有向边<v_i，v_j>，则称顶点v_i邻接到顶点v_j，顶点v_j邻接于顶点v_i，同时称弧<v_i，v_j>依附于顶点v_i和顶点v_j 。
  - 如：v_1邻接到v_2， v_1 邻接于v_4

4.1 基本定义(cont.)

定义2 度(Dgree)





- **顶点的度：**在无向图中，顶点*v*的**度**是指依附于该顶点的边数，通常记为D (*v*)。
- **顶点的入度：**在有向图中，顶点v的**入度**是指以该顶点为弧头的弧的数目，记为ID (*v*)；
- **顶点的出度：**在有向图中，顶点v的**出度**是指以该顶点为弧尾的弧的数目，记为OD (*v*)。
- **在有向图中，** D (*v*)= ID (*v*) + OD (*v*)

4.1 基本定义(cont.)

定义2 度(Dgree)

- 在具有n个顶点、e条边的无向图G中，各顶点的度之和与边数之和的关系？

$$\sum_{i=1}^{n} D(v_i) = 2e$$

- 在具有n个顶点、e条边的有向图G中，各顶点的入度之和与各顶点的出度之和的关系？与边数之和的关系？

$$\sum_{i=1}^{n} ID(v_i) = \sum_{i=1}^{n} OD(v_i) = e$$

4.1 基本定义(cont.)

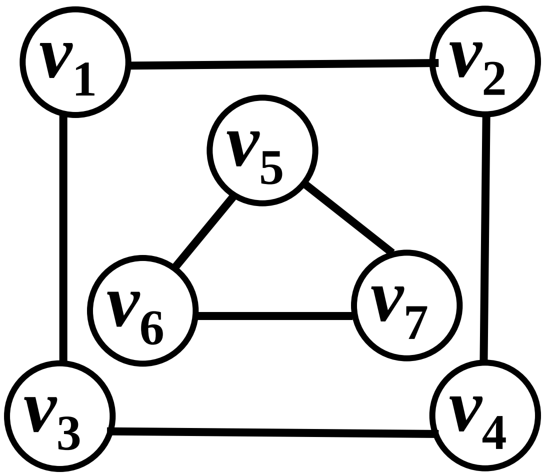
定义3 路径（Path）和路径长度、简单路和简单回路

- 在无向图G=(V, E) 中，若存在一个顶点序列$v_p, v_{i1}, v_{i2}, \ldots v_{im}, v_q$，使得（$v_p, v_{i1}$），（$v_{i1}, v_{i2}$），…，（$v_{im}, v_q$）∈E(G)，则称顶点$v_p$路到$v_q$有一条路径。
- 在有向图G =(V, E)中，若存在一个顶点序列$v_p, v_{i1}, v_{i2}, \ldots v_{im}, v_q$，使得有向边$<v_p, v_{i1}>, <v_{i1}, v_{i2}>, \ldots, <v_{im}, v_q>$∈E(G)，则称顶点$v_p$路到$v_q$有一条有向路径。
- 非带权图的路径长度是指此路径上边的条数。
- 带权图的路径长度是指路径上各边的权之和。
- 简单路径：若路径上各顶点 $v_1, v_2, \ldots, v_m$ 均互不相同(第一个顶点和最后一个顶点可以相同）, 则称这样的路径为简单路径。
- 简单回路：若路径上第一个顶点 v_1与最后一个顶点v_m重合, 则称这样的简单路径为简单回路或环。
- 一条环路的长度至少为1（无向图为3），且起点和终点相同的简单路径。

4.1 基本定义(cont.)

定义4 图的连通性

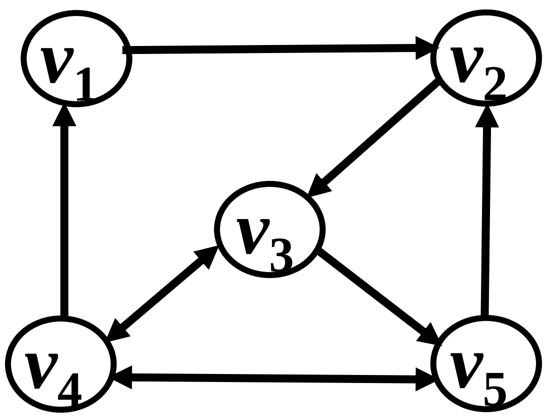
- 连通图与连通分量
  - 顶点的连通性：在无向图中,若从顶点v_i到顶点v_j $(i\neq j)$有路径,则称顶点v_i与v_j是连通的。
  - 连通图：如果一个无向图中任意一对顶点都是连通的,则称此图是连通图。
  - 连通分量：非连通图的极大连通子图叫做连通分量。



4.1 基本定义(cont.)

定义4 图的连通性

- 强连通图与强连通分量
  - 顶点的强连通性：在有向图中, 若对于每一对顶点v_i和v_j ($i \neq j$), 都存在一条从v_i到v_j和从v_j到v_i的有向路径,则称顶点v_i与v_j是强连通的。
  - 强连通图：如果一个有向图中任意一对顶点都是强连通的, 则称此有向图是强连通图。
  - 强连通分量:非强连通图的极大强连通子图叫做强连通分量



4.1 基本定义(cont.)

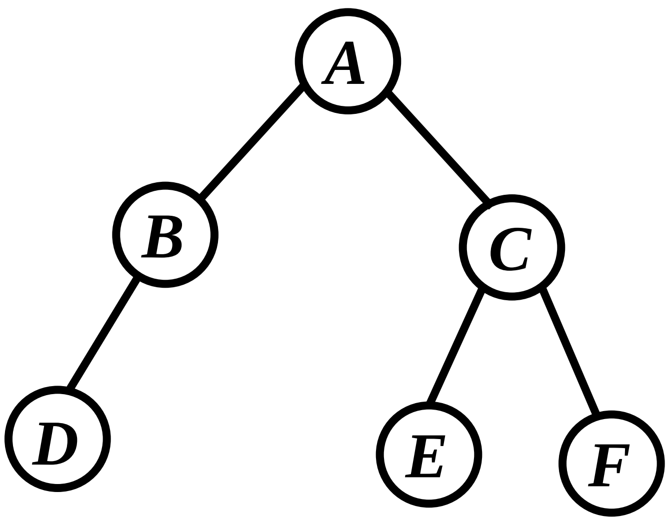
图的操作

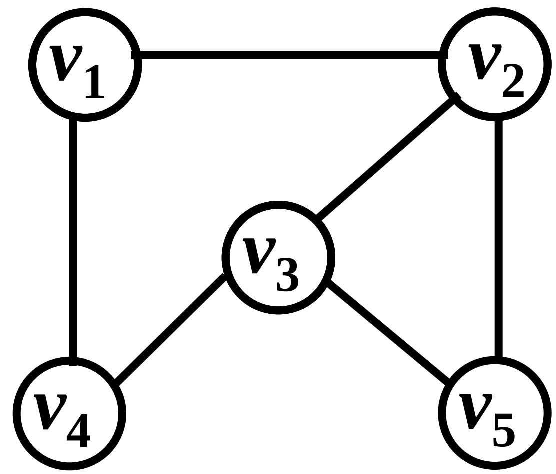
设图G=(V,E)，图上定义的基本操作如下：

- NewNode (G, v)：建立一个新顶点，V=V∪{v}
- DelNone (G, v)：删除顶点v以及与之相关联的所有边
- SetSucc (G, v1, v2):增加一条边，E = E∪(v1,v2) ,V=V
- DelSucc (G, v1, v2)：删除边（v1,v2）,V不变
- Succ (G, v)：求出v的所有直接后继结点
- Pred (G, v)：求出v的所有直接前导结点
- IsEdge (G, v1, v2)：判断（v1,v2）∈E
- FirstAdjVex(G , v)：顶点v 的第一个邻接顶点
- NextAdjVex(G, v, w)：顶点v 的某个邻接点w的下一个邻接顶点。

4.1 基本定义(cont.)

不同逻辑结构之间的比较

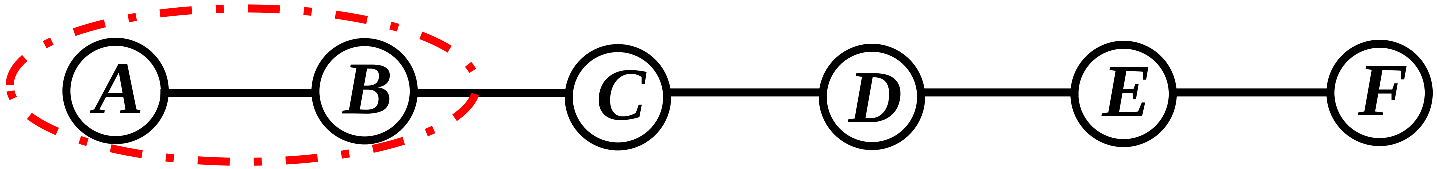


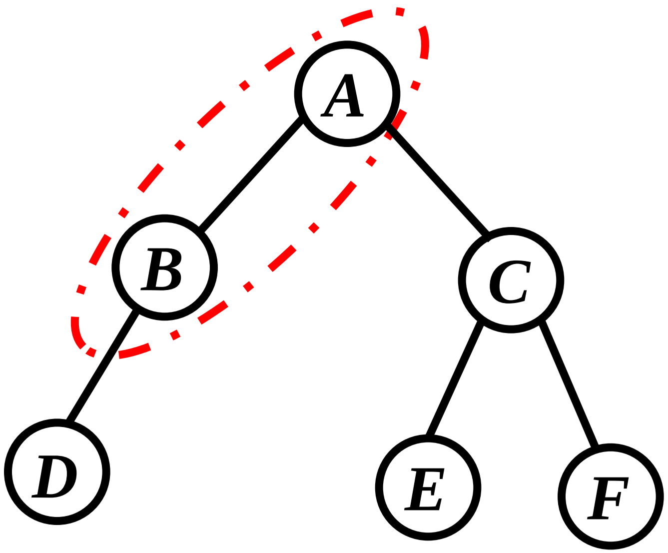


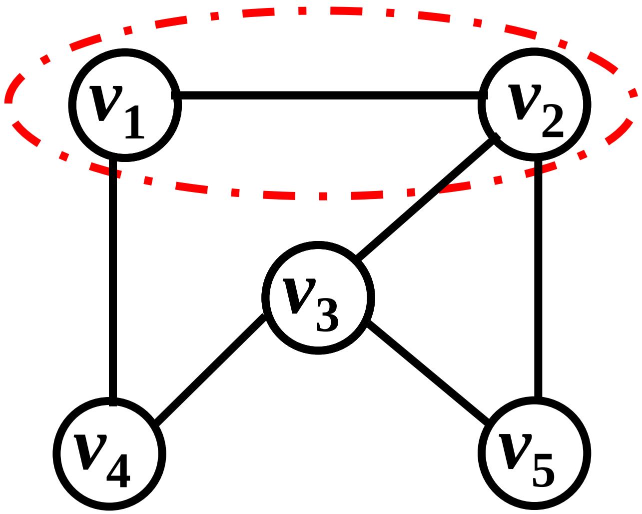
- 在线性结构中，数据元素之间仅具有线性关系(1:1)；
- 在树型结构中，结点之间具有层次关系(1:m)；
- 在图型结构中，任意两个顶点之间都可能有关系(m:n)。

4.1 基本定义(cont.)

不同逻辑结构之间的比较



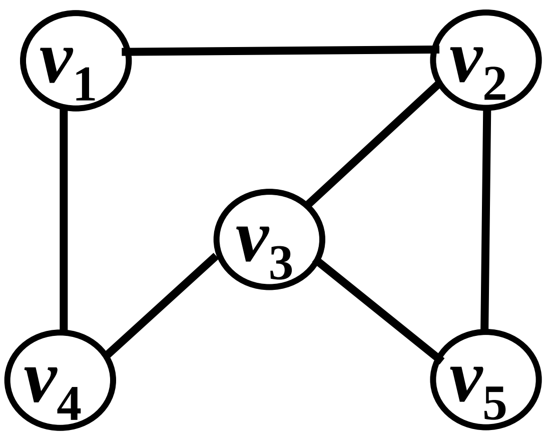


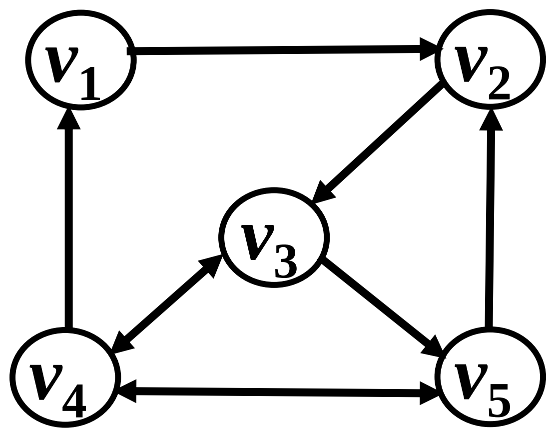


- 在线性结构中，元素之间的关系为前驱和后继；
- 在树型结构中，结点之间的关系为双亲和孩子；
- 在图型结构中，顶点之间的关系为邻接。

4.2 图的存储结构

- 是否可以采用顺序存储结构存储图(一维数组）？
  - 图的特点：顶点之间的关系是*m*:*n*，即任何两个顶点之间都可能存在关系（边），无法通过存储位置表示这种任意的逻辑关系，所以，图无法采用顺序存储结构。
- 如何存储图？
  - 考虑图的定义，图是由顶点和边组成的；
  - 如何存储顶点、如何存储边----顶点之间的关系。



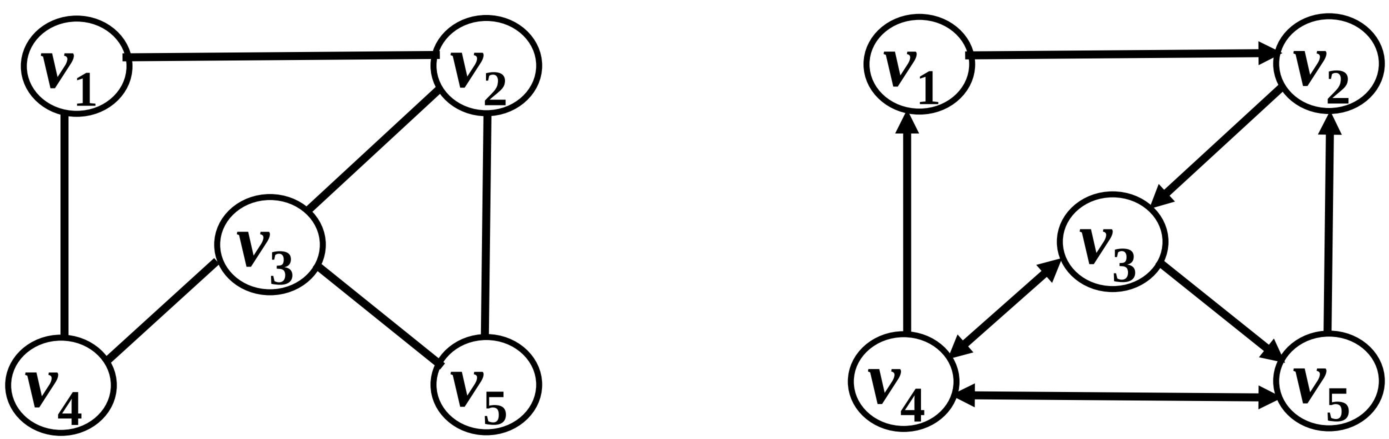


4.2 图的存储结构(cont.)

邻接矩阵 （*Adjacency Matrix*）表示（数组表示法）

- 基本思想：
  - 用一个一维数组存储图中顶点的信息，用一个二维数组（称为邻接矩阵）存储图中各顶点之间的邻接关系。
  - 假设图$G=(V, E)$有n个顶点，则邻接矩阵是一个$n \times n$的方阵，定义为：

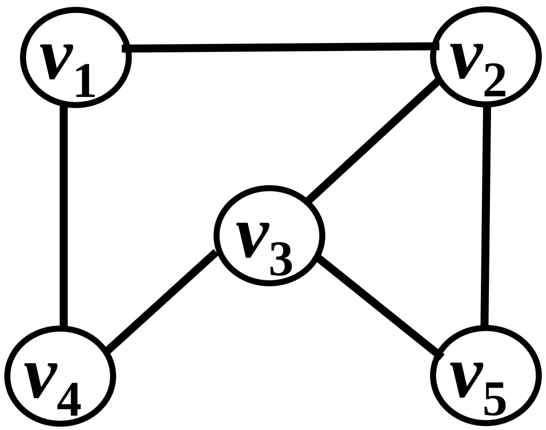
$$\text{edge}[i][j] = \begin{cases} 1 & \text{若 } (i, j) \in E \text{ 或 } \langle i, j \rangle \in E \\ 0 & \text{否则} \end{cases}$$



4.2 图的存储结构(cont.)

邻接矩阵 （*Adjacency Matrix*）表示（数组表示法）

- 无向图的邻接矩阵：



vertex= | v_1 | v_2 | v_3 | v_4 | v_5 |

$$
\text{edge} = \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}
$$

- 存储结构特点：
  - 主对角线为 0 且一定是对称矩阵；

问题：1. 如何求顶点v_i的度？

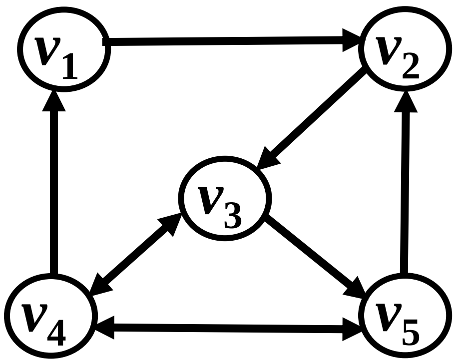
2.如何判断顶点 v_i和 v_j 之间是否存在边？

3.如何求顶点 v_i的所有邻接点？

4.2 图的存储结构(cont.)

邻接矩阵 （*Adjacency Matrix*）表示（数组表示法）

- 有向图的邻接矩阵：



vertex=

v_1	v_2	v_3	v_4	v_5

$$\text{edge} = \begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} & \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} & \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \end{array}$$

- 存储结构特点：
  - 有向图的邻接矩阵一定不对称吗？

问题：1.如何求顶点v_i的出度？

2. 如何判断顶点 v_i和 v_j 之间是否存在有向边？

3.如何求邻接于顶点 v_i的所有顶点？

4.如何求邻接到顶点 v_i的所有顶点？

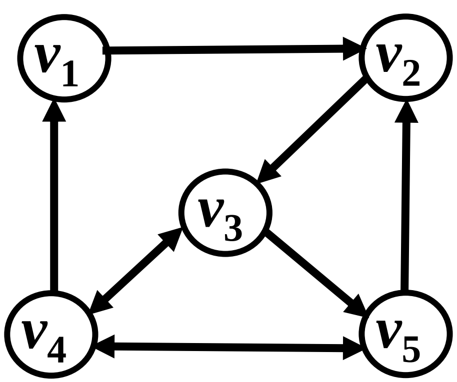
4.2 图的存储结构(cont.)

邻接矩阵 （*Adjacency Matrix*）表示（数组表示法）

- 存储结构定义：

假设图*G*有*n*个顶点*e*条边，则该图的存储需求为 $O(n+n^2) = O(n^2)$，与边的条数e无关。

```
typedef struct {
    VertexData verlist [NumVertices];  //顶点表
    EdgeData edge[NumVertices][NumVertices];
        //邻接矩阵—边表, 可视为边之间的关系
    int n, e;  //图的顶点数与边数
} MTGraph;
```



vertex

v_1	v_2	v_3	v_4	v_5

$$
\text{edge}=\begin{array}{c} \\ \\ \\ \\ \\ \end{array}
\begin{array}{c c}
\begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} & \\
\begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix} & \begin{array}{c} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}
\end{array}
$$

4.2 图的存储结构(cont.)

- 存储结构的建立----算法实现的步骤：

1.确定图的顶点个数n和边数e；

2.输入顶点信息存储在一维数组vertex中；

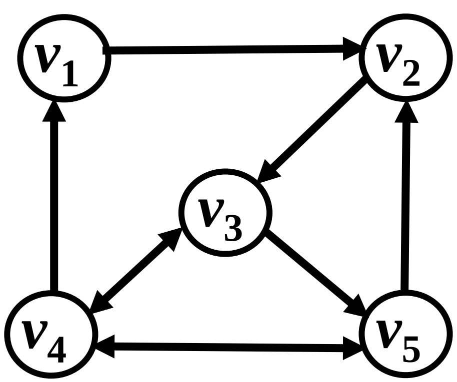
3.初始化邻接矩阵；

4.依次输入每条边存储在邻接矩阵edge中；

  4.1 输入边依附的两个顶点的序号i, j；

  4.2 将邻接矩阵的第i行第j列的元素值置为1；

  4.3 将邻接矩阵的第j行第i列的元素值置为1。



vertex

v_1	v_2	v_3	v_4	v_5

$$\text{edge}=\begin{array}{c} \begin{array}{ccccc} v_1 & v_2 & v_3 & v_4 & v_5 \end{array} \\ \left[\begin{array}{ccccc} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{array}\right] \end{array} \begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array}$$

4.2 图的存储结构(cont.)

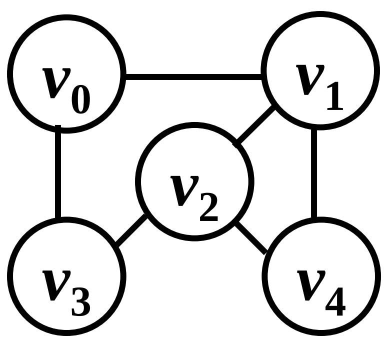
- 存储结构的建立算法的实现：

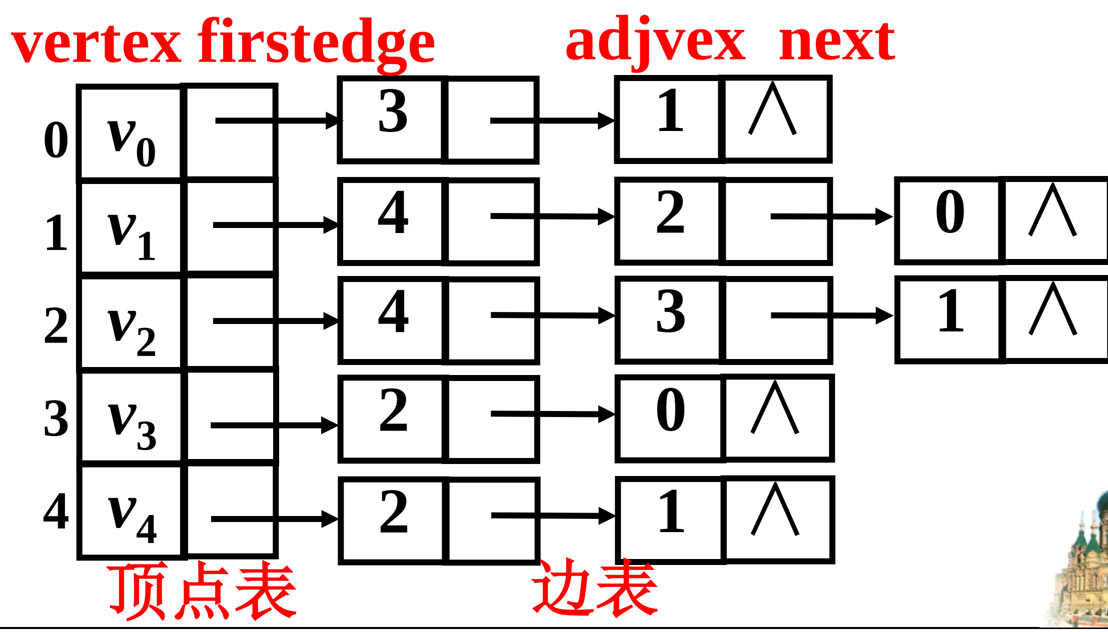
```
void  CreateMGragh (MTGragh *G) //建立图的邻接矩阵
{      int  i, j, k, w;
       cin>>G→n>>G→e;                 //1.输入顶点数和边数
       for (i=0; i<G→n; i++)          //2.读入顶点信息，建立顶点表
            G→vertlist[i]=getchar( );
       for (i=0; i<G→n; i++)
            for (j=0;j<G→n;j++)
                 G→edge[i][j]=0;         //3.邻接矩阵初始化
       for (k=0; k<G→e; k++) {        //4.读入e条边建立邻接矩阵
            cin>>i>>j>>w;              // 输入边（i,j）上的权值w
            G→ edge[i][j]=w; G→ edge[j][i]=w;
       }
} //时间复杂度：T = O( n+ n² +e) 。当e < <n, T = O( n² ) ?
```

4.2 图的存储结构(cont.)

邻接表（Adjacency List）表示

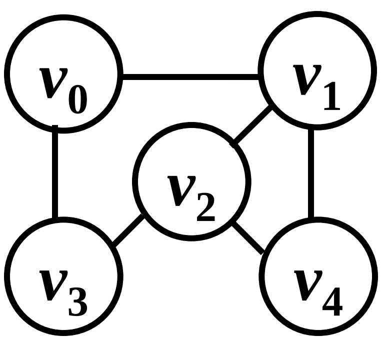
- 无向图的邻接表：
  - 对于无向图的每个顶点v_i，将所有与v_i相邻的顶点链成一个单链表，称为顶点v_i的边表（顶点v_i的邻接表）；
  - 再把所有边表的指针和存储顶点信息的一维数组构成顶点表。

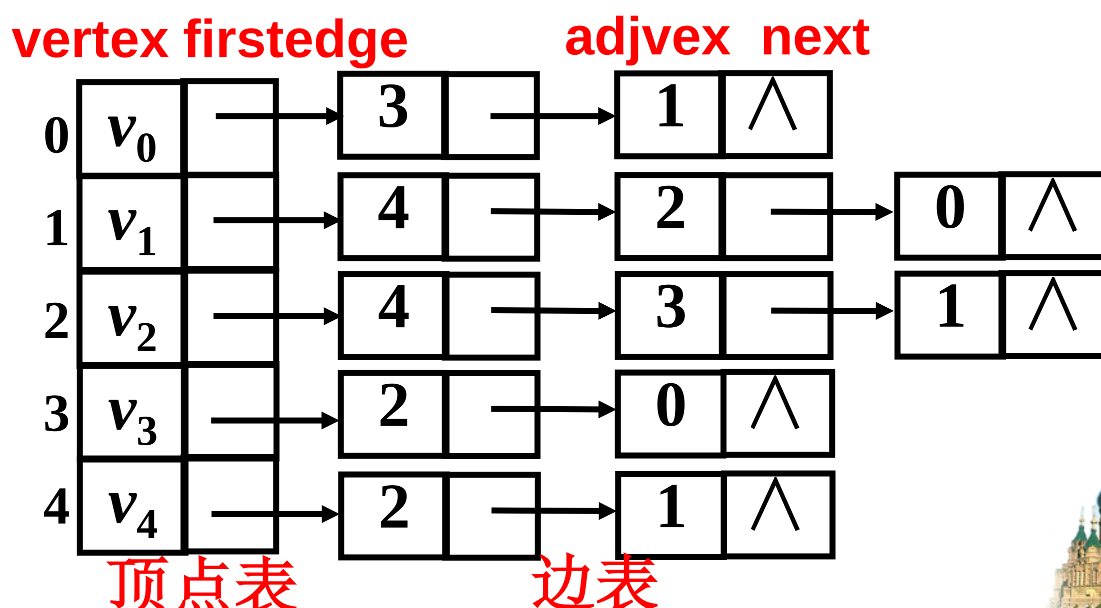




4.2 图的存储结构(cont.)

- 无向图的邻接表存储的特点:
  - 边表中的结点表示什么?
  - 如何求顶点 v_i的度?
  - 如何判断顶点v_i和顶点v_j之间是否存在边?
  - 如何求顶点 v_i的所有邻接点?
  - 空间需求O(n+2e)

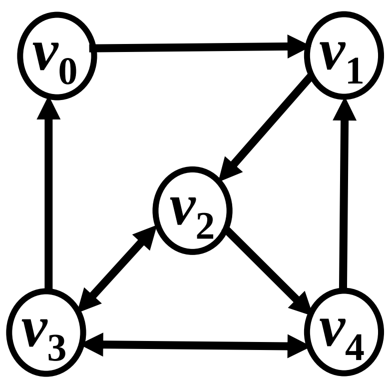


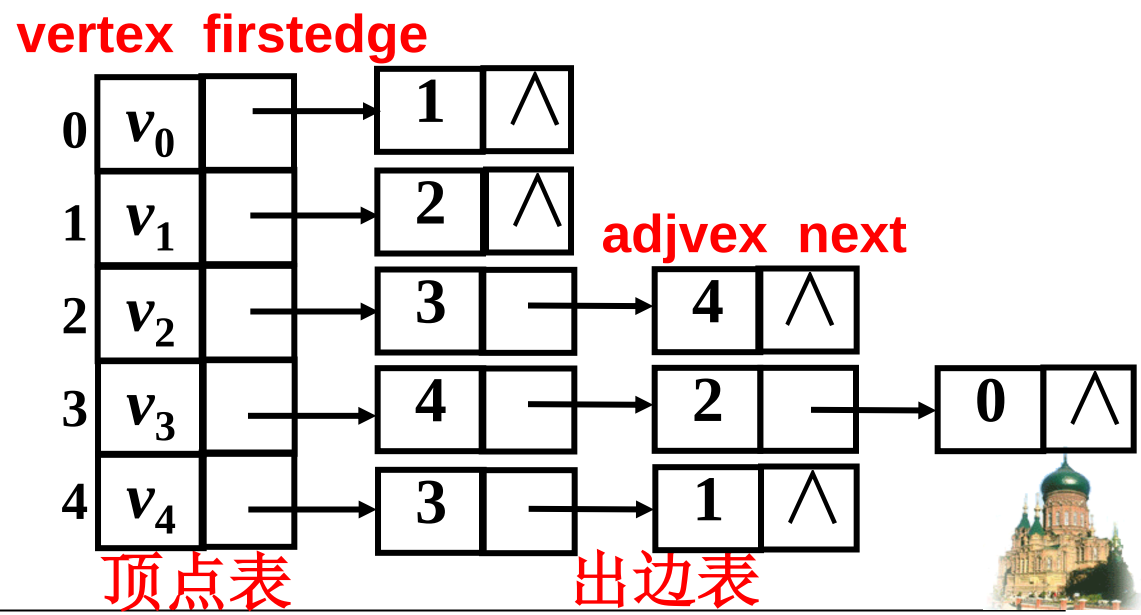


4.2 图的存储结构(cont.)

邻接表（Adjacency List）表示

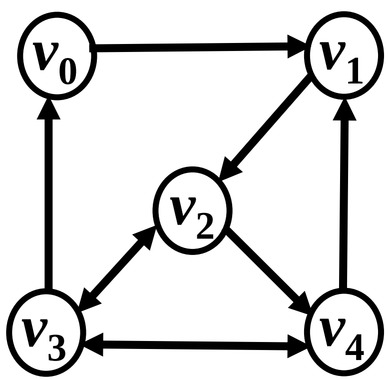
- 有向图的邻接表---正邻接表
  - 对于有向图的每个顶点v_i，将邻接于v_i的所有顶点链成一个单链表，称为顶点v_i的出边表；
  - 再把所有出边表的指针和存储顶点信息的一维数组构成顶点表。

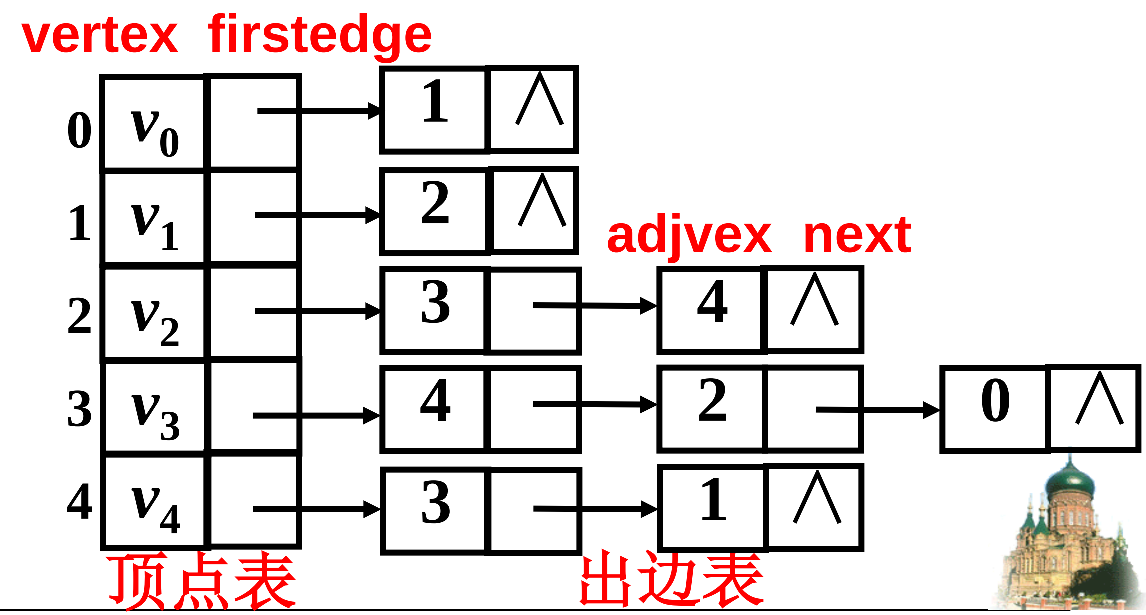




4.2 图的存储结构(cont.)

- 有向图的正邻接表的存储特点
  - 出边表中的结点表示什么？
  - 如何求顶点 v_i的出度？如何求顶点 v_i的入度？
  - 如何判断顶点 v_i和顶点v_j之间是否存在有向边?
  - 如何求邻接于顶点 v_i的所有顶点?
  - 如何求邻接到顶点 v_i的所有顶点?
  - 空间需求:O(n+e)

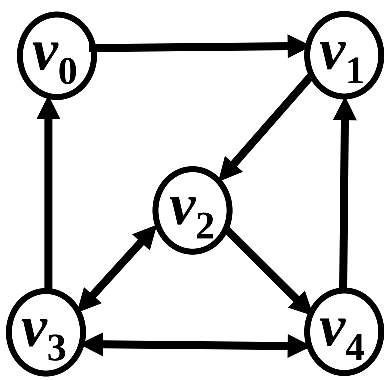


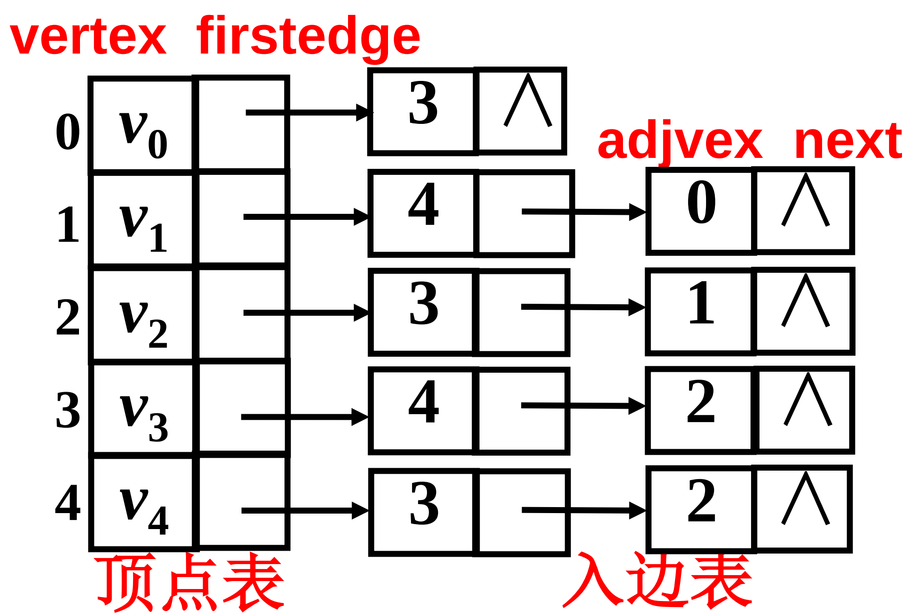


4.2 图的存储结构(cont.)

邻接表（Adjacency List）表示

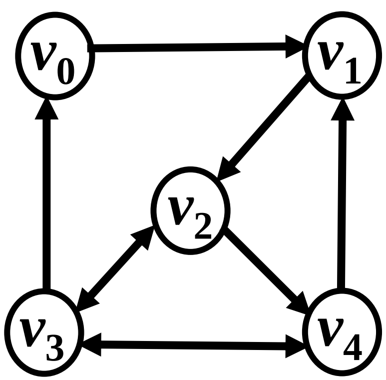
- 有向图的邻接表----逆邻接表
  - 对于有向图的每个顶点v_i，将邻接到v_i的所有顶点链成一个单链表，称为顶点v_i的入边表；
  - 再把所有入边表的指针和存储顶点信息的一维数组构成顶点表。

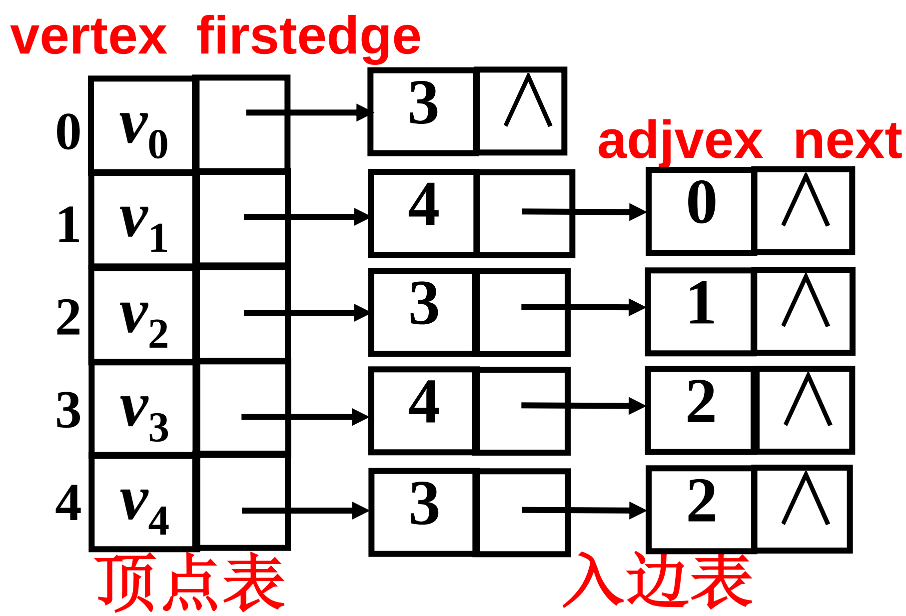




4.2 图的存储结构(cont.)

- 有向图的逆邻接表的存储特点
  - 出边表中的结点表示什么?
  - 如何求顶点 v_i的入度? 如何求顶点 v_i的出度?
  - 如何判断顶点 v_i和顶点v_j之间是否存在有向边?
  - 如何求邻接到顶点 v_i的所有顶点?
  - 如何求邻接于顶点 v_i的所有顶点?
  - 空间需求:O(n+e)





4.2 图的存储结构(cont.)

- 邻接表存储结构的定义

```
typedef struct node {//边表结点
    int adjvex;          //邻接点域（下标）
    EdgeData cost;            //边上的权值
    struct node *next; //下一边链接指针
} EdgeNode;
typedef struct {             //顶点表结点
    VertexData  vertex; //顶点数据域
    EdgeNode * firstedge;//边链表头指针
} VertexNode;
typedef struct {         //图的邻接表
   VertexNode vexlist [NumVertices];
    int n, e;            //顶点个数与边数
} AdjGraph;
```

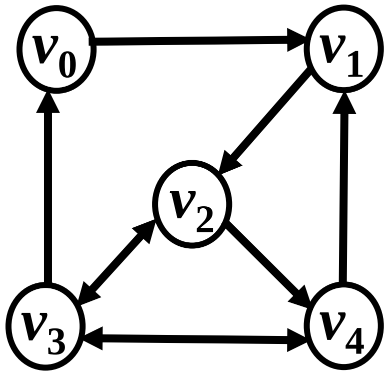
边表结点

adjvex	cost	next

顶点表结点

vertex	firstedge

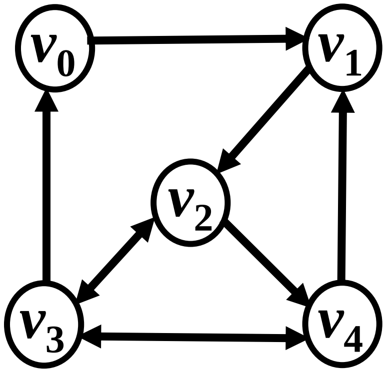
4.2 图的存储结构(cont.)



- 邻接表存储结构建立算法实现的步骤：

1. 确定图的顶点个数和边的个数；
2. 建立顶点表：
   2.1 输入顶点信息；
   2.2 初始化该顶点的边表；
3. 依次输入边的信息并存储在边表中；
   3.1 输入边所依附的两个顶点的序号tail和head和权值w；
   3.2 生成邻接点序号为head的边表结点p；
   3.3 设置边表结点p；
   3.4 将结点p插入到第tail个边表的头部；

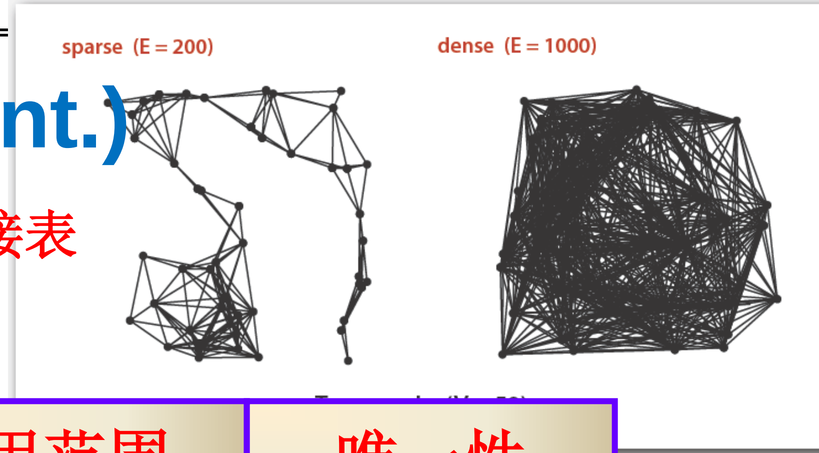
4.2 图的存储结构(cont.)



- 邻接表存储结构建立算法的实现：

```
void CreateGraph (AdjGraph G)
{  cin >> G.n >> G.e;                          //1.输入顶点个数和边数
   for ( int i = 0; i < G.n; i++) {            //2.建立顶点表
      cin >> G.vexlist[i].vertex;              //2.1输入顶点信息
      G.vexlist[i].firstedge = NULL; }         //2.2边表置为空表
   for ( i = 0; i < G.e; i++) {                //3.逐条边输入,建立边表
      cin >> tail >> head >> weight;           //3.1输入
      EdgeNode * p = new EdgeNode;             //3.2建立边结点
      p→adjvex = head;   p→cost = weight;      //3.3设置边结点
      p→next = G.vexlist[tail].firstedge;      //3.4链入第 tail 号链表的前端
      G.vexlist[tail].firstedge = p;
      p = new EdgeNode;
      p→adjvex = tail;   p→cost = weight;
      p→next = G.vexlist[head].firstedge; //链入第 head 号链表的前端
     G.vexlist[head].firstedge = p; }
} //时间复杂度：O(2e+n)
```

4.2 图的存储结构(cont.)



- 图的存储结构的比较——邻接矩阵和邻接表

	空间性能	时间性能	适用范围	唯一性
邻接矩阵	O（n^2）	O（n^2）	稠密图	唯一？
邻接表	O（n+e）	O（n+e）	稀疏图	不唯一？

◆ 十字链表(有向图)

- 十字链表是有向图的另一种链式存储结构。
- 将有向图的邻接表和逆邻接表结合起来的结构。
- 在十字链表中有两种结点：
  - 弧结点：存储每一条弧的信息，用链表链接在一起。

弧结点结构:

ivex	jvex	jlink	ilink	info

  - jlink:指向j的边；ilink:i发出的边

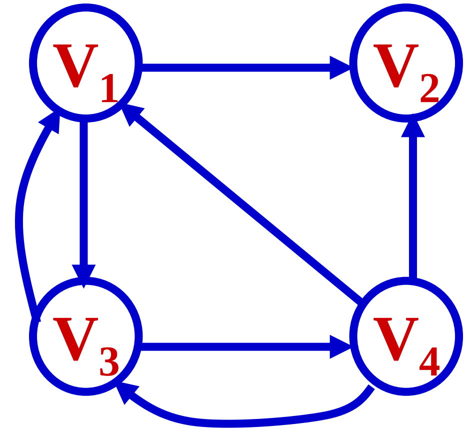
  - 顶点结点：存储每一个顶点的信息，使用一维数组来存储。

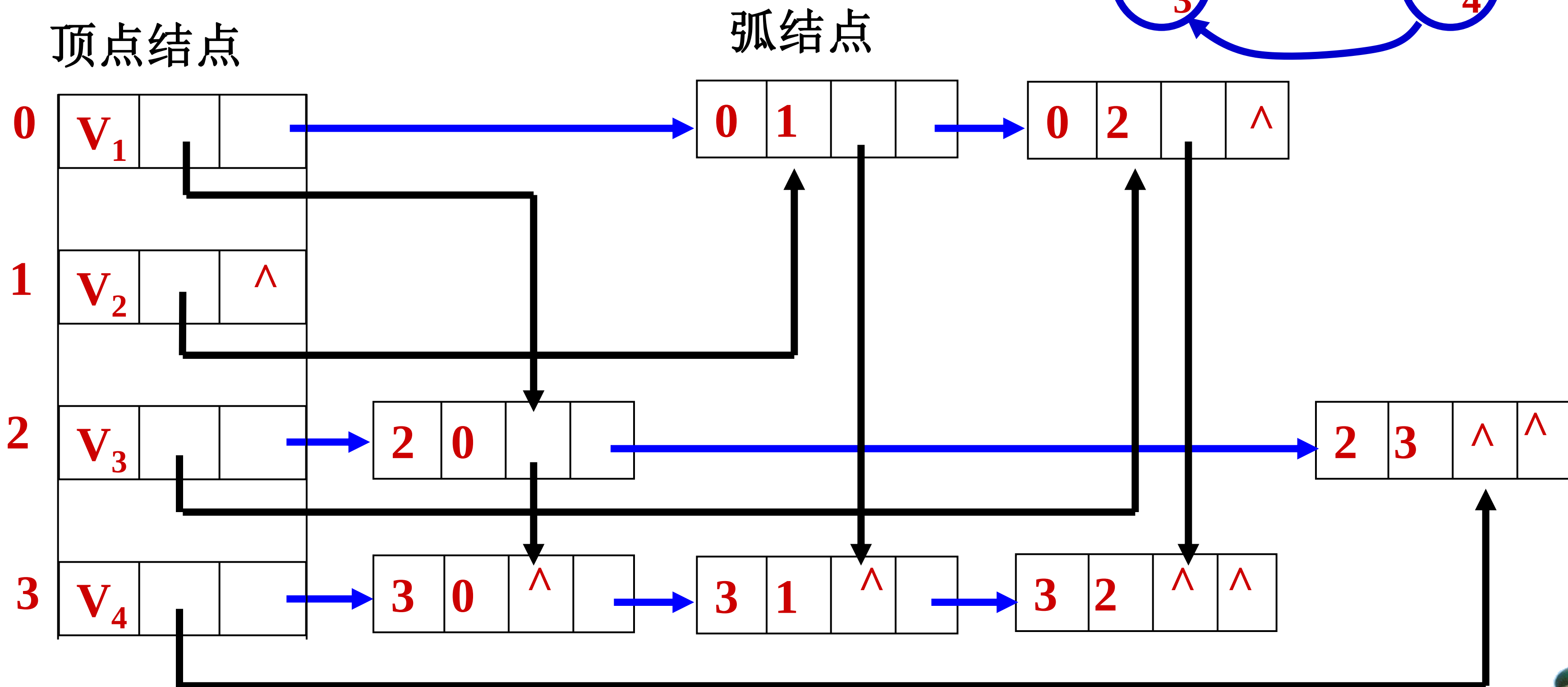
顶点结点结构:

data	firstin	firstout

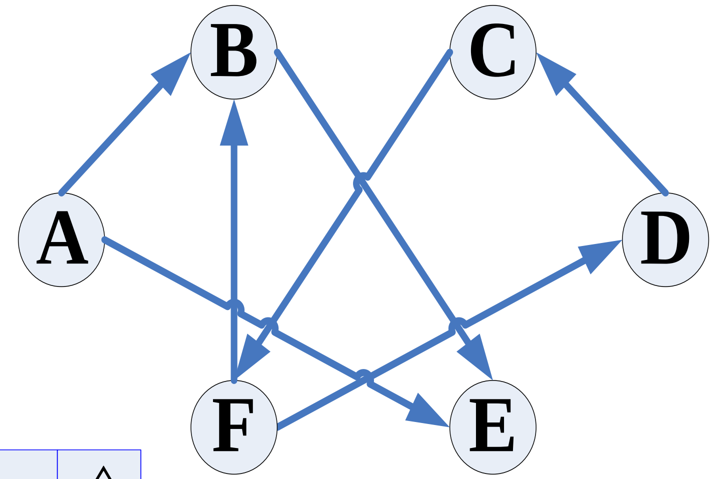
ivex	jvex	jlink	ilink	info

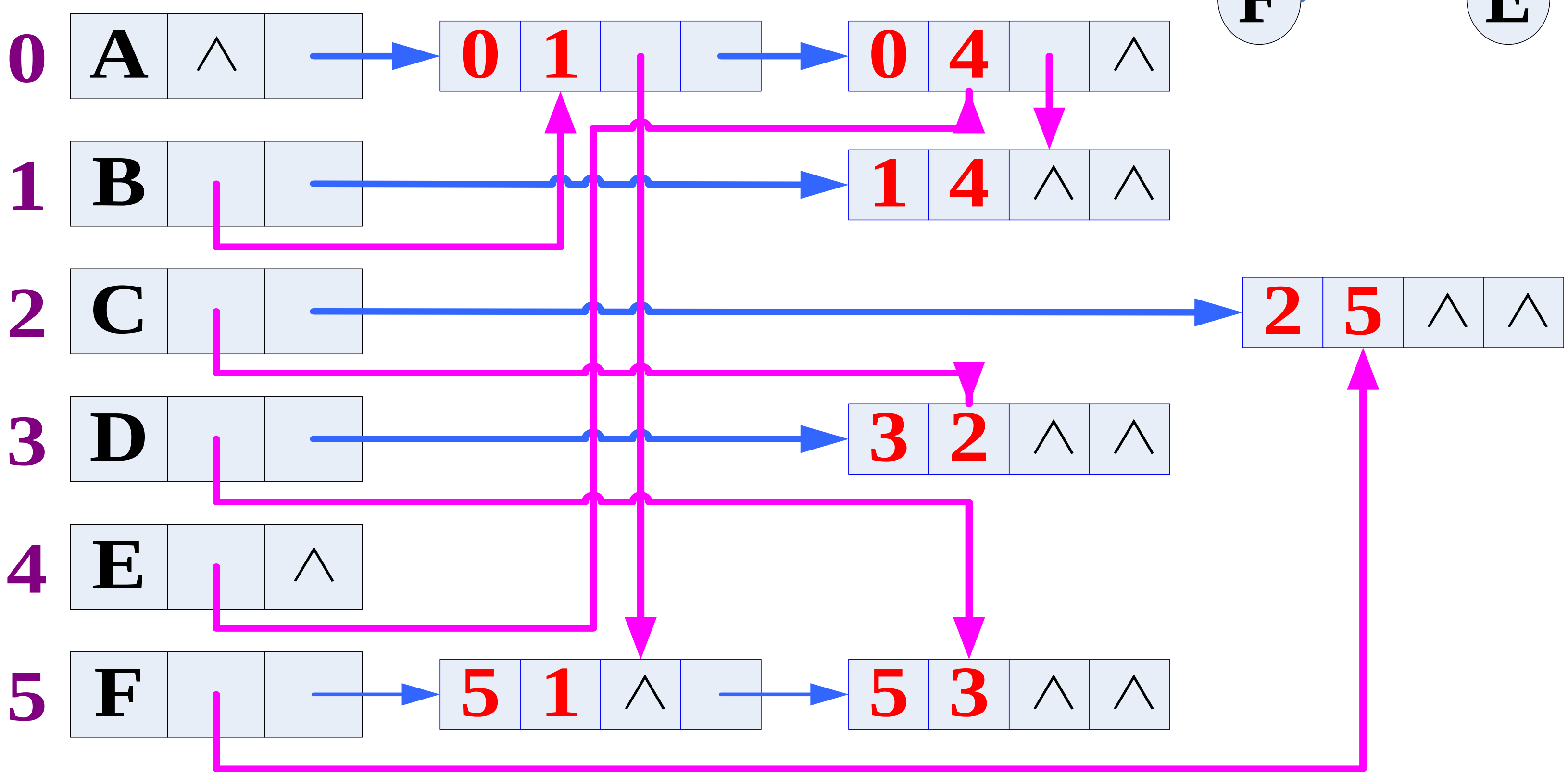
data	firstin	firstout





➢十字链表中既容易找到以v_i为尾的弧，也容易找到以v_i为头的弧，因而容易求得顶点的出度和入度。





◆ 邻接多重表(无向图)

- 邻接多重表是无向图的另一种链式表示结构。
- 和十字链表类似。邻接多重表中，每一条边用一个结点表示。
- 在邻接多重表中有两种结点：
  - 边结点：存储每一条边的信息，用链表链接在一起。

边结点结构:

mark	ivex	ilink	jvex	jlink	info

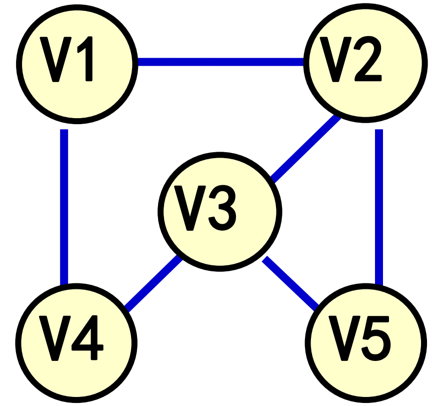
  - **mark:**标识边是否被搜索过
  - 顶点结点：存储每一个顶点的信息，使用一维数组来存储。

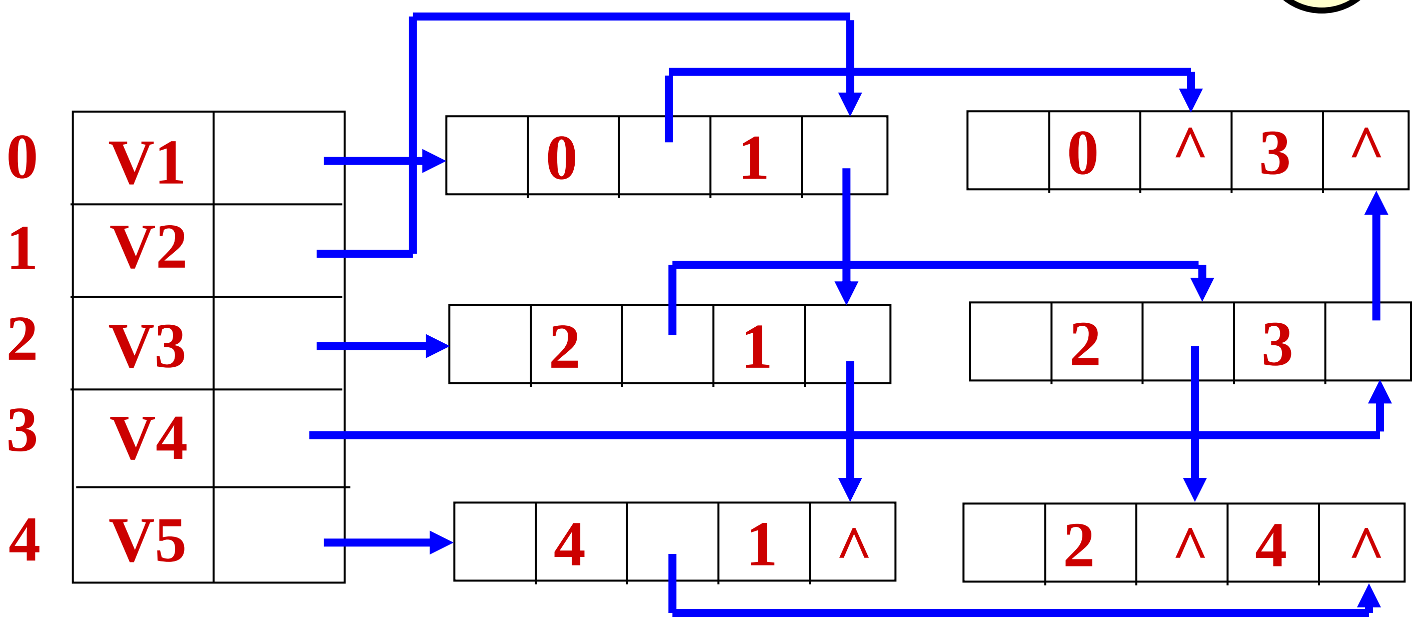
顶点结点结构:

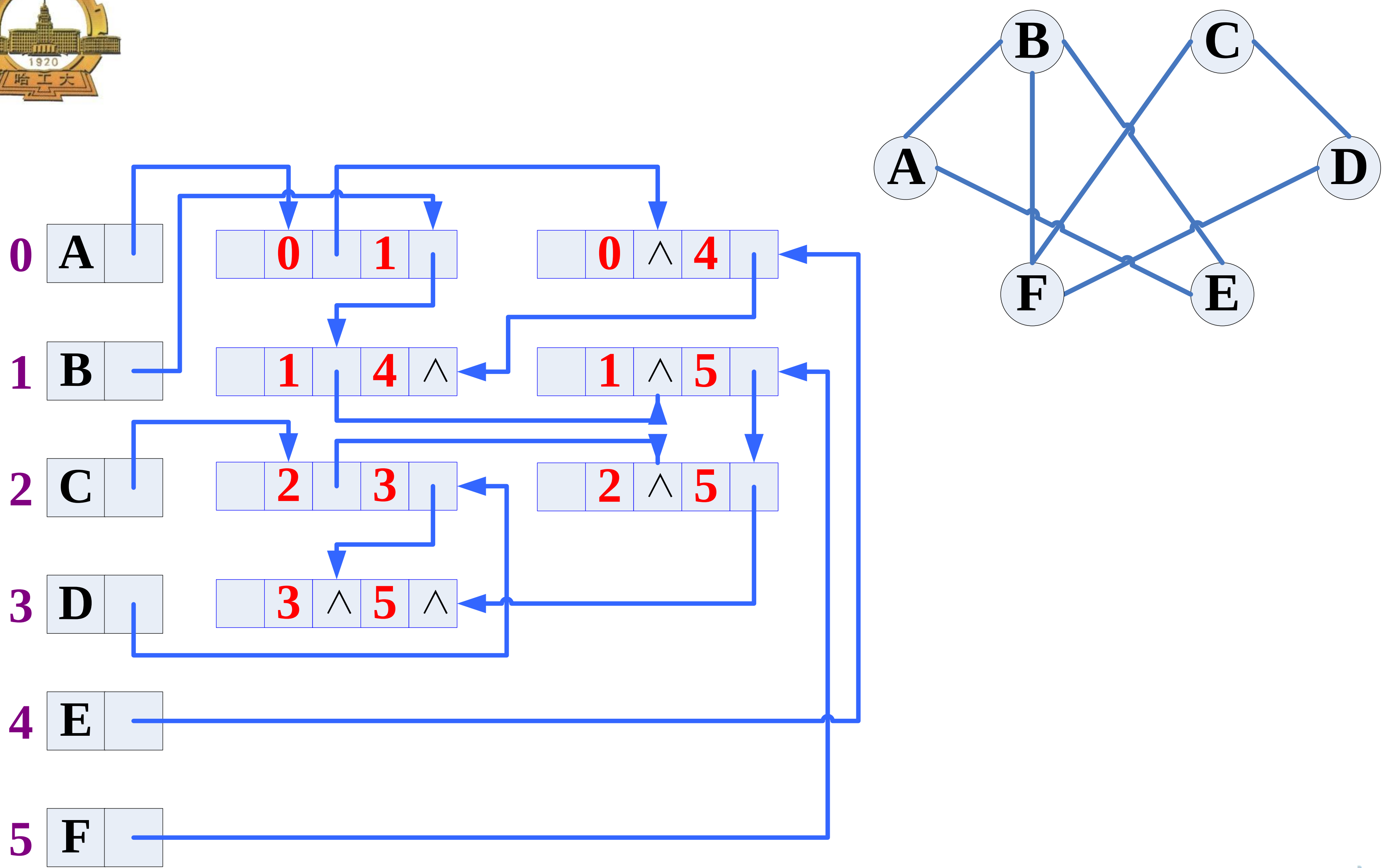
data	firstedge

mark	ivex	ilink	jvex	jlink	info

data	firstedge







0 A
1 B
2 C
3 D
4 E
5 F
0 1
0 ∧ 4
1 4 ∧
1 ∧ 5
2 3
2 ∧ 5
3 ∧ 5 ∧
B C
A D
F E