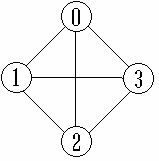
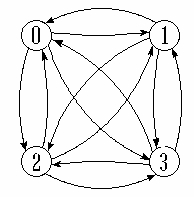
# 第六章知识小结

一、基础（总结了容易被忽略的点）

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

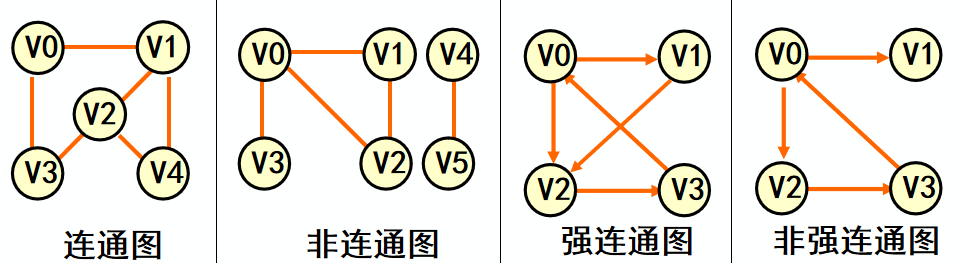
有向完全图

n(n-1) 条边

连通图:在无向图G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。

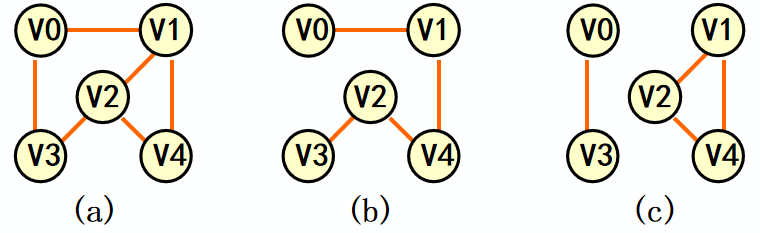
强连通图:在有向图G中，若对任何两个顶点 v、u 都存在从v 到 u 的路径，则称G是强连通图。

举个（强）连通图的栗子：



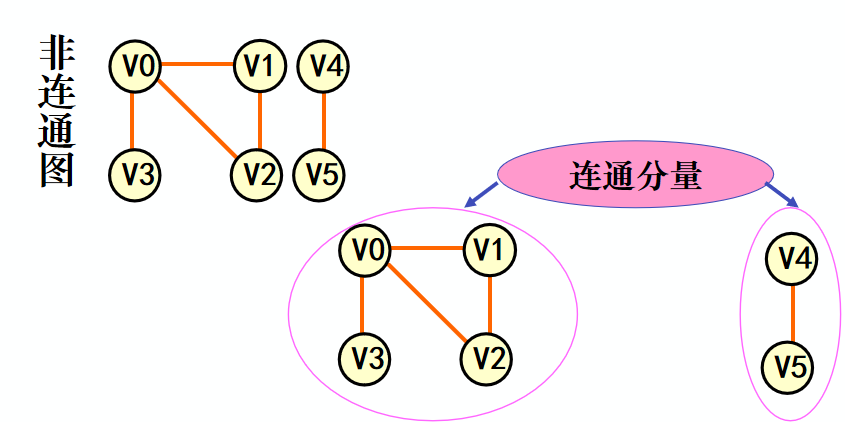
子图：设有两个图G=（V，E）、G1=（V1，E1），若V1 V，E1  E，则称 G1是G的子图。

举个栗子：(b)、(c) 是 (a) 的子图



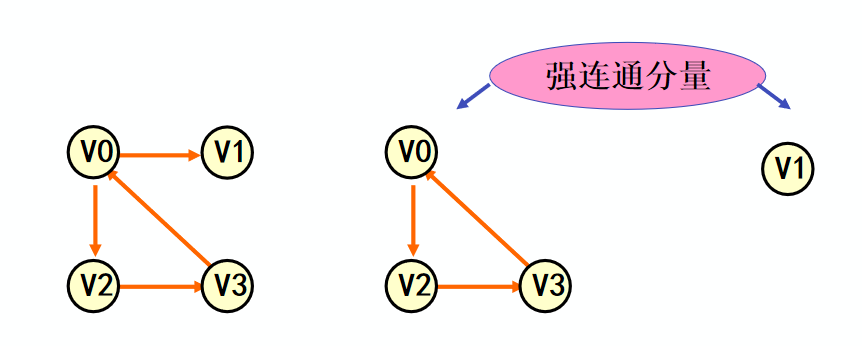
连通分量：无向图G 的极大连通子图称为G的连通分量。

极大连通子图：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。



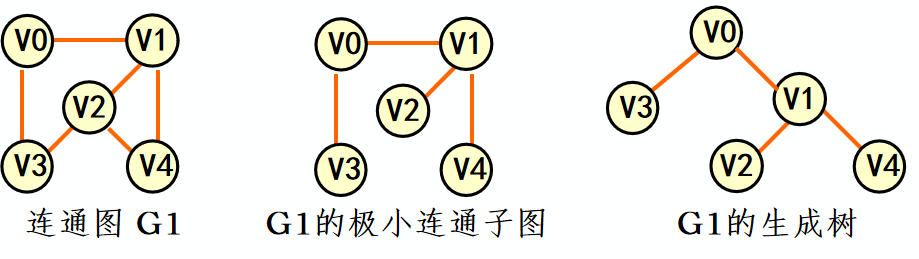
强连通分量：有向图G的极大强连通子图称为G的强连通分量。

极大强连通子图：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。



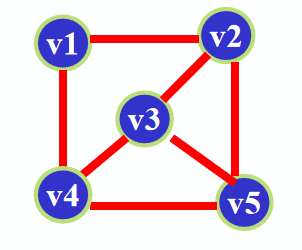
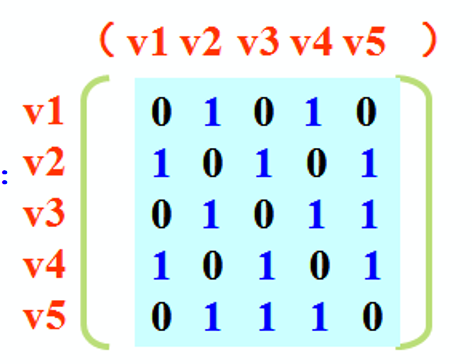
1. 图的存储结构

1、邻接矩阵（数组）表示法

邻接矩阵：表示顶点之间相邻关系的矩阵。

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 [n][n]，定义为：

无向图的邻接矩阵表示法

邻接矩阵

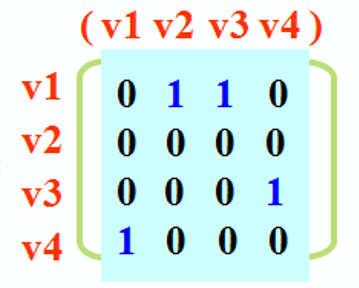
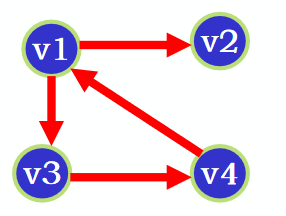
总结：

1.无向图的邻接矩阵是对称的；

2.顶点i 的度＝第 i 行 (列) 中1 的个数；

3.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



邻接矩阵

总结

1.第i行含义：以结点vi为尾的弧(即出度边）

2.第i列含义：以结点vi为头的弧(即入度边）

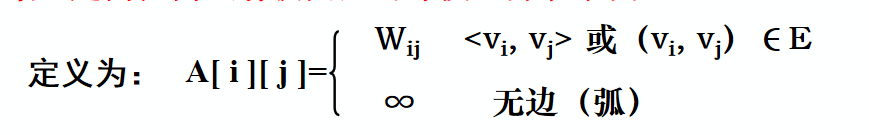
3.有向图的邻接矩阵可能是不对称的。

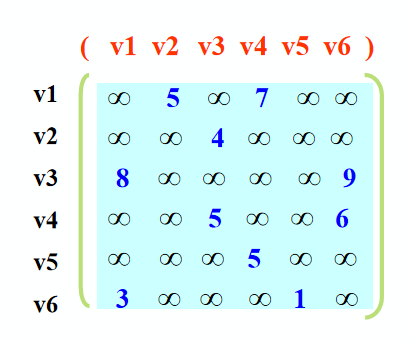
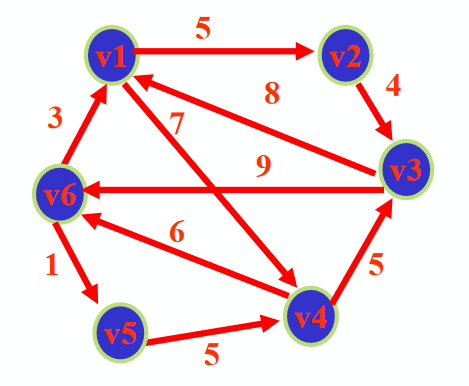
4.顶点的出度=第i行元素之和

5.顶点的入度=第i列元素之和

6.顶点的度=第i行元素之和+第i列元素之和

若G是网，网（有权图）的邻接矩阵表示法





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。

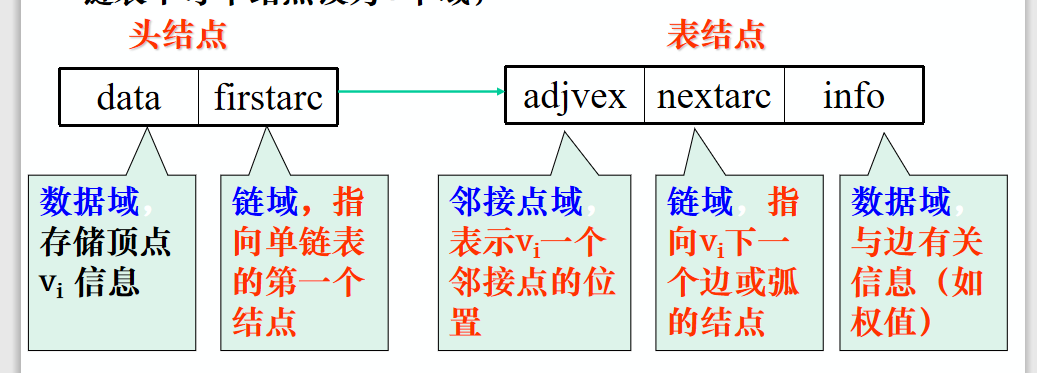
缺点：n个顶点需要n\*n个单元存储边;空间效率为O(n2)。 对稀疏图而言尤其浪费空间。

1. 邻接表表示法

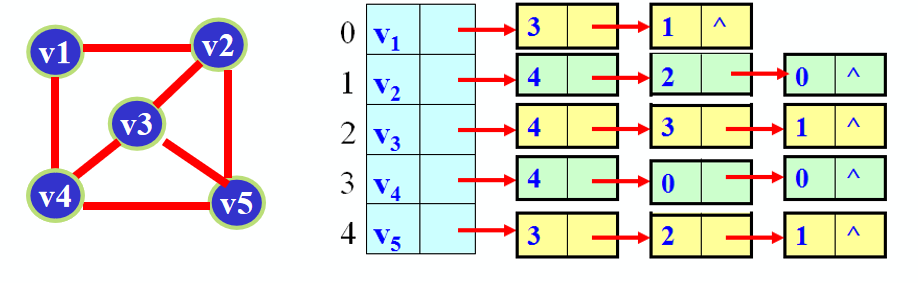
(1)对每个顶点vi 建立一个单链表，把与vi相邻接的顶点放在这个链表中每个结点设为3个域。

(2)每个单链表有一个头结点（设为2个域），存vi信息；

(3)每个单链表的头结点另外用顺序存储结构存储。



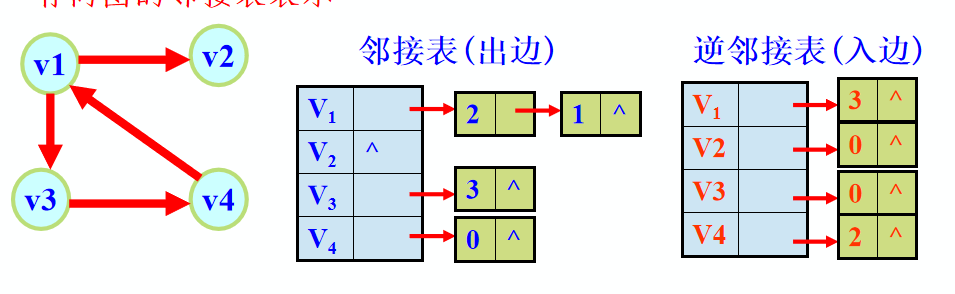
无向图的邻接表表示



1.邻接表不唯一，因各个边结点的链入顺序是任意的。

2.TD(Vi)=单链表中链接的结点个数。

有向图的邻接表表示



1.出度OD(Vi)＝单链出边表中链接的结点数

2.入度ID(Vi)＝邻接点域为Vi的弧个数

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

邻接表表示法的特点

优点：

（1）空间利用率高。

（2）容易寻找顶点的邻接点。

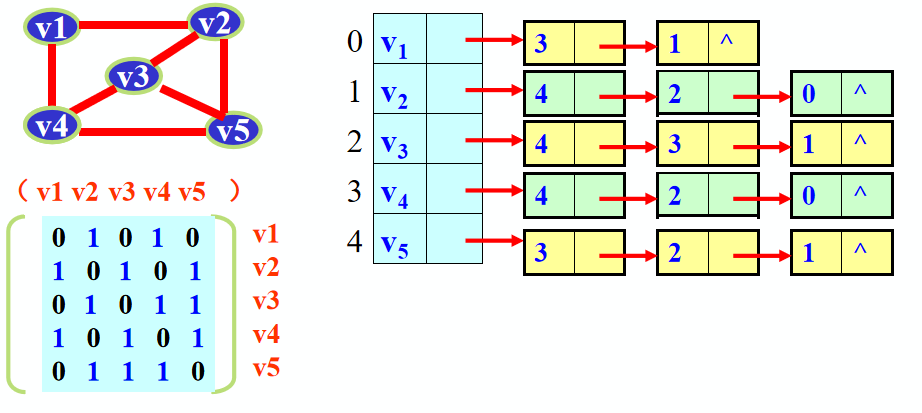
（3）便于统计变得数目。

缺点：

（1）不便于判断两点之间是否有边。判断两顶点间是否有边或弧，需搜索两结点对应的单链表，没有邻接矩阵方便。

（2）不便于计算有向图各个顶点的度。

邻接矩阵与邻接表表示法的关系



1. 联系：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 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）。

② 邻接矩阵的空间复杂度为O(n2),而邻接表的空间复杂度为O(n+e)或者O(n+2e) 。

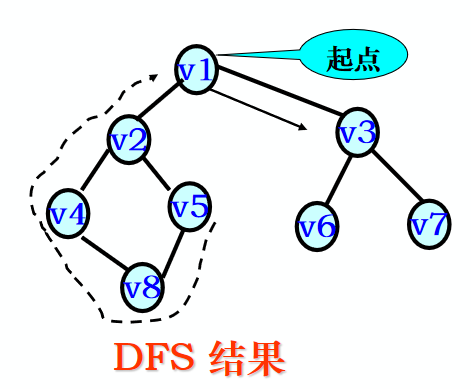
3. 用途：邻接矩阵多用于稠密图；而邻接表多用于稀疏图。

三、图的遍历

1、深度优先搜索（基本思想：——仿树的先序遍历过程。）

深度优先搜索( DFS － Depth\_First Search)

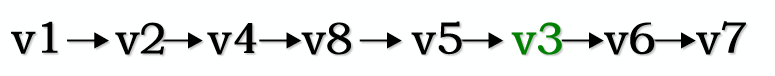
）



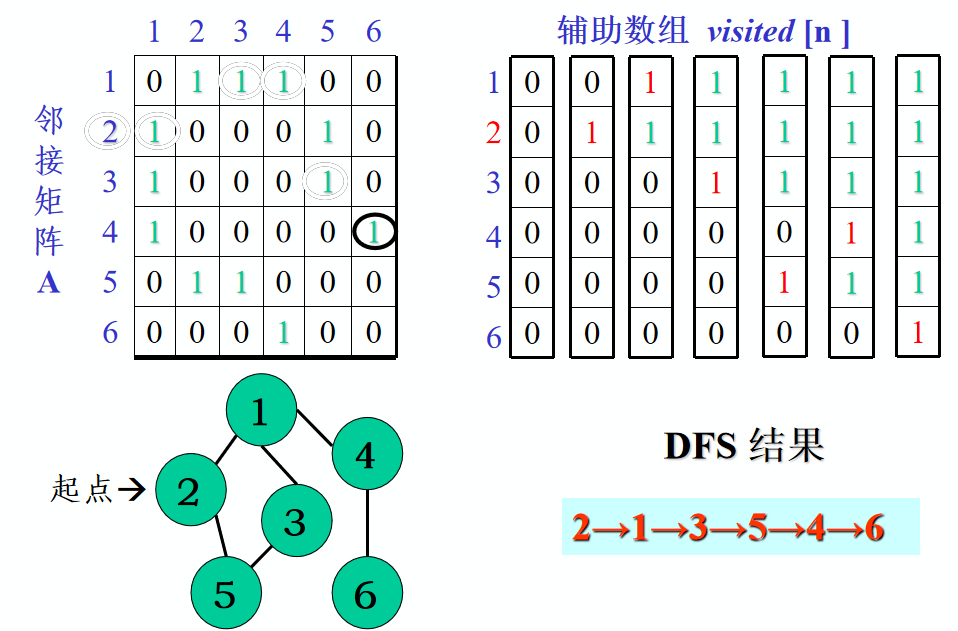
1.访问起始点v;

2.若v的第1个邻接点没访问过，深度遍历此邻接点；

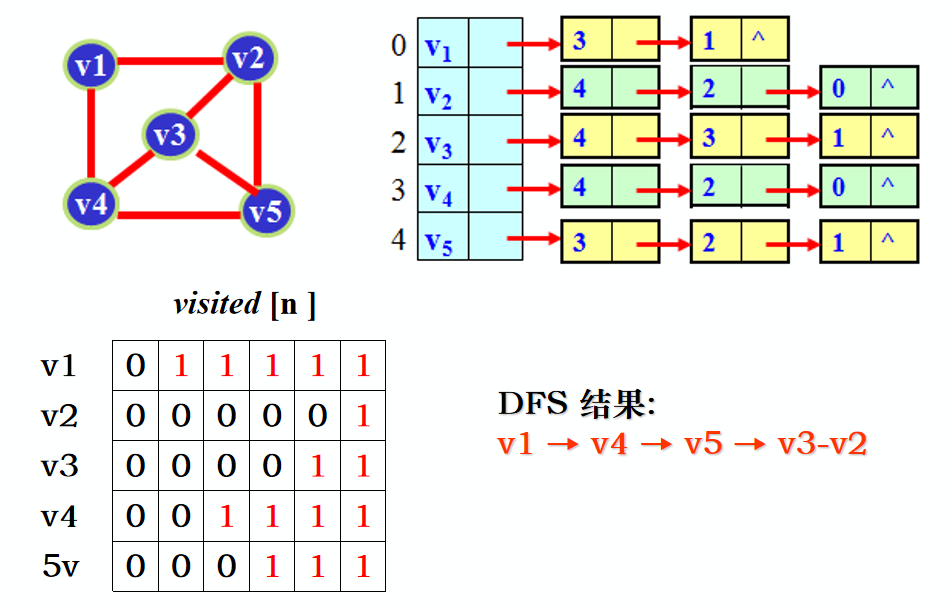
3.若当前邻接点已访问过，再找v的第2个邻接点深度遍历。



利用邻接矩阵实现DFS



利用邻接表进行DFS



DFS算法效率分析

(1)用邻接矩阵来表示图，遍历图中每一个顶点都要从头扫描该顶点所在行，时间复杂度为O(n2)。

(2)用邻接表来表示图，虽然有 2e 个表结点，但只需扫描 e 个结点即可完成遍历，加上访问 n个头结点的时间，时间复杂度为O(n+e)。

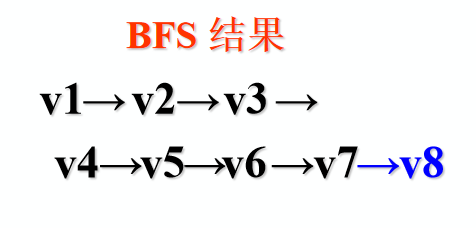
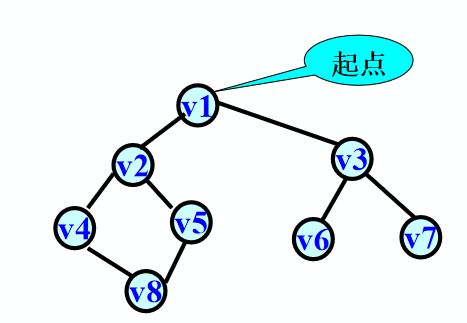
结论：

(1)稠密图适于在邻接矩阵上进行深度遍历；

(2)稀疏图适于在邻接表上进行深度遍历。

2、 广度优先搜索(基本思想：——仿树的层次遍历过程)

广度优先搜索( BFS － Breadth\_First Search)



(1)在访问了起始点v之后，依次访问 v的邻接点；

(2)然后再依次访问这些顶点中未被访问过的邻接点；

(3)直到所有顶点都被访问过为止。

BFS算法效率分析

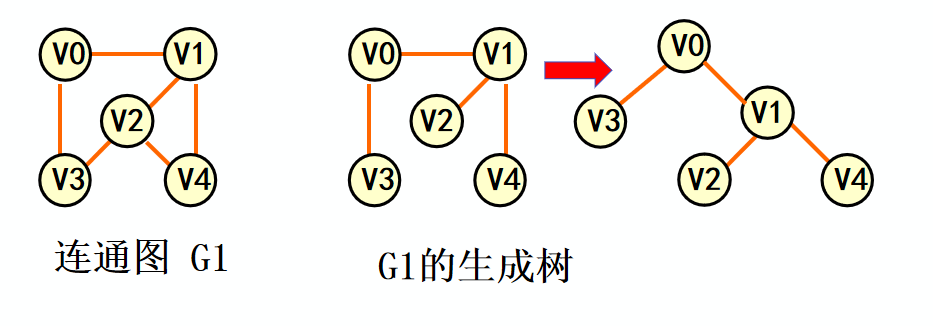
(1)如果使用邻接矩阵，则BFS对于每一个被访问到的顶点，都要循环检测矩阵中的整整一行（n个元素），总的时间代价为O(n2)。

(2)用邻接表来表示图，虽然有2e个表结点，但只需扫描e个结点即可完成遍历，加上访问n个头结点的时间，时间复杂度为O(n+e)。

四、最小生成树

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。

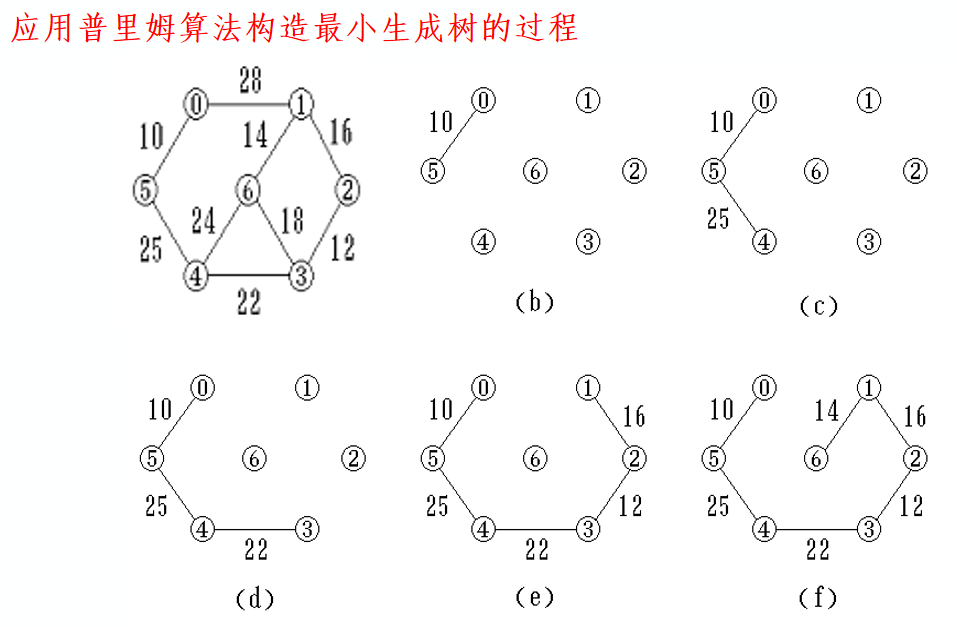
生成树：包含图G所有顶点的极小连通子图（n-1条边）。



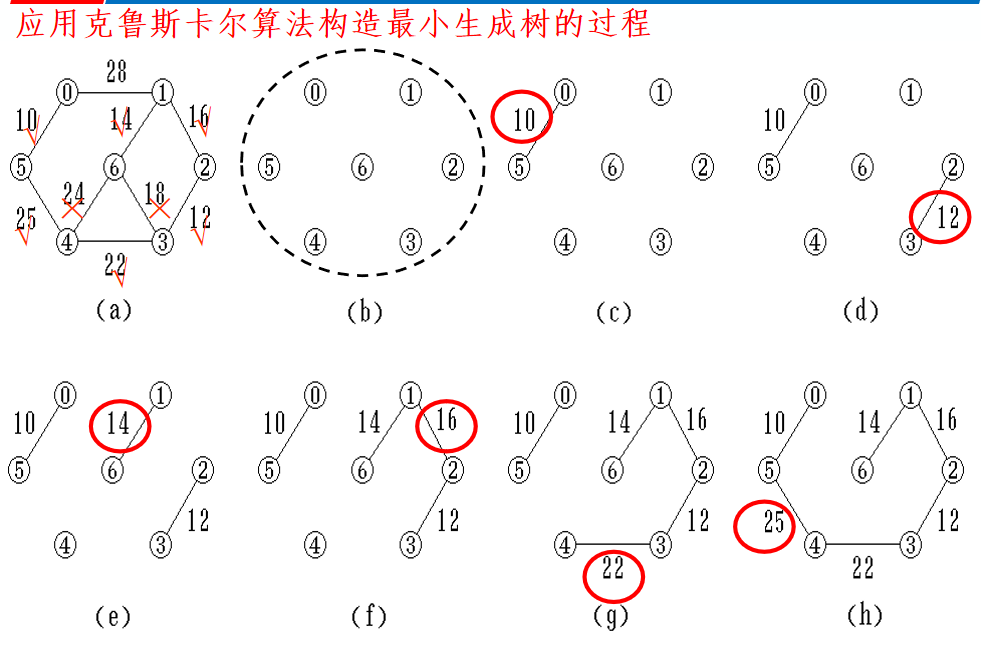
Prim（普里姆）算法: 归并顶点，与边数无关，适于稠密网

Kruskal（克鲁斯卡尔）算法：归并边，适于稀疏网

应用普里姆算法构造最小生成树的过程



应用克鲁斯卡尔算法构造最小生成树的过程

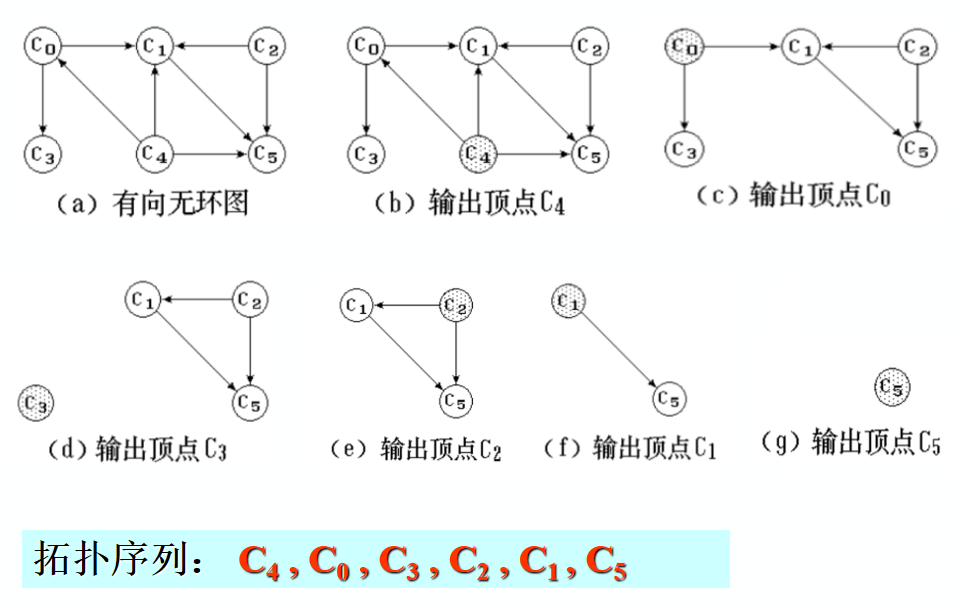


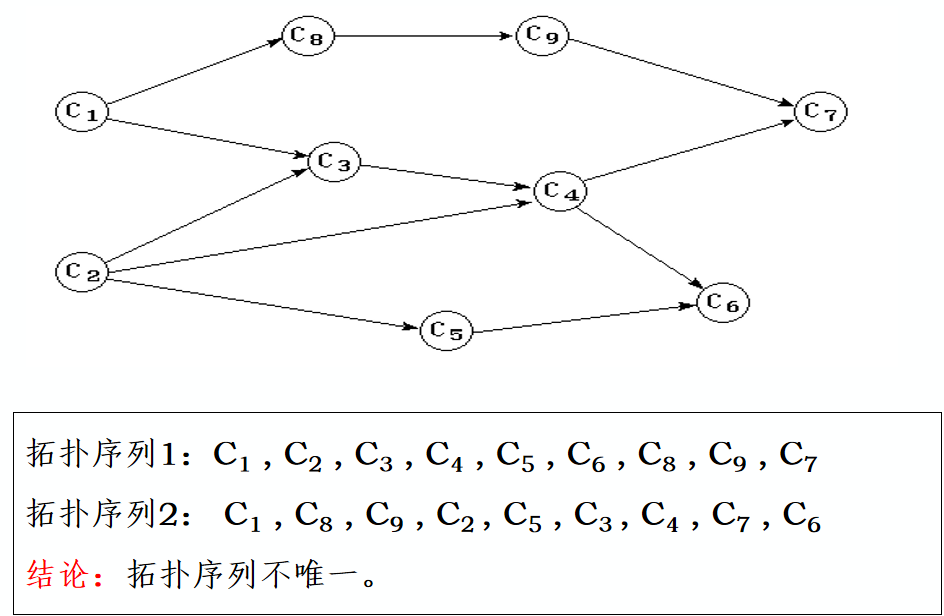
两种常见的最短路径求解算法：

1. Dijkstra（迪杰斯特拉）算法：适用于单源最短路径。

2.Floyd（弗洛伊德）算法：适用于求解所有顶点间的最短路径

拓扑排序





关键路径