

# 数据结构

北京邮电大学 网络空间安全学院

武 斌



# 上次课内容

上次课（图(上)）内容：

- 领会图的类型定义。
- 熟悉图的各种存储结构及其构造算法，了解各种存储结构的特点及其选用原则。
- 熟练掌握图的两遍遍历算法。





# 本次课程学习目标

学习完本次课程，您应该能够：

- 掌握无向网的最小生成树方法。
- 理解有向无环图及其应用。
- 领会最短路径、拓扑排序、关键路径。
- 理解各种图的应用问题的算法。





# 图的连通性问题

## ● 7.1 图的定义和术语

---

## ● 7.2 图的存储结构

---

## ● 7.3 图的遍历

---

## ● 7.4 图的连通性问题（二）

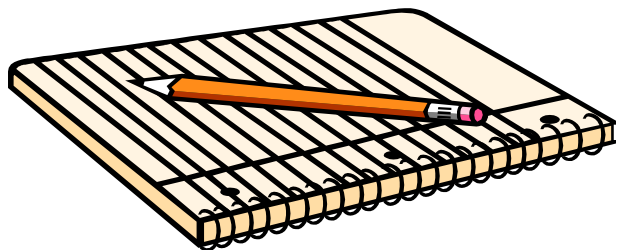
---

## ● 7.5 有向无环图及其应用

---

## ● 7.6 最短路径

---





# 网络工程建设问题

●建设某应用业务网络，连通若干用户单位，使得整体工程成本低、可实施度高。

→ 计算机网络建设：校园网

→ 基础设施网络：燃气、水电、暖气等

 **CERNET主干网**  
China Education and Research Network

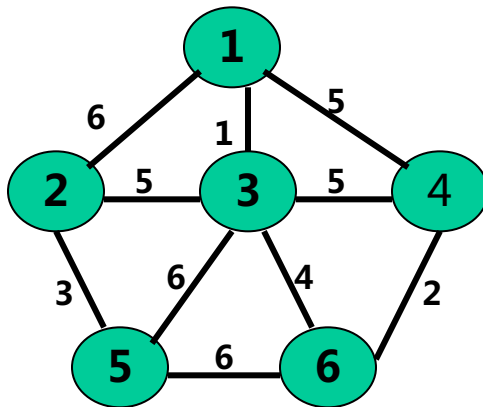




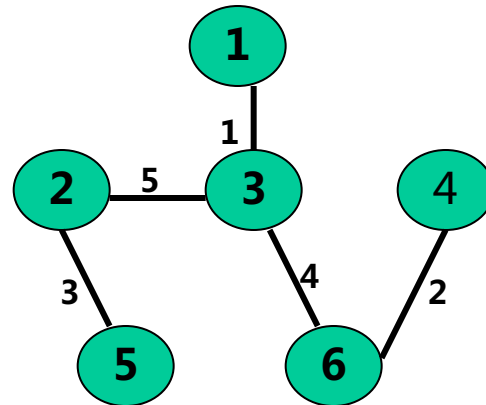
# 最小生成树

●**最小生成树(Minimum Cost Spanning Tree)**: 生成树中边的权值（代价）之和最小的树。

●**实例:**



左图的最小代价生成树



思考：类似的最优树？ 霍夫曼树

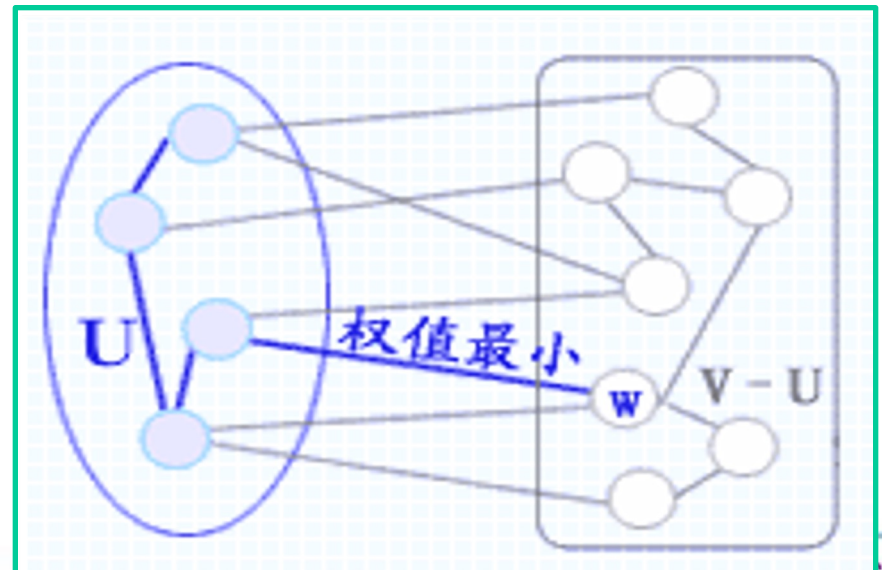


# 最小生成树

## ● MST 性质:

→ 假设  $G = \{V, \{E\}\}$  是一个连通图,  $U$  是结点集合  $V$  的一个非空子集。若  $(u, v)$  是一条代价最小的边, 且  $u$  属于  $U$ ,  $v$  属于  $V-U$ , 则必存在一棵包括边  $(u, v)$  在内的最小代价生成树。

● 普里姆(Prim)算法和克鲁斯卡尔(Kruskal)算法是两个利用MST性质构造最小生成树的算法。

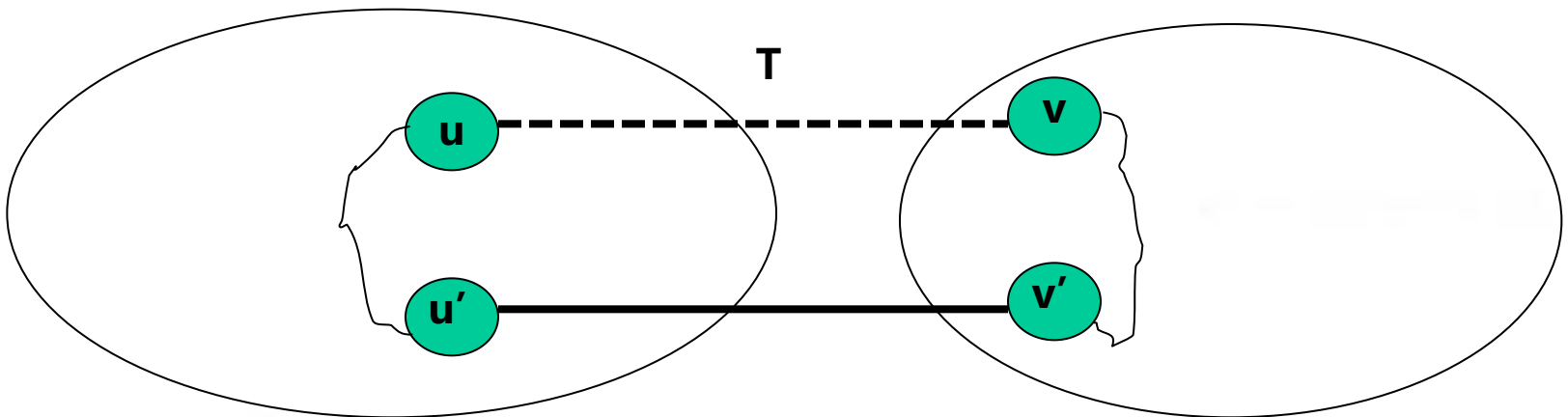




# 最小生成树

## ● MST 性质证明（反证法）：

1. 假定存在一棵**不包括边  $(u, v)$**  在内的最小代价生成树，设其为  **$T$** 。
2. 将边  $(u, v)$  添加到树  $T$ ，则形成一条**包含  $(u, v)$  的回路**。
3. 因此，**必定存在另一条边  $(u', v')$** ，且  $u'$  属于  $U$ ,  $v'$  属于  $V - U$ 。删去边  $(u', v')$ ，得到另一棵生成树  **$T'$** ；因为边  $(u, v)$  的代价小于边  $(u', v')$  的代价，所以**新的生成树  $T'$  将是代价最小的树**。
4. 和原假设矛盾。



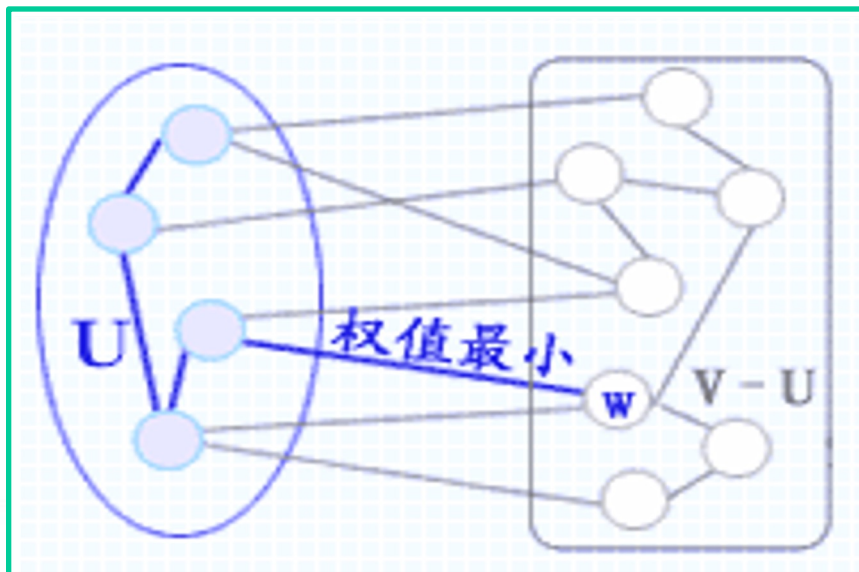




# 普里姆算法

## ● 基本思想:

- 将图中顶点集合 $V$ 分成生成树顶点集 $U$ ，和尚未落在生成树上的顶点集为 $V-U$
  - 首先选取图中任意一个顶点  $v$  作为生成树的根
  - 往生成树中添加顶点  $w$ 。  $w$  和  $v$  之间必须有边，且权值在和  $v$  相邻接的边中最小。
- 顶点  $w$  应满足下列条件：它和生成树上的顶点之间的边上的权值是在连接这两类顶点的所有边中权值属最小。





# 普里姆算法

## ● 普里姆算法（Prim算法）

- 假设 $N=\{V, \{E\}\}$ 是连通网， $TE$ 是 $N$ 上最小生成树中边的集合。
- 从 $U=\{u_0\} (u_0 \in V)$ ， $TE=\{\}$ 开始，重复执行下述操作：
  - 在所有 $u \in U$ ， $v \in V-U$ 的边 $(u, v) \in E$ 中找一条代价最小的边 $(u_0, v_0)$ 并入集合 $TE$ ，同时 $v_0$ 并入 $U$ ，
  - 重复上面操作，直至 $U=V$ 为止。
- 此时 $TE$ 中必有 $n-1$ 条边，则 $T=(V, \{TE\})$ 为 $N$ 的最小生成树。

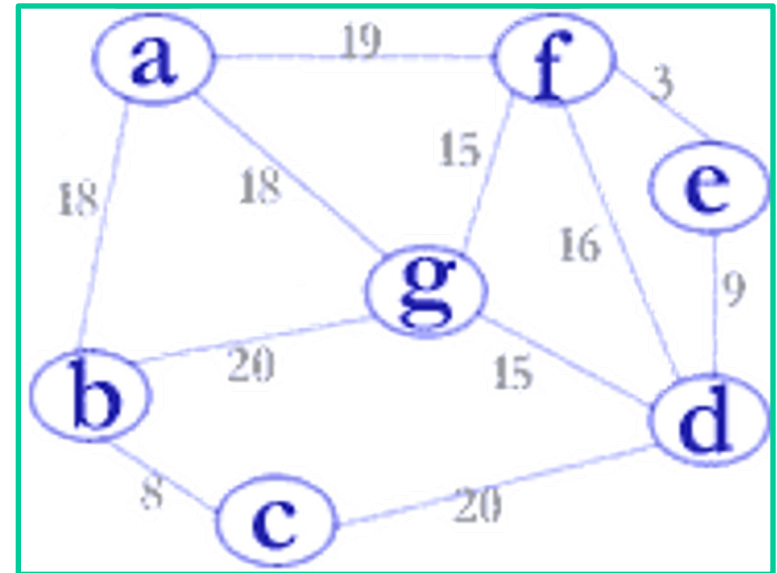


# 普里姆算法

●如右所示连通网G5

→  $G5.vexs = [a,b,c,d,e,f,g]$

$$G5.arcs = \begin{bmatrix} 0 & 18 & \infty & \infty & \infty & 19 & 18 \\ 18 & 0 & 8 & \infty & \infty & \infty & 20 \\ \infty & 8 & 0 & 20 & \infty & \infty & \infty \\ \infty & \infty & 20 & 0 & 9 & 16 & 15 \\ \infty & \infty & \infty & 9 & 0 & 3 & \infty \\ 19 & \infty & \infty & 16 & 3 & 0 & 15 \\ 18 & 20 & \infty & 15 & \infty & 15 & 0 \end{bmatrix}$$

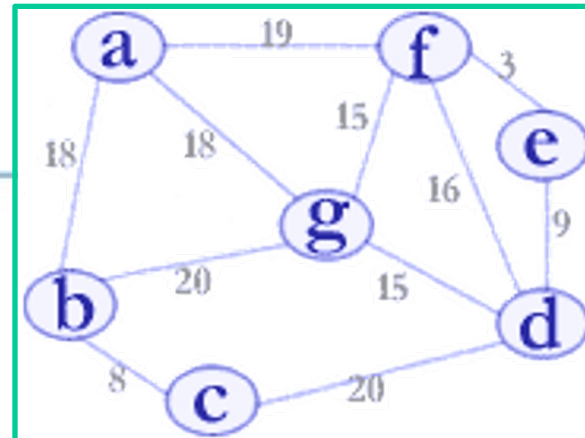


连通网G5



# 普里姆算法

连通网G5



● 例如，右图所示连通网 G5。

1. 假设首先**选顶点 a** 作为生成树的**根**
2. 此时只有顶点 a 在生成树中，其余顶点 b,c,d,e,f,g 均不在生成树上，
3. 联接这两类顶点的边只有(a,b)、(a,f) 和 (a,g)，其中以边 (a,b) 的权值为最小，由此**应该选择边 (a,b) 将顶点b加入到生成树中**
4. 之后链接 {a,b} 和 {c,d,e,f,g} 这两个顶点集的边除了原有的(a,f)，(a,g) 之外，又增加了 (b,c) 和 (b,g)，在这四条边中，权值最小的边为 (b,c)(权值=8)，自然应选边 (b,c)，即将顶点c加入到生成树中。
5. 之后应在链接顶点集 {a,b,c} 和 {d,e,f,g} 的边集 {(a,f),(a,g),(b,g),(c,d)} 中选择权值最小的边(a,g),.....，
6. 依次类推，**直至所有顶点都落到生成树上为止**。上述构筑生成树的过程如演示所示。

→ 演示 7-4-2



# 普里姆算法

- 从上述生成树的构造过程中可以发现，即**每个顶点都是通过“一条边”加入到生成树上的**
- 因此对集合 $V-U$ 中的每个顶点，当它和集合 $U$ 中的顶点有一条以上的边相连时，只需要**保留一条权值最小的边**即可。
  - ➔ **例如**，当顶点  $a$  和  $b$  加入到生成树之后，顶点  $g$  有两条和生成树中顶点相联接的边，由于边  $(b,g)$  的权值大于边  $(a,g)$  的权值，显然顶点  $g$  将来肯定不会通过边  $(b,g)$  加入到生成树。
  - ➔ 类似地，当顶点  $g$  加入到生成树之后，对顶点  $f$  而言，不再考虑边  $(a,f)$  (因为它的权值比边  $(g,f)$  大)。
- 由此，在普里姆算法中需要附设一个**辅助数组 closedge**，以记录从集合  $U$  到集合  $V-U$  中每个顶点当前的权值最小边。
- **普里姆算法**构造生成树的过程如动画所示。**演示 7-4-3**



# 普里姆算法

## ● 算法7.11

**void** MiniSpanTree\_PRIM( MGraph G, VertexType u)

```
{ // 用普里姆算法从第u个顶点出发构造网G的最小生成树，输出T的各条边。
  // 记录从顶点集U到V-U的代价最小的边的辅助数组定义：
  // typedef struct {
  //     VertexType adjvex; U中的顶点，该顶点与V最近
  //     VRType lowcost;   顶点关系（权值）
  // }closededge[MAX_VERTEX_NUM];
  k = LocateVex ( G, u );
  for ( j=0; j<G.vexnum; ++j )      // 辅助数组初始化，到U{u}的最近距离
      if (j!=k) closededge[j] = { u, G.arcs[k][j].adj };    //{adjvex,lowcost}
  closededge[k].lowcost = 0;        // 初始状态，U={u}
  for (i=1; i<G.vexnum; ++i)
  { // 选择其余 G.vexnum-1 个顶点
      k = minimum(closededge); // 求出T的下一个结点(图中第k顶点)
      //此时closededge[k].lowcost = Min{ closededge[vi].lowcost |
                                     closededge[vi].lowcost>0, vi∈V-U }
  }
```



# 普里姆算法

```
printf(closedge[k].adjvex, G.vexs[k]); // 输出生成树的边
closedge[k].lowcost = 0; // 第 k 顶点并入U集
for (j=0; j<G.vexnum; ++j)
    if (G.arcs[k][j].adj < closedge[j].lowcost) // 新顶点并入U后重新选择最小边
        closedge[j] = { G.vexs[k], G.arcs[k][j].adj };
} // for
} // MiniSpanTree
```

## ● 算法分析

- 假设网中有 $n$ 个顶点，则第一个进行初始化的循环语句的频度为 $n$ ，第二个循环语句的频度为 $n-1$ 。
- 其中有两个内循环：其一是在`closedge[v].lowcost`中求最小值，其频度为 $n-1$ ；其二是重新选择具有最小代价的边，其频度为 $n$ 。
- 由此，普里姆算法的时间复杂度为 $O(n^2)$ ，与网中的边数无关，因此适用于求边稠密的网的最小生成树。





# 克鲁斯卡尔算法

- **基本思想**：为使生成树上总的权值之和达到最小，则应使每一条边上的权值尽可能地小，自然**应从权值最小的边选起，直至选出  $n-1$  条互不构成回路的权值最小边为止。**
- **具体作法如下**：
  - ➔ 首先构造一个只含  $n$  个顶点的森林
  - ➔ 然后依权值从小到大从连通网中选择**不使森林中产生回路的边**加入到森林中去
  - ➔ 直至该**森林变成一棵树**为止，这棵树便是连通网的最小生成树。



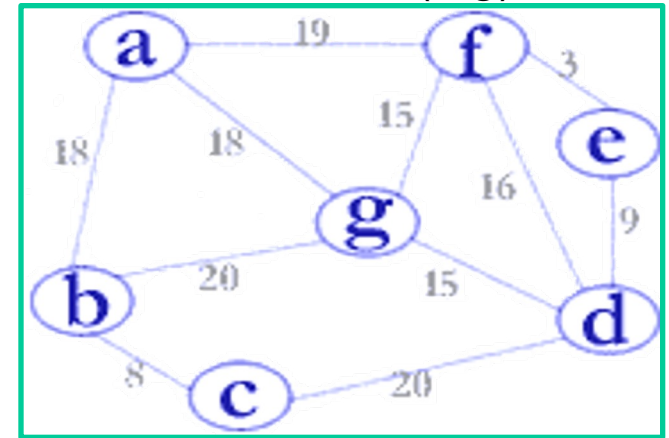


# 克鲁斯卡尔算法

- 生成树上不允许有回路，因此并非每一条居当前权值最小的边都可选，例如，对下图所示连通网G5，选中了(e,f),(b,c),(e,d) 和 (f,g) 的四条边之后，权值最小边为 (g,d)，由于 g 和 d 已经连通，若加上(g,d) 这条边将使生成树上产生回路，显然这条边不可取，同理边 (f,d) 也不可取，之后则依次取 (a,g) 和 (a,b) 两条边加入到生成树。

- 演示 7-4-1

- 思考：在算法中如何判别当前权值最小边的两个顶点之间是否已经连通？



连通网G5

- 分析：从生成树的构造过程可见，初始态为  $n$  个顶点分属  $n$  棵树，互不连通，每加入一条边，就将两棵树合并为一棵树，在同一棵树上的两个顶点之间自然相连通。由此判别当前权值最小边是否可取只要判别它的两个顶点是否在同一棵树上即可。



# 克鲁斯卡尔算法

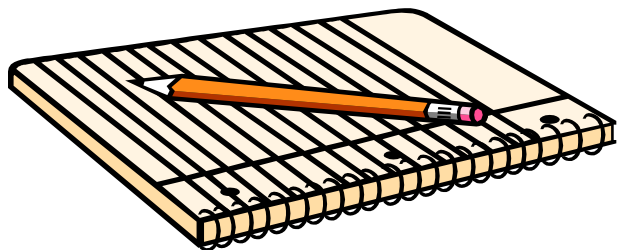
---

- 如果**不考虑建立图的存储结构所需时间**，则克鲁斯卡尔算法的时间复杂度为 **$O(e \log e)$**  ( $e$ 为网中边的数目)，因此它相对于普里姆算法而言，**适合于求边稀疏的网的最小生成树**。



## 本章课程内容（第七章 图）

- 7.1 图的定义和术语
- 7.2 图的存储结构
- 7.3 图的遍历
- 7.4 图的连通性问题
- 7.5 有向无环图及其应用
- 7.6 最短路径





# 最短路径

- 图或网还经常用于描述一个城市或城市间的交通运输网络，以图中顶点表示一个城市或某个交通枢纽，以边或弧表示两地之间的交通状况，边或弧上的权值可表示各种相关信息，例如两地之间的路程长度，交通费用或行程时间等等。
- 对于网来说，两个顶点之间的路径长度是路径中“弧的权值之和”。当两个顶点之间存在多条路径时，必然存在一条“最短路径”。
- 考虑到交通图的有向性，本节将讨论带权有向图(有向网)，并称路径中的第一个顶点为“源点”，路径中的最后一个顶点为“终点”。以下讨论两种最常见的路径问题。

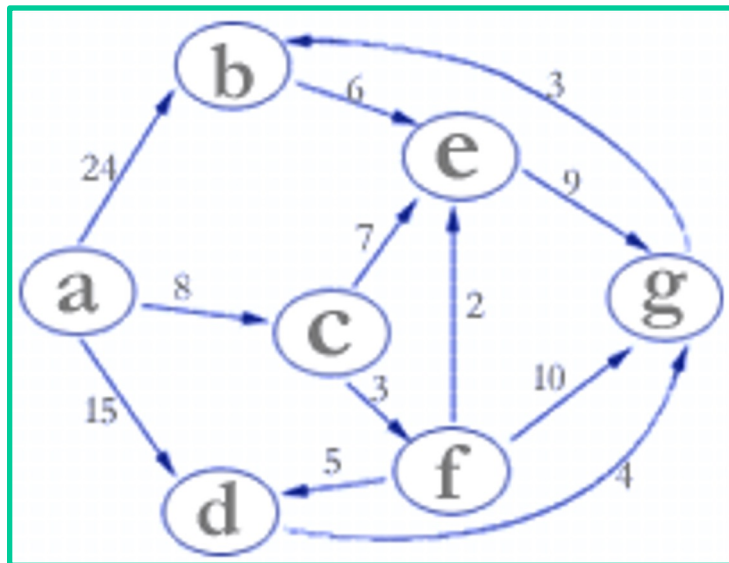


# 最短路径

## ● 1、单源点路径问题

单源点路径问题是指，已知一个有向网和网中某个源点，求得从该源点到图中其它各个顶点之间的最短路径。

→ 例如，下图 G6 中，从源点 a 到终点 b 存在多条路径，如路径 {a,b} 的长度为24，路径 {a,c,e,g,b} 的长度为27等，其中以路径 {a,d,g,b} 的长度(=22)为最短。类似地，从源点 a 到其它各顶点也都存在一条路径长度最短的路径，如右下所列。



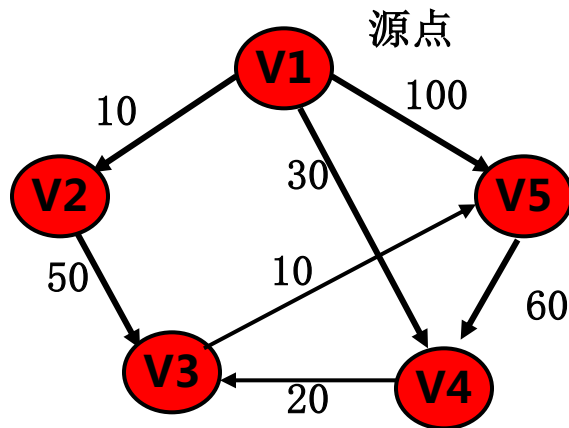
➤ 从源点 a 到图中其它各终点的最短路径按路径长度从短到长依次为：

路径{a, c}的长度为8，  
路径{a, c, f}的长度为11，  
路径{a, c, f, e}的长度为13，  
路径{a, d}的长度为15，  
路径{a, d, g}的长度为19，  
路径{a, d, g, b}的长度为22。



# 最短路径

## 实例:加权有向图



加权邻接矩阵:

	0	1	2	3	4
0	0	10	∞	30	100
1	∞	0	50	∞	∞
2	∞	∞	0	∞	10
3	∞	∞	20	0	0
4	∞	∞	∞	60	0

源点	终点	最短路径	路径长度
V <sub>1</sub>	V <sub>2</sub>	(V <sub>1</sub> , V <sub>2</sub> )	10
	V <sub>3</sub>	(V <sub>1</sub> , V <sub>4</sub> , V <sub>3</sub> )	50
	V <sub>4</sub>	(V <sub>1</sub> , V <sub>4</sub> )	30
	V <sub>5</sub>	(V <sub>1</sub> , V <sub>4</sub> , V <sub>3</sub> , V <sub>5</sub> )	60

注意: 这里0、1、2、3、4用于标识结点 V<sub>1</sub>、V<sub>2</sub>、V<sub>3</sub>、V<sub>4</sub>、V<sub>5</sub>。



# 最短路径

- 如何求得这些最短路径？**迪杰斯特拉提出了一个“按各条最短路径长度递增的次序”产生最短路径的算法。**
  - ➔ 从有向网G6的例子可见，如果从源点到某个终点存在路径，必存在一条路径长度取最小值的路径，这些最短路径彼此之间的长度不一定相等，则**迪杰斯特拉算法（Dijkstra）**正是按从源点a到其它各终点的最短路径长度递增的次序先后产生这些最短路径。





# 迪杰斯特拉算法

●可以如下分析这些最短路径的特点。

- ➔ 首先，在这些最短路径中，**长度最短的这条路径上必定只有一条弧**，且它的权值是从源点出发的所有弧上权的最小值。
- ➔ 其次，第二条长度次短的最短路径只可能有两种情况：
  - 或者**只含一条从源点出发的弧**且弧上的权值大于已求得最短路径的那条弧的权值，但小于其它从源点出发的弧上的权值；
  - 或者是一条**只经过已求得最短路径的顶点的路径**，换句话说，如果第一条长度最短的最短路径是从源点 $s$ 到 $v_1$ 的话，若从 $v_1$ 到 $v_2$ 之间存在一条弧，且路径  $\{s, v_1, v_2\}$  的长度比  $\langle s, v_2 \rangle$  的权值小，则  $\{s, v_1, v_2\}$  是第二条长度次短的最短路径。
- ➔ 依次类推，按迪杰斯特拉算法先后求得的每一条最短路径必定只有两种情况，或者是由源点直接到达终点，或者是只经过已经求得最短路径的顶点到达终点。

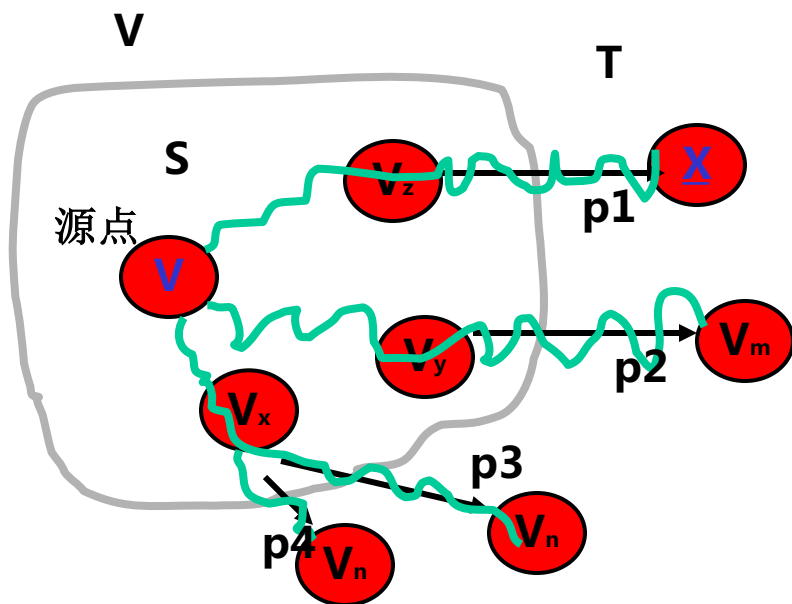




# 迪杰斯特拉算法

## • Dijkstra 算法证明（反证法）

设  $V$  是顶点的集合， $S$  是已经得到最短路径（由源点至本结点）的结点的集合。 $T = V - S$

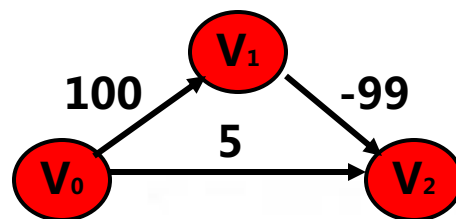


证明：设源点为 $V$ ，终点为 $X$ ，设 $V$ 经 $S$ 中的 $n$ 个结点到 $V_z$ 后直接经有向边到达结点 $X$ ，且路径是所有到 $X$ 的路径中最短的。

如图所示，路径 $p1 < p2$ ;  $p1 < p3$ ;  $p1 < p4$ 。则  $p1$  就是源点 $V$ 至 $X$ 的最短路径。

假定存在一条更短的路径，那么肯定要经过  $T - \{X\}$  中的一个顶点，设为  $V_n$ ；但  $p3 > p1$ ，同假设矛盾，不可能。

•Dijkstra 算法: 边的权值不可以为负值。



Dijkstra 算法和Prim算法本质是相同的，甚至可以用同一个函数求出，区别仅在于权值不同。



# 迪杰斯特拉算法

● **例如**，求得图G8中从源点a到其它终点的最短路径的过程如动画所示。[演示](#) 7-5-0

- ➔ 从这个过程可见，类似于普里姆算法，在算法中应保存当前已得到的从源点到每个终点的最短路径
- ➔ 初值为：如果从源点到该点有弧，则存在一条路径，其路径长度即为该弧上的权值
- ➔ 之后每求得一条到达某个终点  $w$  的“最短路径”之后，就需要检查一下，是否存在经过这个顶点  $w$  的其它路径(即是否存在从顶点  $w$  出发到尚未求得最短路径顶点的弧)，如果存在，其长度是否比当前求得的路径长度短，如果是，则应修改当前路径。



## 迪杰斯特拉算法

- 假设  $n$  为图  $G=(V,E)$  中的顶点数,  $D[n]$  存放从源点到每个终点当前最短路径的长度,  $P[n]$  存放相应路径,  $S$  为已求得最短路径的终点的集合。则迪杰斯特拉算法求最短路径的过程为:

→ (1) 令  $S=\{v_s\}$ ; //  $v_s$  为源点

并设定  $D[i]$  的初始值为:

$$D[i] = \begin{cases} 0 & \text{当 } i = s \text{ 时} \\ w_{si} & \text{当 } i \neq s, \text{ 且 } \langle v_s, v_i \rangle \in E, w_{si} \text{ 为弧上的权值} \\ \infty & \text{当 } i \neq s, \text{ 且 } \langle v_s, v_i \rangle \notin E \end{cases}$$
$$P[i] = \begin{cases} \text{空串} & \text{当 } i = s \text{ 或 } \langle v_s, v_i \rangle \notin E \\ "v_s v_i" & \text{当 } i \neq s \text{ 且 } \langle v_s, v_i \rangle \in E \end{cases}$$

- (2) 选择顶点  $v_j$  使得  $D[j] = \min\{D[k] \mid v_k \in V - S\}$ , 并将顶点  $v_j$  并入到集合  $S$  中;
- (3) 对集合  $V - S$  中所有顶点  $v_k$ , 若存在从  $v_j$  指向该顶点的弧, 且  $D[j] + w_{jk} < D[k]$ , 则修改  $D[k]$  和  $P[k]$  的值。
- (4) 重复(2)和(3)直至求得从源点到所有其它顶点的最短路径为止。



# 迪杰斯特拉算法

● **算法7.16** void ShortestPath\_DIJ( MGraph G, int  $v_0$ , PathMartrix &P, ShortPathTable &D ){// 求有向网G的顶点 $v_0$ 到其余顶点v的最短路径P[v]及其带权长度d[v]。若P[v][w]为TRUE, 则w是从 $v_0$ 到v当前求得最短路径上的顶点。final[v]为TRUE当且仅当 $v \in S$ , 即已经求得从 $v_0$ 到v的最短路径。

```
for ( v=0; v<G.vexnum; ++v) {
```

```
    final[v] = FALSE; D[v] = G.arcs[s][i];
```

```
    for(w=0;w<G.vexnum;++w)    P[v][w]=FALSE; // 设空路径
```

```
    if (D[v] < INFINITY) { P[v][ $v_0$ ]=TRUE; P[v][v]=TRUE; }
```

```
} // for
```

```
D[ $v_0$ ] = 0; final[ $v_0$ ] = TRUE; // 初始化, 顶点 $v_0$ 属于S集
```

```
// 开始主循环, 每次求得 $v_0$ 到某个v顶点的最短路径, 并加v到S集
```

```
for(i=1;i<G.vexnum;++i){           // 其余G.vexnum-1个顶点
```

```
    min = INFINITY;                // 当前所知离 $v_0$  顶点的最近距离
```

```
    for(w=0;w<G.vexnum;++w)
```

```
        if(!final[w])                // w顶点在V-S中
```

```
            if(D[w]<min) {v=w; min=D[w];} // w顶点离 $v_0$ 顶点更近
```

```
    final[v]=TRUE;                // 离 $v_0$ 顶点最近的v加入S集
```



# 迪杰斯特拉算法

```
for(w=0;w<G.vexnum;++w)    // 更新当前最短路径及距离
if(!final[w]&&(min+G.arcs[v][w]<D[w])){ // 修改D[w]和P[w],
                                   w∈V-S
    D[w]=min+G.arcs[v][w];
    P[w]=P[v];    P[w][w]=TRUE;    // P[w]=P[v]+[v][w]
} // if
} // for
} // ShortestPath_DIJ
```

- 容易分析出，迪杰斯特拉算法的时间复杂度和普里姆算法相同，也是  $O(n^2)$
- 即使是只要求求一条从源点到某个特定终点之间的最短路径，但如果是应用迪杰斯特拉算法的话，那么必须先求出所有“路径长度”比它短的最短路径，因此除非它是一条路径长度最短的最短路径，否则时间复杂度仍是  $O(n^2)$  的。



# 弗洛伊德算法

## ● 2、各对顶点间的最短路径问题

如果希望求得图中任意两个顶点之间的最短路径，显然只要依次将每个顶点设为源点，调用迪杰斯特拉算法 $n$ 次便可求出，其时间复杂度为  $O(n^3)$ 。

**弗洛伊德**提出了另外一个算法，虽然其时间复杂度也是  $O(n^3)$ ，但算法形式更简单。

弗洛伊德算法的基本思想是求得一个  $n$  阶方阵的序列

$$D^{(-1)}, D^{(0)}, D^{(1)}, \dots, D^{(k)}, \dots, D^{(n-1)}$$

其中：  $D^{(-1)}[i][j]$  表示从顶点  $v_i$  不经过其它顶点直接到达顶点  $v_j$  的路径长度，即

→  $D^{(-1)}[i][j] = G.\text{arcs}[i][j],$

→  $D^{(k)}[i][j]$  则表示中间只可能经过  $v_0, v_1, \dots, v_k$  等顶点，且不可能经过  $v_{k+1}, v_{k+2}, \dots, v_{n-1}$  等顶点的最短路径长度。

由此，  $D^{(n-1)}[i][j]$  自然是从顶点  $v_i$  到顶点  $v_j$  的最短路径长度。



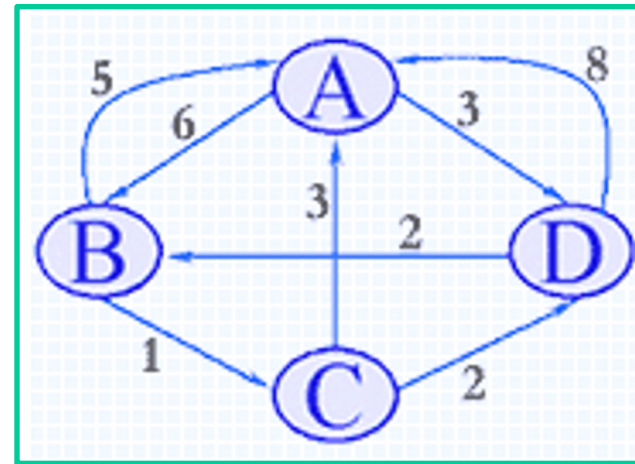
# 最短路径

- 为了记下路径，和上列路径长度序列相对应有路径的方阵序列

$$P^{(-1)}, P^{(0)}, P^{(1)}, \dots, P^{(k)}, \dots, P^{(n-1)}$$

- 弗洛伊德算法的基本操作为：

```
if ( $D[i][k] + D[k][j] < D[i][j]$ )  
{  
     $D[i][j] = D[i][k] + D[k][j]$ ;  
     $P[i][j] = P[i][k] + P[k][j]$   
}
```



有向网G7

- ➔ 其中  $k$  表示在路径中新增添考虑的顶点号， $i$  为路径的起始顶点号， $j$  为路径的终止顶点号。
- 对有向网G7按弗洛伊德算法求得各对顶点之间的最短路径的过程如动画所示。[演示](#) 7-5-1





## 本章小结

- 图是一种比线性表和树更复杂的数据结构。在图形结构中，**结点之间的关系可以是任意的**，图中任意两个元素之间都可能相邻。
- 和树类似，**图的遍历**是图的一种主要操作，可以通过遍历判别图中任意两个顶点之间是否存在路径、判别给定的图是否是连通图并可求得非连通图的各个连通分量。
- 对于带权图（网），其**最小生成树**或**最短路径**都取决于弧或边上的权值，则需要有特定的算法求解。
- 求最小生成树的**普里姆算法**、**克鲁斯卡尔算法**。
- **拓扑排序**、**关键路径**。
- 求最短路径的**迪杰斯特拉算法**、**弗洛伊德算法**。





## 本章（下）知识点与重点

---

### ● 知识点

无向网的最小生成树、最短路径、拓扑排序、关键路径，各种图的算法及其应用场合。

### ● 重点和难点

各种图的算法原理及其实现。