

Map

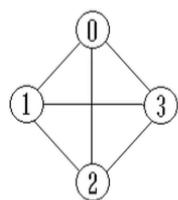
1. 基本属于
2. 存储结构
3. 图的遍历
4. 图的连通性
5. 图的应用

7.1 基本术语

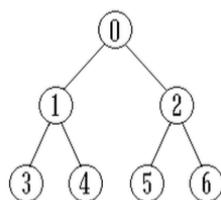
- * $G = (V, E)$
 - * $V = \text{vertex}$
 - * $E = \text{edge}$
- * V 是 G 的顶点集合，是有穷非空集
- * E 是 G 的边集合，是有穷集，可空

- * 有向图：每个 E 都有方向
- * 无向图：每个 E 都无方向
- * 完全图：任意两个 V 都有一条 E
 - * n 个顶点的无向图共 $\frac{n(n-1)}{2}$ 条边，则是无向完全图
 - * n 个顶点的有向图共 $n(n-1)$ 条边，则是有向完全图

例：判断下列4种图形各属什么类型？



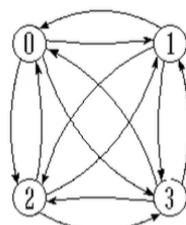
(a) G_1



(b) G_2



(c) G_3



(d) G_4

无向完全图

$n(n-1)/2$ 条边

无向图(树)

有向图

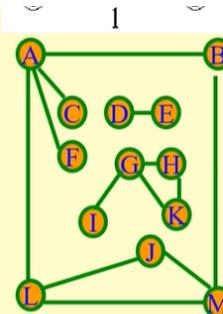
有向完全图

$n(n-1)$ 条边

G_1 的顶点集合为 $V(G_1) = \{0, 1, 2, 3\}$
边集合为 $E(G_1) = \{(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)\}$

- * 稀疏图：边少， $E < V * \log(V)$
- * 稠密图：边数接近 $n * (n-1)$ 或 $n * (n-1) / 2$
- * 子图： $G = (V, E)$, $G' = (V', E')$; 当 V' 在 V 中, E' 在 E 中
- * 带权图：“边, E ” 上带权值
- * 网络：带权图
- * 连通图：
 - 无向图
 - 任意两个顶点间都有路径相通
 - 连通分量：非连通图的极大连通子图
- * 强连通图：
 - 有向图
 - 任意两个顶点之间都存在一条 **有向路径**
 - 强连通分量：强连通子图

连通图：在无向图中，若从顶点 v_i 到顶点 v_j 有路径，则称顶点 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。
非连通图的极大连通子图叫做连通分量。



强连通图：在有向图中，若对于每一对顶点 v_i 和 v_j ，都存在一条从 v_i 到 v_j 和从 v_j 到 v_i 的路径，则称此图是强连通图。

非强连通图的极大强连通子图叫做强连通分量。



- * 生成树：含有全部 n 个 V ，但只有 $n-1$ 条边
 - 任意在生成树上添加一条边，必然构成一个环
 - 若少于 $n-1$ 条边，则必然非连通
- * 生成森林：生成树的集合，且构成的 边 或 弧 **最少**
- * 邻接点：若 (u, v) 是 $E(G)$ 中的一条边，则 u 和 v 互为邻接顶点
- * 弧头、弧尾：1、有向边 (u, v) 为弧；2、从弧头到弧尾
- * 度：与之顶点 v 相关联的边数， $TD(v)$ ；有向图中分出度、入度
 - 出度： $ID(v)$ ，以 v 为终边
 - 入度： $OD(v)$ ，以 v 为始边

- * 路径：从 v_i 出发，到 v_j ，称序列 $(v_i, v_{p1} \dots v_{pn}, v_j)$ 为 v_i 到 v_j 的路径
 - 简单路径：路径顶点不重合（不一定最简）
 - 非简单路径
 - 回路： $v_i = v_j$
- * 路径长度：权值、无权值

图的抽象数据类型

ADT Graph {

数据对象 V: V是具有相同特性的数据元素的集合，称为顶点集。

数据关系 R: R={VR}; VR={<v,w>|v,w∈V 且 P(v,w),

<v,w>表示从v到w的弧，
谓词P(v,w)定义了弧<v,w>的意义或信息}

基本操作P:

CreatGraph (&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。

注意: V 的大小写
含义不同!

InsertVex (&G, v);

初始条件: 图G存在, v和图中顶点有相同特征。

操作结果: 在图G中添加新顶点。

..... (参见教材P156-257)

}ADT Graph

11 

7.2 图的存储结构

- 图的特点: 非线性 (m:n)
- 可以用数组描述元素关系: 邻接矩阵, 关联矩阵
- 链式存储: 多重链表: 邻接表、十字链表、邻接多重表

1. 邻接矩阵(数组)表示法
2. 邻接表(链式)表示法
3. 十字链表表示法
4. 邻接多重表表示法 

- 基本原则: 惟一复原

邻接矩阵 (数组) 表示

- 顶点表、邻接矩阵
- 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][m] = 1连通，0不连通

* A[n][n]的邻接矩阵

* A[v][i] : 从顶点v到顶点i

- 无向图：

例1：下面无向图的邻接矩阵如何表示？

顶点表： $(v_1 \ v_2 \ v_3 \ v_4 \ v_5)$

邻接矩阵： $A.Edge = \begin{matrix} v_1 & \left[\begin{array}{ccccc} 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{array} \right] \\ v_2 & \\ v_3 & \\ v_4 & \\ v_5 & \end{matrix}$

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，其余全1。

13

- 有向图

例2：下面有向图的邻接矩阵如何表示？

顶点表： $(v_1 \ v_2 \ v_3 \ v_4)$

邻接矩阵： $A.Edge = \begin{matrix} v_1 & \left[\begin{array}{cccc} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{array} \right] \\ v_2 & \\ v_3 & \\ v_4 & \end{matrix}$

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*v_i*的出度=第*i*行元素之和， $OD(v_i) = \sum A.Edge[i][j]$ ；
 顶点*v_i*的入度=第*i*列元素之和。 $ID(v_i) = \sum A.Edge[j][i]$

- 权值图：初始化时为无穷大

例3：有权图(即网络)的邻接矩阵如何表示？

定义： $A.Edge[i][j] = \begin{cases} W_{ij} & \langle v_i, v_j \rangle \text{ 或 } (v_i, v_j) \in VR \\ \infty & \text{无边(弧)} \end{cases}$

以有向网为例：

顶点表： $(v_1 \ v_2 \ v_3 \ v_4 \ v_5 \ v_6)$

邻接矩阵： $N.Edge = \begin{matrix} v_1 & \left[\begin{array}{cccccc} \infty & 5 & \infty & 7 & \infty & \infty \\ \infty & \infty & 4 & \infty & \infty & \infty \\ 8 & \infty & \infty & \infty & \infty & 9 \\ \infty & \infty & 5 & \infty & \infty & 6 \\ \infty & \infty & \infty & 5 & \infty & \infty \\ 3 & \infty & \infty & \infty & 1 & \infty \end{array} \right] \\ v_2 & \\ v_3 & \\ v_4 & \\ v_5 & \\ v_6 & \end{matrix}$

邻接矩阵法容易实现图的操作，如：求某顶点的度、判断顶点之间是否有边(弧)、找顶点的邻接点等等。
 邻接矩阵法缺点：
 n个顶点需要n*n个单元存储边(弧)；空间效率为O(n²)。

对稀疏图而言尤其浪费空间。
关联矩阵：顶点和边的关系

基本定义



图的邻接矩阵在机内如何表示? (参见教材P161)

注: 用两个数组分别存储顶点表和邻接矩阵

```
#define INFINITY INT_MAX      //最大值∞  
#define MAX_VERTEX_NUM    100 //假设的最大顶点数
```

```
Typedef enum {DG, DN, AG, AN } GraphKind; //有向/无向图, 有向/无向网
```

```
Typedef struct ArcCell{      //弧(边)结点的定义  
    VRType adj;           //顶点间关系, 无权图取1或0; 有权图取权值类型  
    InfoType *info;        //该弧相关信息的指针  
}ArcCell, AdjMatrix [MAX_VERTEX_NUM] [MAX_VERTEX_NUM];
```

```
Typedef struct{            //图的定义  
    VertexType vexs [MAX_VERTEX_NUM]; //顶点表, 用一维向量即可(n)  
    AdjMatrix arcs;          //邻接矩阵n*n  
    Int Vnum, enum;         //顶点总数n, 弧(边)总数e  
    GraphKind kind;         //图的种类标志  
}Mgraph;
```

对于n个顶点的图或网, 空间效率=O(n²)

16

邻接表(链式)表示

- 对每个顶点建立单链表, 放信息连接
- 顶点v_i: 单链表

邻接点域: 表示v_i邻接点的位置

链域: 指向下一个边或弧的结点

数据域: 与边有关信息

- 头

数据域: 存放顶点v_i信息

链域: 指向第一个结点

2. 邻接表(链式)表示法

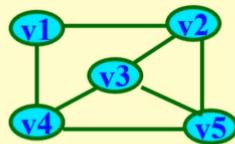
- ① 对每个顶点v_i建立一个单链表, 把与v_i有关联的边的信息
(即度或出度边)链接起来, 表中每个结点都设为3个域:
头结点 **表结点**



- ② 每个单链表还应当附设一个**头结点**(设为2个域), 存v_i信息;

- ③ 每个单链表的**头结点**另外用顺序存储结构存储。

例1：无向图的邻接表如何表示？



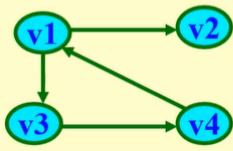
邻接表

0 v ₁	3	1	^
1 v ₂	4	2	0 ^
2 v ₃	4	3	1 ^
3 v ₄	4	2	0 ^
4 v ₅	3	2	1 ^

此无权图未开第3分量

请注意：邻接表不惟一！因各个边结点的链入顺序是任意的。

例2：有向图的邻接表如何表示？



邻接表(出边)

V ₁	2	1	^
V ₂	3	^	
V ₃	0	^	
V ₄	1	^	

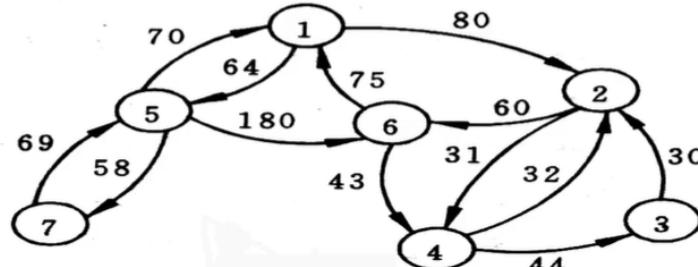
逆邻接表(入边)

V ₁	3	^	
V ₂	0	^	
V ₃	0	^	
V ₄	2	^	

例3：已知某网的邻接（出边）表，请画出该网络。

1	2 80 -	5 64 ▲
2	4 31 -	6 60 ▲
3	2 30 ▲	
4	2 32 -	3 44 ▲
5	1 70 -	6 180 -
6	1 75 -	4 43 ▲
7	5 69 ▲	

当邻接表的存储结构形成后，图便唯一确定！



20

邻接表存储法的特点：—它其实是对邻接矩阵法的一种改进

分析1：对于n个顶点e条边的无向图，邻接表中除了n个头结点外，只有 $2e$ 个表结点，空间效率为 $O(n+2e)$ 。

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

怎样计算无向图顶点的度？ $TD(V_i) = \text{单链表中链接的结点个数}$



分析2：在有向图中，邻接表中除了n个头结点外，只有e个表结点，空间效率为 $O(n+e)$ 。若是稀疏图，则比邻接矩阵表示法合适。

怎样计算有向图顶点的出度？ $OD(V_i) = \text{单链出边表中链接的结点数}$

怎样计算有向图顶点的入度？ $ID(V_i) = \text{邻接点为} V_i \text{ 的弧个数}$

怎样计算有向图顶点 V_i 的度： $TD(V_i) = OD(V_i) + ID(V_i)$ 需遍历全表

邻接表的优点：空间效率高；容易寻找顶点的邻接点；

邻接表的缺点：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。

邻接表 邻接矩阵 异同

- 邻接表：稀疏图
- 邻接矩阵：稠密图

图的邻接表在机内如何表示？（参见教材P163）

```
#define MAX_VERTEX_NUM 20 //假设的最大顶点数
```

```
Typedef struct ArcNode { //弧结构
    int adjvex;           //该弧所指向的顶点位置
    struct ArcNode *nextarc; //指向第一条弧的指针
    InfoArc *info;         //该弧相关信息的指针
} ArcNode;
```

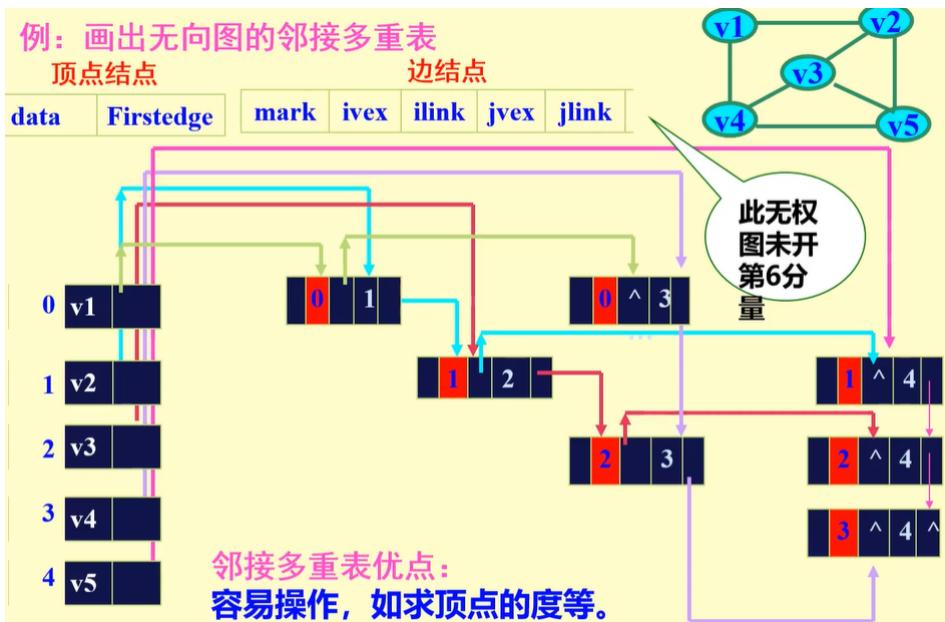
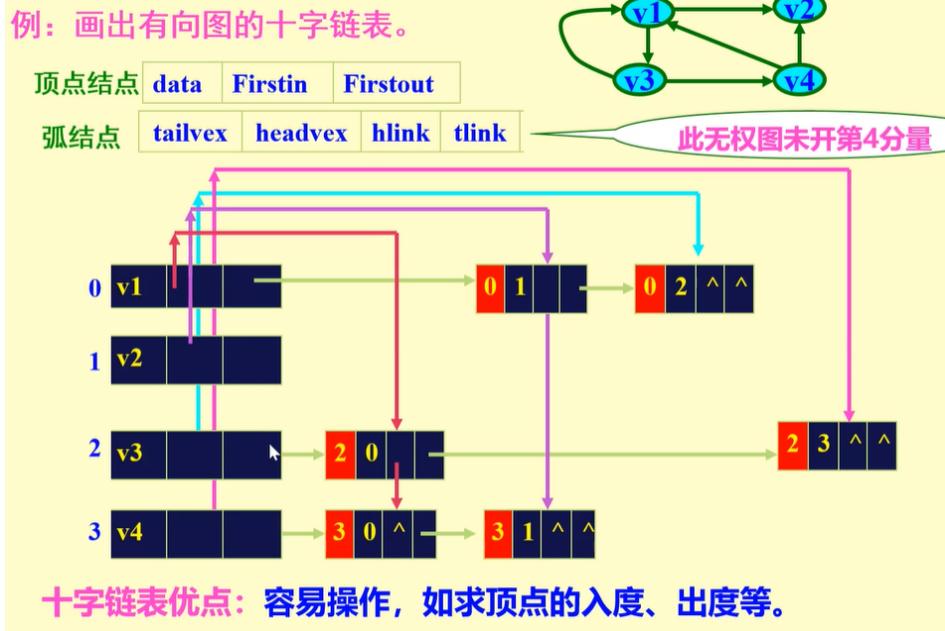
```
Typedef struct VNode{ //顶点结构
    VertexType data;     //顶点信息
    ArcNode *firstarc;   //指向依附该顶点的第一条弧的指针
}VNode, AdjList[ MAX_VERTEX_NUM];
```

```
Typedef struct {          //图结构
    AdjList vertices;      //应包含邻接表
    int vexnum, arenum;    //应包含顶点总数和弧总数
    int kind;               //还应说明图的种类（用标志）
}ALGraph;
```

空间效率为 $O(n+2e)$ 或 $O(n+e)$

时间效率为 $O(n+e*n)$

十字链表 与 邻接多重表



7.3 图的遍历

- 实质：找每个点的邻接点
- 特点：存在回路，可能回到曾经访问过的顶点
- 加一个辅助数组visit[n]
- 深度优先搜索
- 广度优先搜索

7.3.1 DFS

- 基本思路：仿照树的先序遍历
- 栈
- 1. 访问起始点v
2. 若v的第一个邻接点，没访问过，深度遍历之
3. 若已访问过，找第2个邻接点……
- 递归

DFS 算法效率分析:

(设图中有 n 个顶点, e 条边)

- 如果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, 因此遍历全部顶点所需的时间为 $O(n^2)$ 。
- 如果用邻接表来表示图, 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, 因此遍历图的时间复杂度为 $O(n+e)$ 。

结 论:

稠密图适于在邻接矩阵上进行深度遍历;

稀疏图适于在邻接表上进行深度遍历。



效率分析:

- 稠密图: 邻接矩阵
- 稀疏图: 邻接链表

7.3.2 BFS

- 基本思路: 仿照树的按层遍历
- 队列
- 1. 访问起始点 v
 2. 依次访问 v 的邻接点
 3. 不是递归

BFS 算法效率分析:

(设图中有 n 个顶点, e 条边)

- 如果使用邻接表来表示图, 则 BFS 循环的总时间代价为 $d_0 + d_1 + \dots + d_{n-1} = O(e)$, 其中的 d_i 是顶点 i 的度。
- 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 $O(n^2)$ 。

DFS与BFS之比较:

- 空间复杂度相同, 都是 $O(n)$ (借用堆栈或队列装 n 个顶点) ;
- 时间复杂度只与存储结构 (邻接矩阵或邻接表) 有关, 而与搜索路径无关。

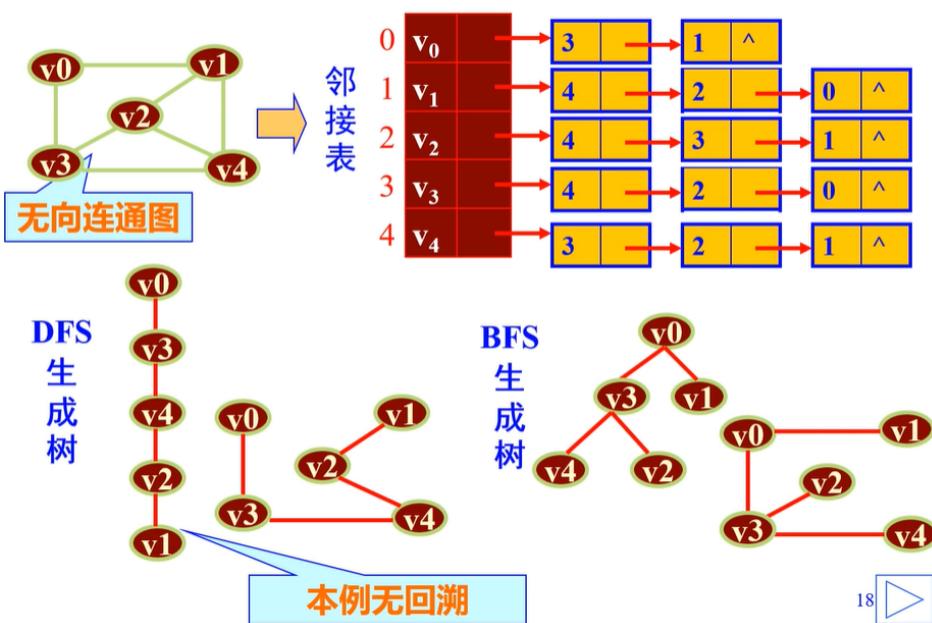
7.4 图的其他运算

- 图的生成树
- 最小生成树
- 最短路径
- 关键路径

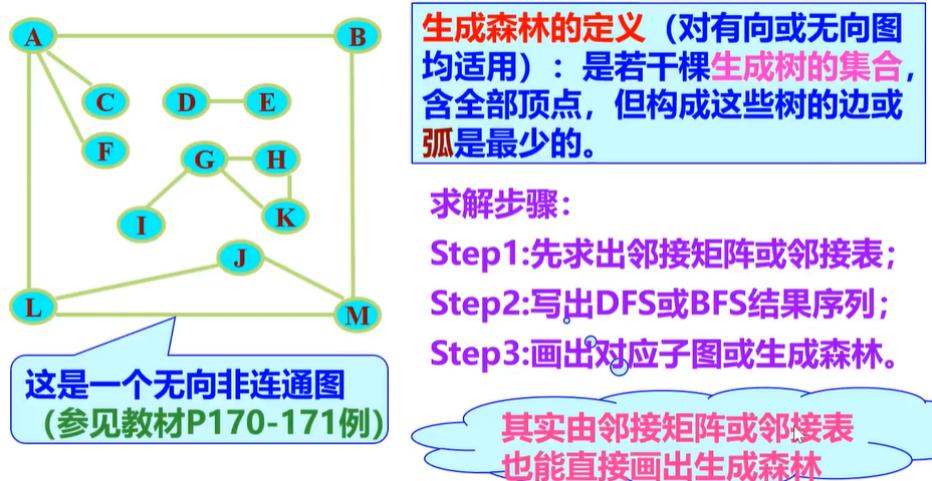
1. 生成树、生成森林

- 对图进行遍历：得到连通子图
 - 深度优先搜索生成树
 - 广度优先搜索生成树
 - 生成树森林（非连通图）

例1：画出下图的生成树



例2：画出下图的生成森林（或极小连通子图）



2. 最小生成树、生成森林

- 在网络中的多个生成树中，寻找一个各边权值之和最小的生成树
 - 类似于Huffman树
 - 针对：有权图
- 使用不同方法遍历，得到不同生成树
- 生成树的边来自于连通网络的n个顶点，和n-1条边
- 准则
 - 使用该网络中的边构造
 - 仅用 n-1 条边连结
 - 不产生回路
- 用途

典型用途：

欲在n个城市间建立通信网，则n个城市应铺n-1条线路；但因为每条线路都会有对应的经济成本，而n个城市可能有 $n(n-1)/2$ 条线路，那么，如何选择n-1条线路使总费用最少？

先建立数学模型：

顶点——表示城市，有n个；
边——表示线路，有n-1条；
边的权值——表示线路的经济代价；
连通网——表示n个城市间的通信网。

显然此连通网是
一棵生成树！

问题抽象：n个顶点的生成树很多，需要从中选一棵代价最小的生成树，即该树各边的代价之和最小。此树便称为最小生成树MST。

Minimum cost Spanning Tree

- 设计思想

* 以权值给边排序，最小者放入生成树内，逐个递增，舍去回路边

- 最常用的两种算法

- Kruskal (库鲁斯卡尔) 算法

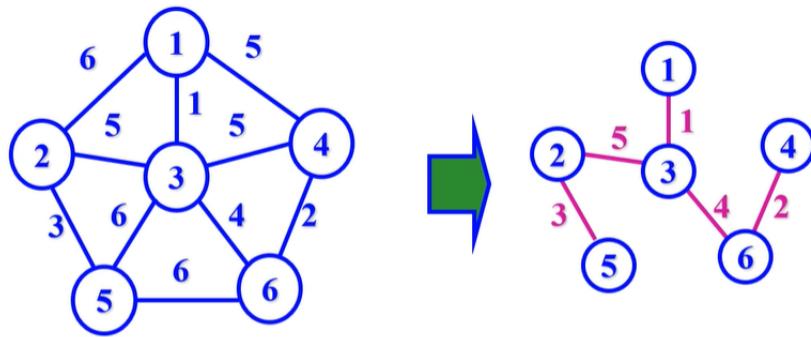
将边进行归并，适用于 **稀疏网**

* 按边的权值排序，依次放入图中，并舍去构成回路的边

* 算法效率分析： $O(e * \log_2(e))$

* 边少的图，稀疏图，邻接表

Kruskal 算法示例：对边操作，归并边



Kruskal 算法效率分析：

Kruskal 算法的时间效率 = $O(e \log_2 e)$

Kruskal 算法是归并边，适用于稀疏图（用邻接表）

- Prim (普里姆) 算法

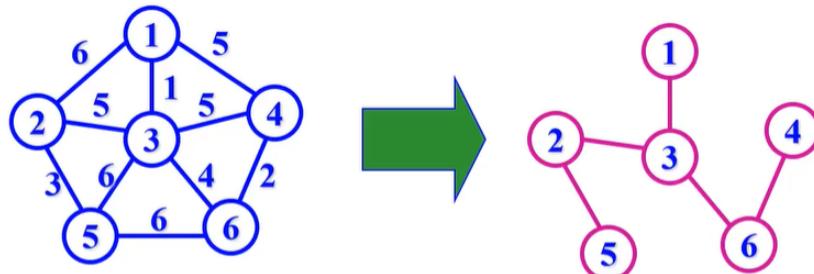
将点进行归并，与边数无关，适用于 **稠密网**

* 按点递归，在已选顶点集中选A，未选顶点集中选B，使得 $E(AB)$ 最短，且一定没有回路

* 算法效率： $O(n^2)$

* 边多的图，稠密网

普利姆 (Prim) 算法示例：归并顶点



Prim 算法效率分析：

Prim 算法的时间效率 = $O(n^2)$

Prim 算法是归并顶点，适用于稠密网。

3. 最短路径

- 源点：起点
- 终点：其他点
- 目标：各边权值之和最小

1. 单源最短路径：Dijkstra 算法
2. 所有顶点间的最短路径：Floyd 算法

Dijkstra算法

- 限定正值
- 按照路径长度递增的次序，依次产生最短路径

Floyd算法

图的应用

活动图

AOV网

活动网络 Activity on Vertices

v_i 必须先于 v_j 进行，有向网络

AOE网

边表示活动：时间

Activity on Edges

AOE网络的用途：

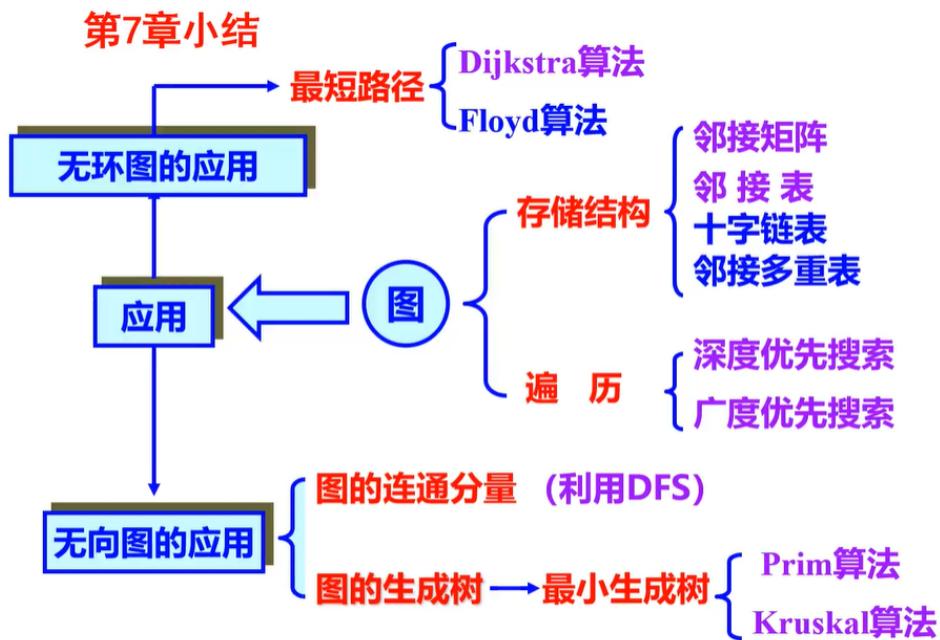
常用于大型工程的计划管理。利用AOE网络可以解决以下两个问题：

- (1) 完成整个工程至少需要多少时间（假设网络中没有环）？
- (2) 为缩短工期，应当加快哪些活动？或者说，哪些活动是影响工程进度的关键？

学习AOE网需要掌握哪些预备知识？

- ① 图的基本概念，邻接矩阵、邻接表等；
- ② 拓扑排序概念
- ③ 关键路径概念





7.7 拓扑排序的方法

基本思想：

1. 输入AOV网络，令n为顶点个数
2. 在AOV中选一个“没有直接前驱”的顶点，输出
3. 在图中删去之（同时删去边）
4. 重复2、3，直到（a或b）
 - a. 全部顶点均已输出
 - b. 还有未输出的顶点，说明必有有向环

算法

1. 建立入度为0的顶点栈
2. while(栈不空):
 - a. 退出一个顶点，输出之
 - b. 在AOV中删除它，及其边（发出边，输入边可以不删）
 - c. 更改其余入度
 - d. 如果有入度为0的点，push进栈
3. 如果输出顶点数小于AOV顶点，报告存在有向环

AOE网

- 由于活动信息记录在Edge上，所以“完成整个工程所需的时间取决于从“源点”到“汇点”的最长路径长度”，该最长路径，即是关键路径

错题集

课后习题

```
{  
    "12": "对于一个具有n个结点和e条边的无向图，用邻接表表示，则顶点表大小为()，所有边链  
    表中结点的总个数为()，  
    "15": "设G采用邻接表存储，则拓扑排序算法的时间复杂度为"，  
}  
}
```

- 普里姆算法：

- 滚动搜索结点，将所有结点v从“图集合”“搬运”到“生成树集合”，故时间复杂度： $O(n^2)$ 。
- 注意，选择时一定是A从“图集合”，B从“生成树集合”

- 克鲁斯卡尔算法：

- 将边排序，根据大小顺序依次生成“生成树集合”
- 时间复杂度： $O(e * \log(e))$