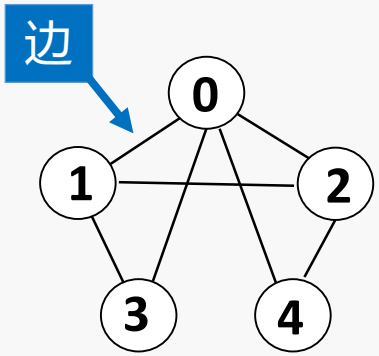
问题求解与实践
——图结构

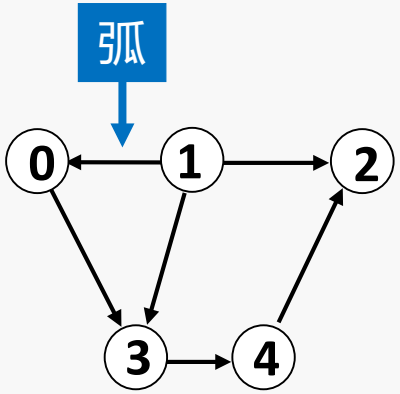
主讲教师： 陈雨亭、沈艳艳

图的基本概念

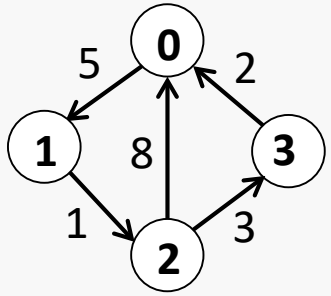
- 图结构来源于生活中诸如通信网、交通网之类的事物，它表现了数据对象间多对多的联系
- 在该结构中，数据元素一般称为顶点
- **图**是由顶点集合及顶点间的关系集合组成的一种数据结构。一般记作（V, E）。其中V是顶点的有限集合；E是顶点之间关系的有限集合



无向图



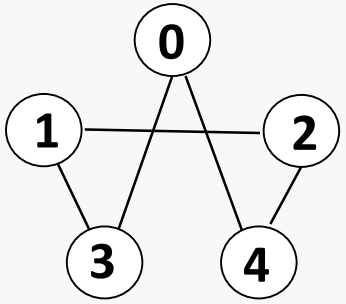
有向图



有权图(网络)

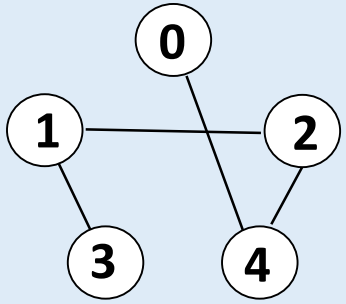
图的基本概念

- 路径：若从顶点v_i出发，沿一些边或弧，经过顶点 v_{p1}，v_{p2}，…，v_{pm} 到达顶点 v_j。则顶点序列(v_i，v_{p1}，…，v_{pm}，v_j)为从顶点 v_i 到顶点 v_j 的路径
- 路径长度：非带权图的路径长度是指此路径上边或弧的条数，带权图的路径长度是指路径上各边或弧的权之和
- 连通图：在无向图中，若从顶点 v_i 到 v_j 有路径，则顶点 v_i 与 v_j 是连通的。如果图中任意一对顶点都是连通的，则称此图是连通图
- 生成树：在无向图中，一个连通图的生成树是它的极小连通子图

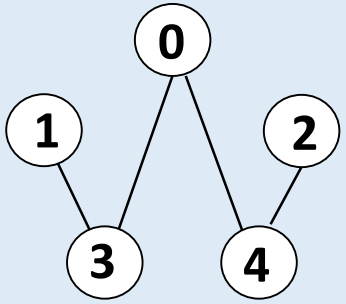


连通图

(1，3，0，4) 或
(1，2，4)
都是 1 到 4 的路径



生成树1



生成树2

图的存储方式

- 图的存储形式有多种，无论哪种形式都要存储两方面的信息：
  1. 顶点信息
  2. 顶点间的关系信息

- 图的存储形式最常见的有：
  - 邻接矩阵
  - 邻接表

图的存储方式——邻接矩阵

◆ 要点：

1. 利用一维数组存储顶点信息
2. 利用二维数组存储顶点间边或弧的信息。此二维数组称**邻接矩阵**

➢ 对于无向图G=(V,E)，邻接矩阵 A 的元素：

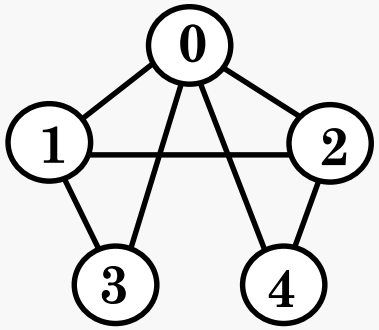
$$A[i][j]=\begin{cases}1 & \text{当}v_i\text{到}v_j\text{有边或弧直连}\\ 0 & \text{其它}\end{cases}$$

➢ 对于带权的图，邻接矩阵 A 的元素：

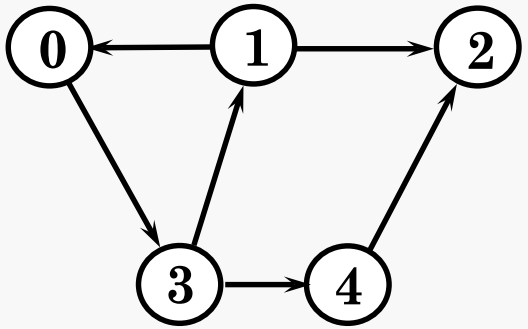
$$A[i][j]=\begin{cases}W(i,j) & \text{当}v_i\text{到}v_j\text{有边或弧直连}\\ \infty & \text{其它}\end{cases}$$

其中 $W(i,j)$ 是与边或弧相关的权

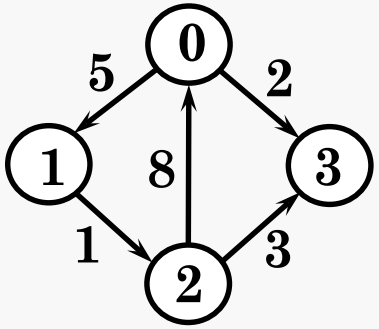
图的存储方式——邻接矩阵示例



(a)无向图



(b)有向图



(c)网络

$$A=\begin{array}{c}\begin{matrix}0&1&2&3&4\end{matrix}\\ \begin{bmatrix}0&1&1&1&1\\1&0&1&1&0\\1&1&0&0&1\\1&1&0&0&0\\1&0&1&0&0\end{bmatrix}\end{array}\begin{matrix}0\\1\\2\\3\\4\end{matrix}$$

(a)无向图邻接矩阵

$$A=\begin{array}{c}\begin{matrix}0&1&2&3&4\end{matrix}\\ \begin{bmatrix}0&0&0&1&0\\1&0&1&0&0\\0&0&0&0&0\\0&1&0&0&1\\0&0&1&0&0\end{bmatrix}\end{array}\begin{matrix}0\\1\\2\\3\\4\end{matrix}$$

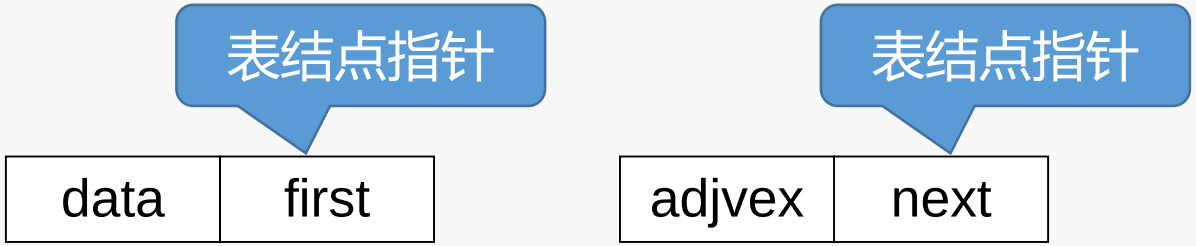
(b)有向图邻接矩阵

$$A=\begin{array}{c}\begin{matrix}0&1&2&3\end{matrix}\\ \begin{bmatrix}\infty&5&\infty&2\\\infty&\infty&1&\infty\\8&\infty&\infty&3\\\infty&\infty&\infty&\infty\end{bmatrix}\end{array}\begin{matrix}0\\1\\2\\3\end{matrix}$$

(c)网络邻接矩阵

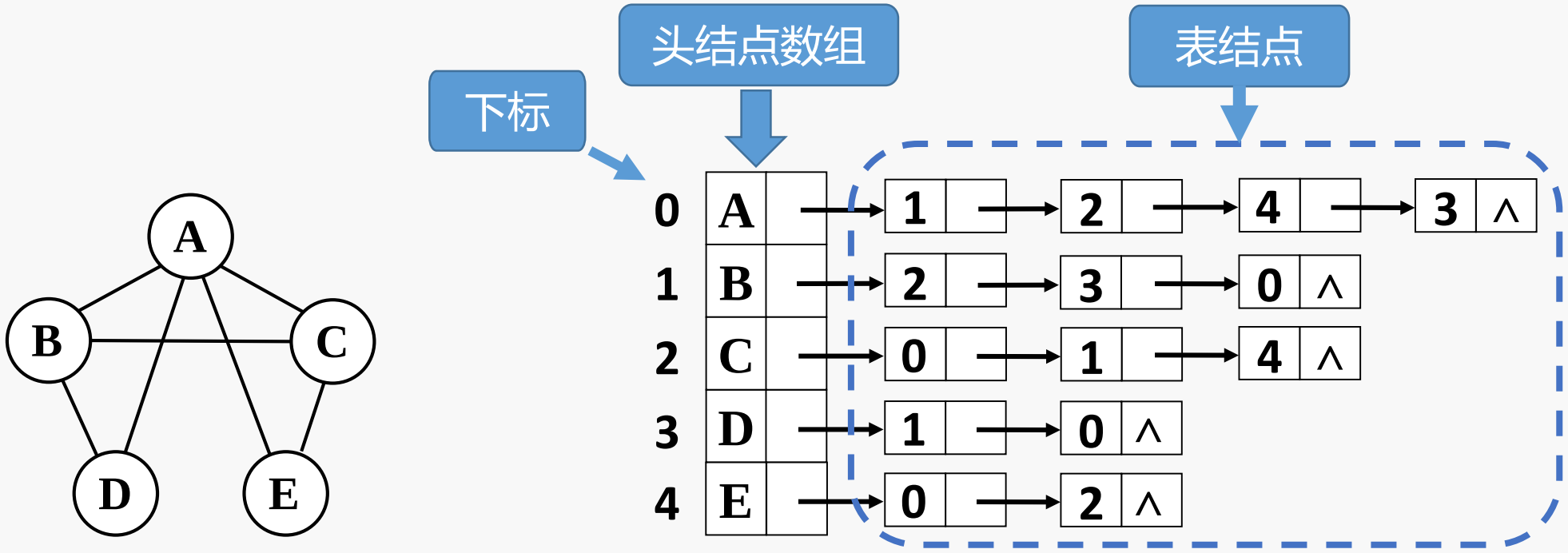
图的存储方式——邻接表

- 邻接表是数组与链表结合的存储形式
  - 邻接表中有两种结点，**头结点**和**表结点**
  - 每个头结点存储一个顶点的信息，一般头结点都存放在一个数组中
  - 对于某个顶点而言，需要将它的邻接点存储为表结点形式，并将它们链接成单链表，这个单链表就称为该顶点的邻接表
  - 每个头结点后面连接其对应的邻接表



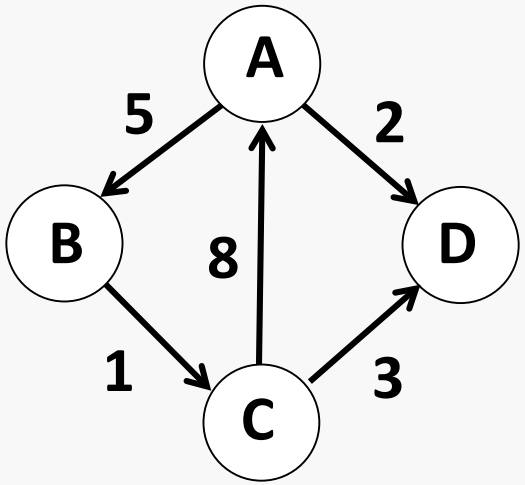
(a)头结点　　(b)无权图的表结点

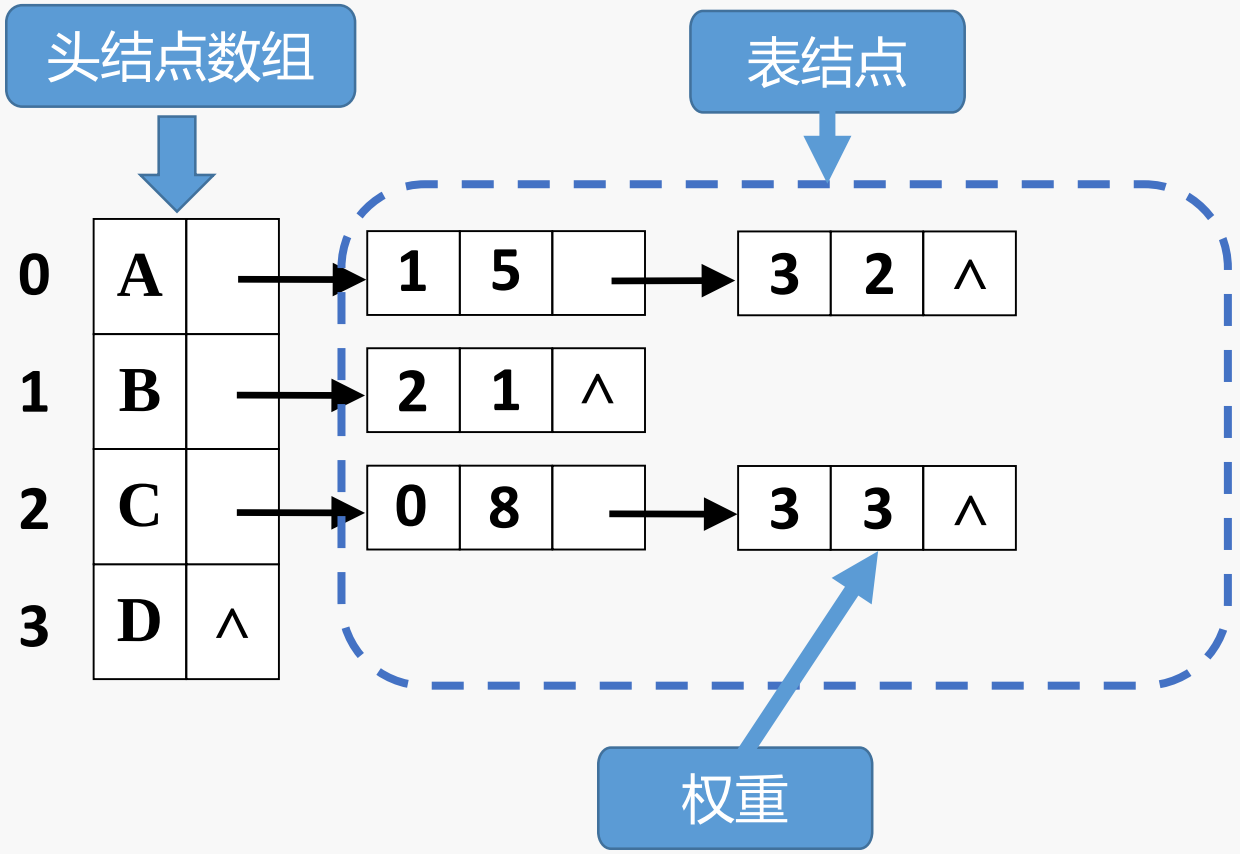
图的存储方式——邻接表示例



无向图的邻接表

图的存储方式——邻接表示例





网络的邻接表

问题求解与实践
——图的遍历

主讲教师： 陈雨亭、沈艳艳

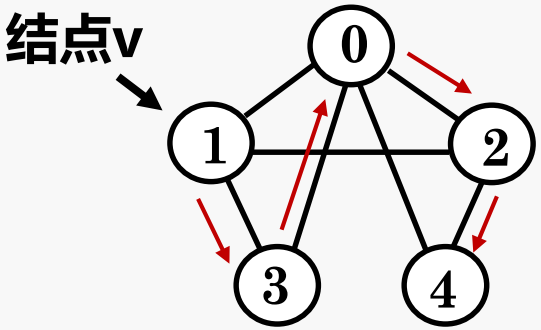
图的遍历

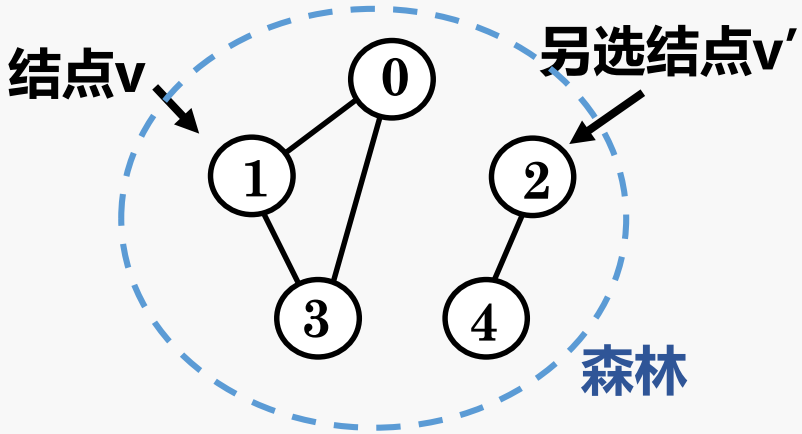
- 图的遍历是指从图的某个顶点出发访问图中所有顶点，并且使图中的每个顶点仅被访问一次的过程
- 图的遍历算法主要有深度优先搜索和广度优先搜索两种

深度优先搜索遍历

◆ **图的深度优先搜索遍历过程：**

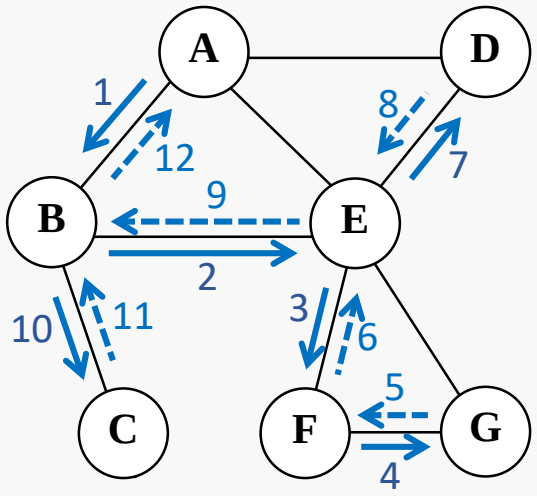
- 假定从图中某个顶点v 出发进行遍历，则首先访问此顶点；
- 然后依次**从**v的各个**未被访问的邻接点出发**，**从该结点继续执行**深度优先搜索，直至图中所有和v有路径相通的顶点都被访问到；
- 若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。

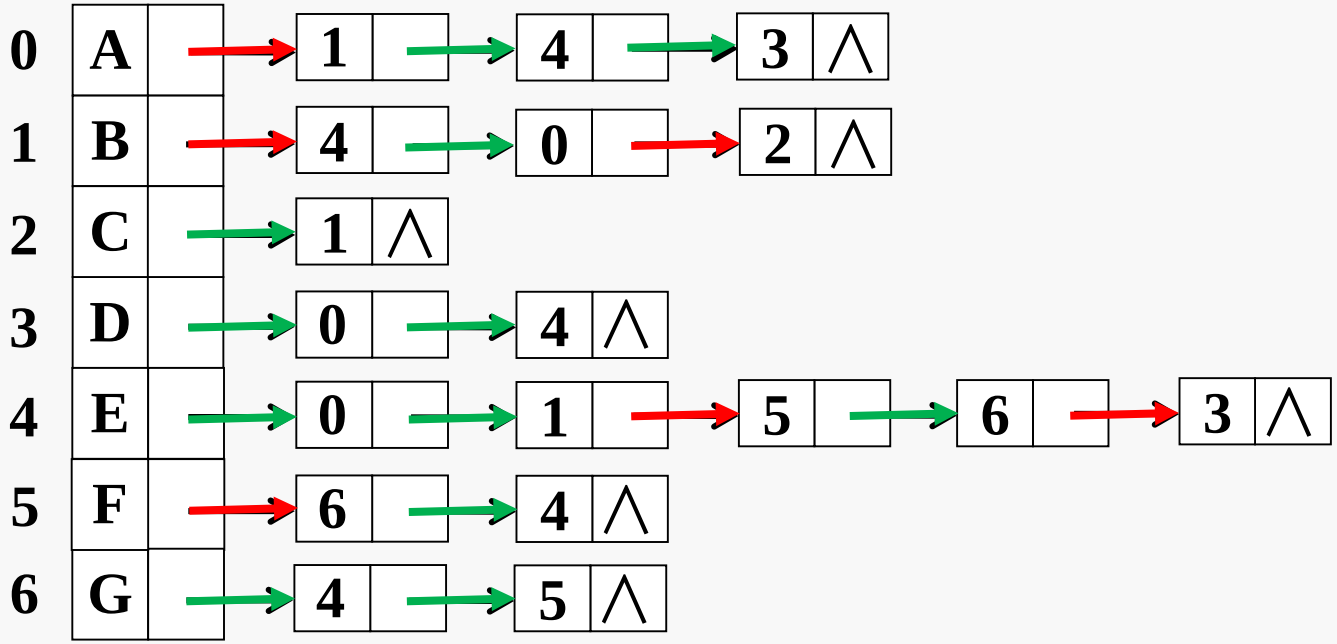




深度优先搜索遍历

◆ **基于邻接表的深度优先搜索过程：**





A B E F G D C

深度优先搜索遍历

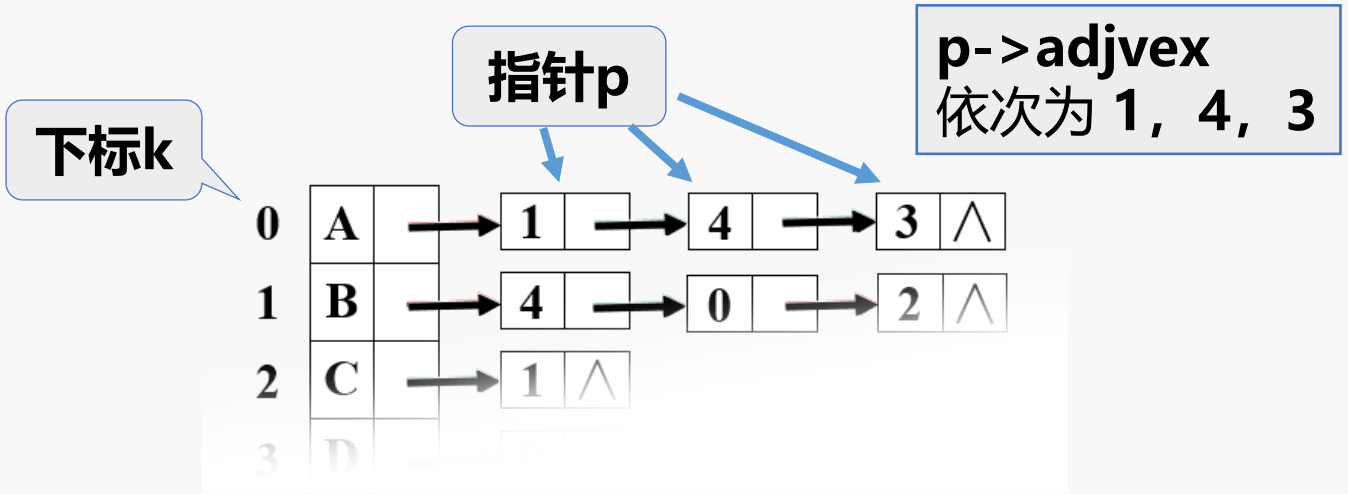
◆ **基于邻接表的深度优先搜索算法实现要点：**

```
DeepSearch( 下标k )                    //从 k 号顶点开始搜索
{
    visit（k 号下标顶点）；
    沿着 k 号顶点邻接表搜索未被访问的结点p, 若p存在，则
        DeepSearch( p->adjvex )；   //这里 p 为表结点指针
}
```

是一个循环过程

未被访问的结点p ：

是指该表结点中数字对应下标的顶点未被访问



深度优先搜索遍历

◆ **基于邻接表的深度优先搜索算法实现要点：**

```
DeepSearch( 下标 k  )          //从下标为 k 的顶点开始搜索
{
    访问顶点数组元素G[k];      //访问下标为k的顶点数据
    p = G[k].first;            // k 号顶点的第一个邻接点
    while( p != NULL)
    {   DeepSearch( p->adivex) ;     // 从 p->adivex 顶点开始搜索
        p = p->next;                 // 寻找下一个邻接点
    }
}
```

p 可能不为空，但其指向的顶点已经被访问过了，所以仅仅p != NULL是不够的

深度优先搜索遍历

定义V[]，初始全为0；

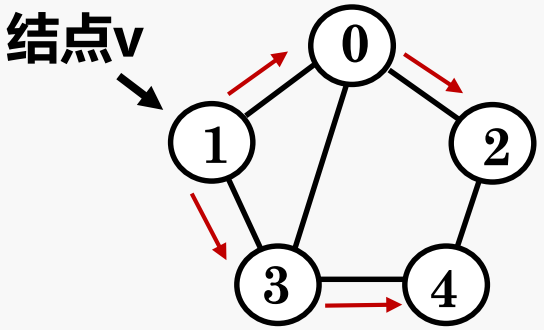
V[k]表示顶点k是否被访问过，0-未访问，1-已访问

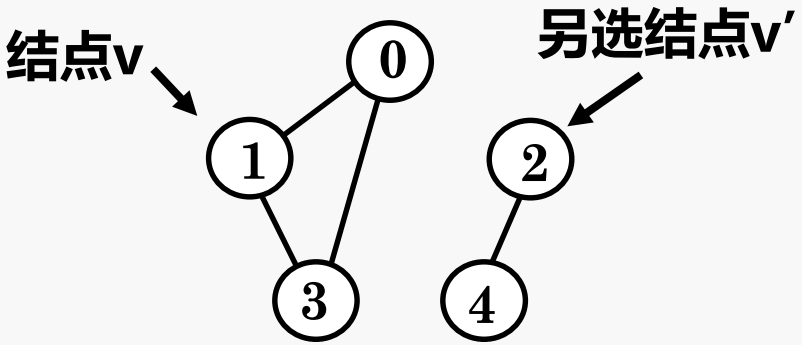
```
DeepSearch( 下标 k  )              //从下标为 k 的顶点开始搜索
{
    访问顶点数组元素G[k];          //访问下标为k的顶点数据
    V[k]=1;
    p = G[k].first;                // k 号顶点的第一个邻接点
    while( p != NULL && V[p->adjvex]==0 )
    {
        DeepSearch( p->adivex) ;   // 从 p->adivex 顶点开始搜索
        p = p->next;               // 寻找下一个邻接点
    }
}
```

广度优先搜索遍历

◆ 图的广度优先搜索遍历过程：

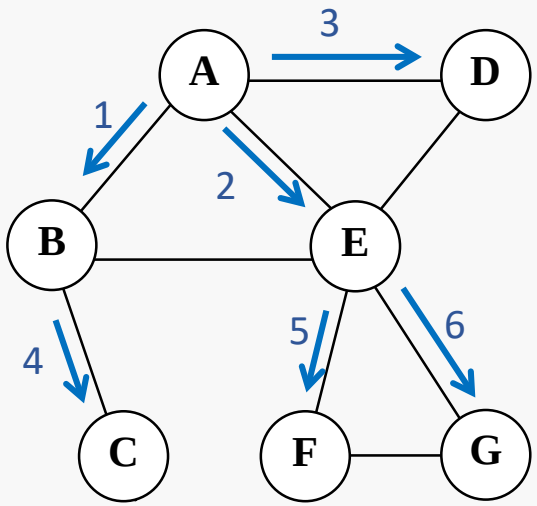
- 假定从图中某个顶点v 出发进行遍历，则首先访问此顶点；
- 再依次访问v的**所有未被访问过的邻接点**，然后按这些顶点被访问的先后次序**再依次访问它们的邻接点**，直至图中所有和v有路径相通的顶点都被访问到；
- 若此时图中尚有顶点未被访问，则另选图中一个未曾被访问的顶点作起始点，重复上述过程，直至图中所有顶点都被访问到为止。

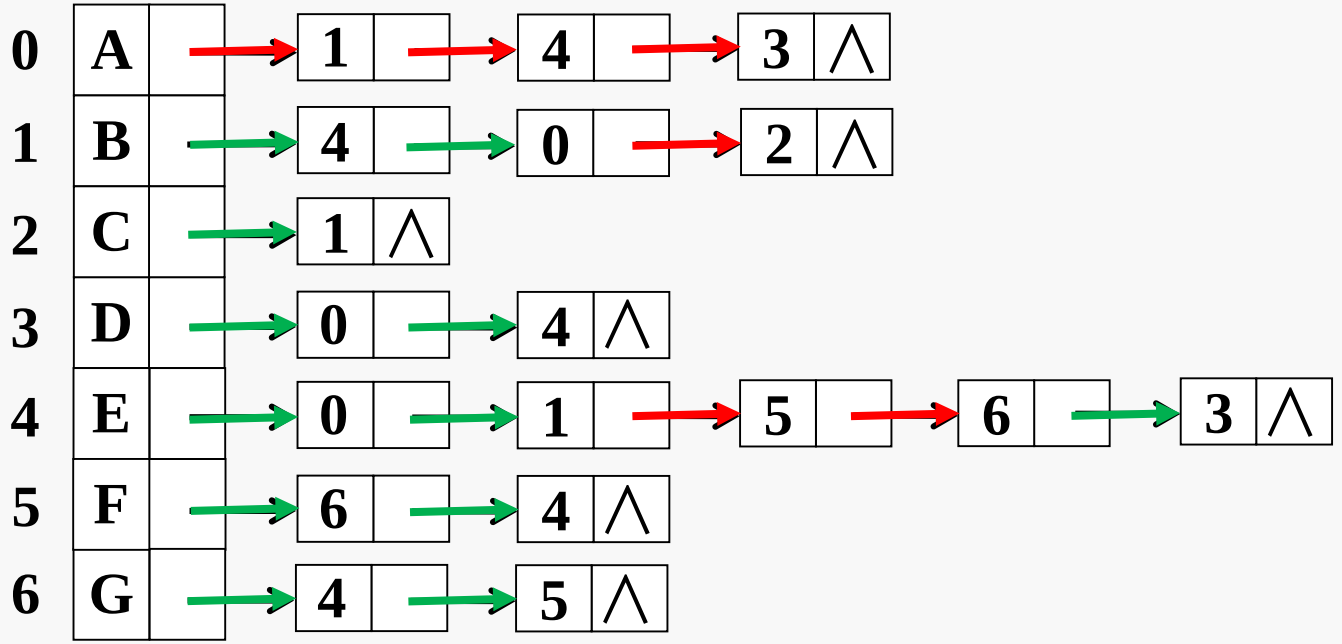




广度优先搜索遍历

◆ **基于邻接表的广度优先搜索过程：**

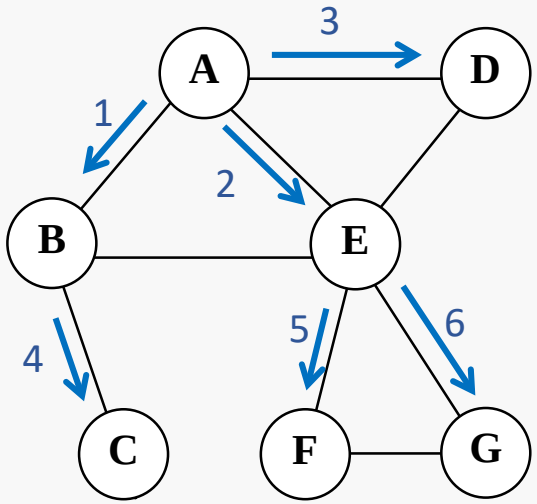




访问：

广度优先搜索遍历

- **基于邻接表的广度优先搜索过程：**



队列：

A	B	E	D	C	F	G

访问：

A	B	E	D	C	F	G

广度优先搜索遍历

- **广度优先搜索算法实现要点：**

```
定义整数队列 queue;              // 用于存放表示顶点G[k]的下标k
定义V[]，初始全为0;              // V[k]表示顶点k是否被访问过
将 0 放入queue;                  // 假设从 0 号结点开始遍历
while( queue 不空 ) {            // 队空则搜索结束
    元素从 queue 中出队放入k;        //出队列
    访问顶点 G[k];    V[k]=1;        //0-未访问，1-已访问
    p = G[k].first;              // k 号顶点的第一个邻接点的指针
    while( p != NULL )
    {   若V[p->adjvex]=0 则将 p->adjvex 入队列;
        p = p->next;
    }
}
```