数据结构

北京邮电大学信息安全中心
武斌 杨榆

上次课内容

上次课（图(上)）内容：

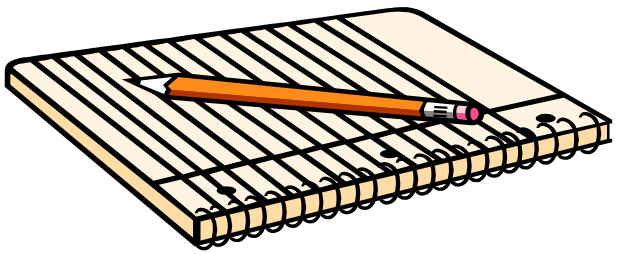
- 领会图的类型定义。
- 熟悉图的各种存储结构及其构造算法，了解各种存储结构的特点及其选用原则。
- 熟练掌握图的两种遍历算法。

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

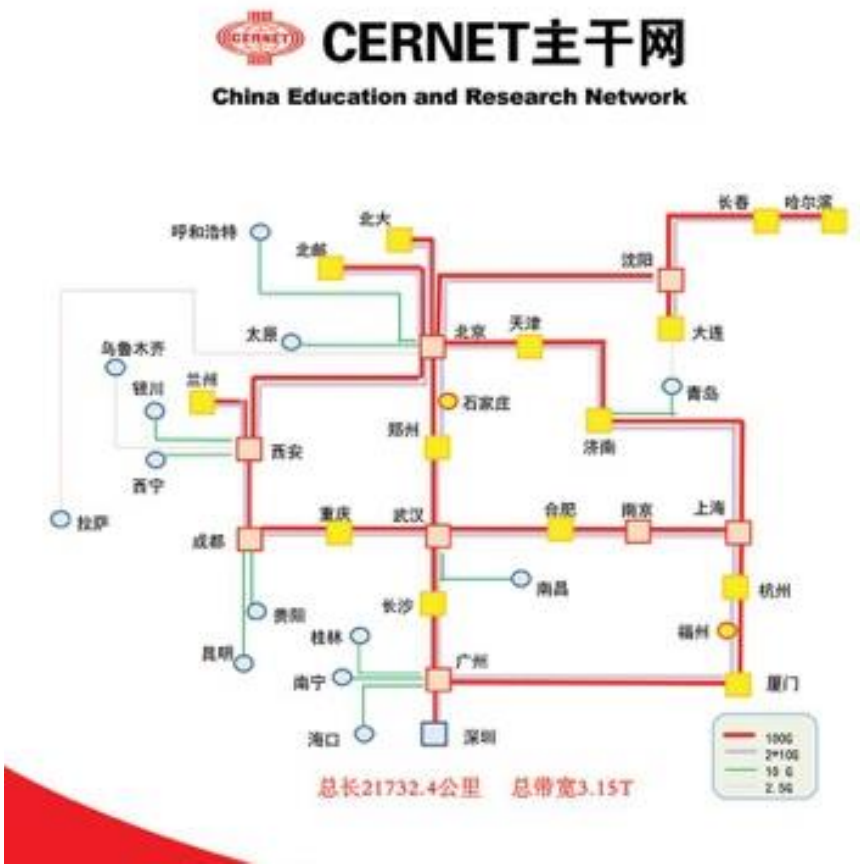


- 7.1 图的定义和术语
- 7.2 图的存储结构
- 7.3 图的遍历
- 7.4 图的连通性问题（二）
- 7.5 有向无环图及其应用
- 7.6 最短路径

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。

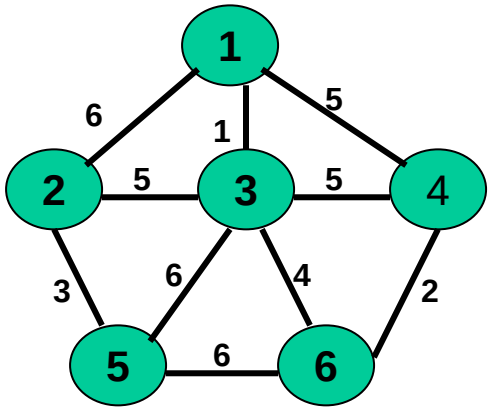
- ➔ 计算机网络建设：校园网
- ➔ 基础设施网络：燃气、水电、暖气等



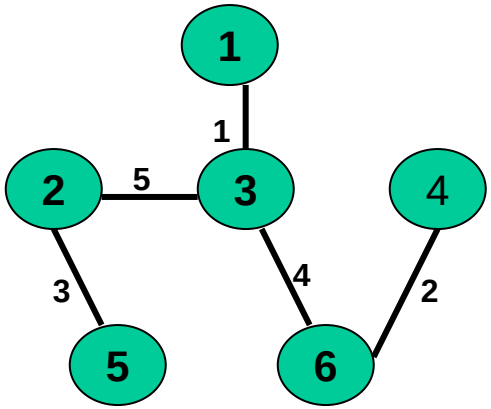
最小生成树

●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

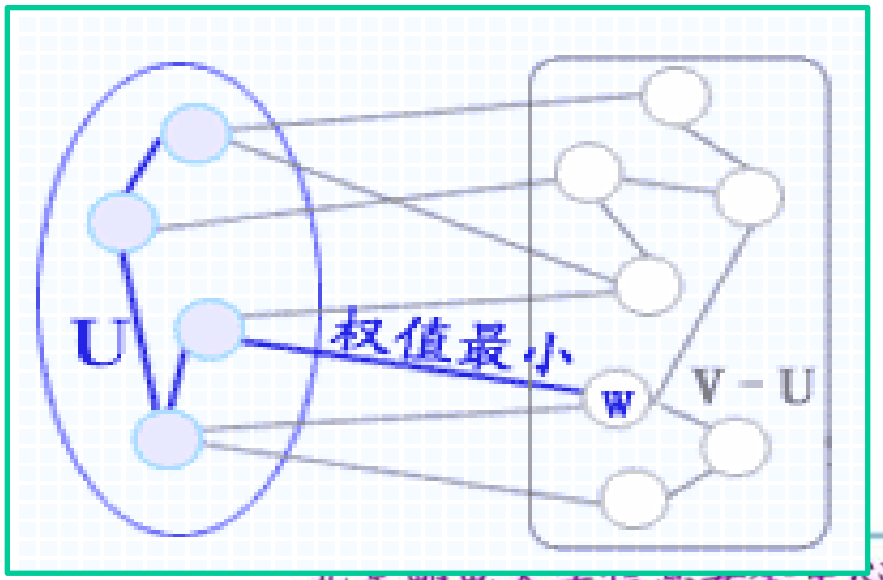


思考：类似的最优树？ 霍夫曼树

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**。

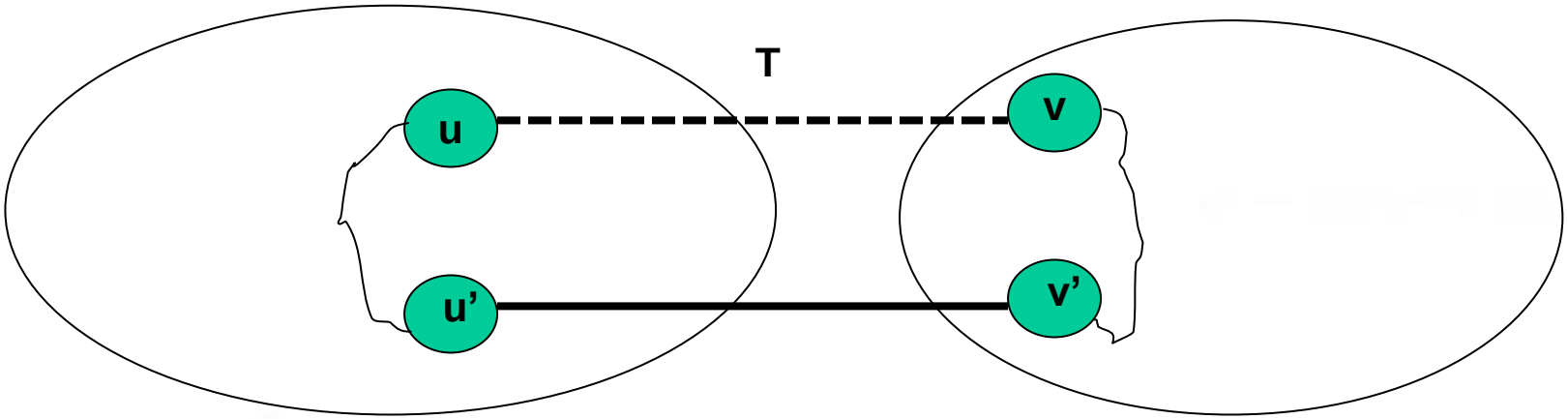
- **普里姆(Prim)算法**和**克鲁斯卡尔(Kruskal)算法**是两个利用MST性质构造最小生成树的算法。



最小生成树

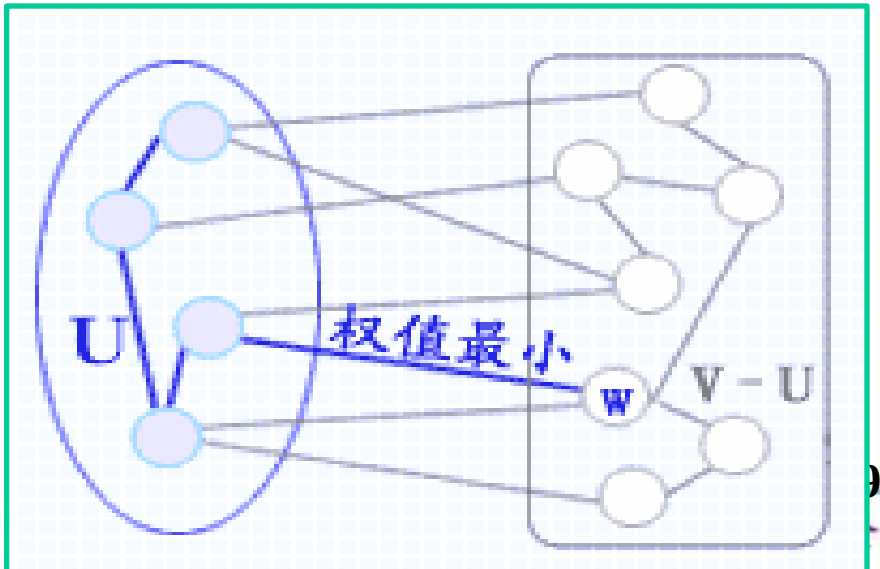
- **MST 性质证明（反证法）：**

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（Prim）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**。



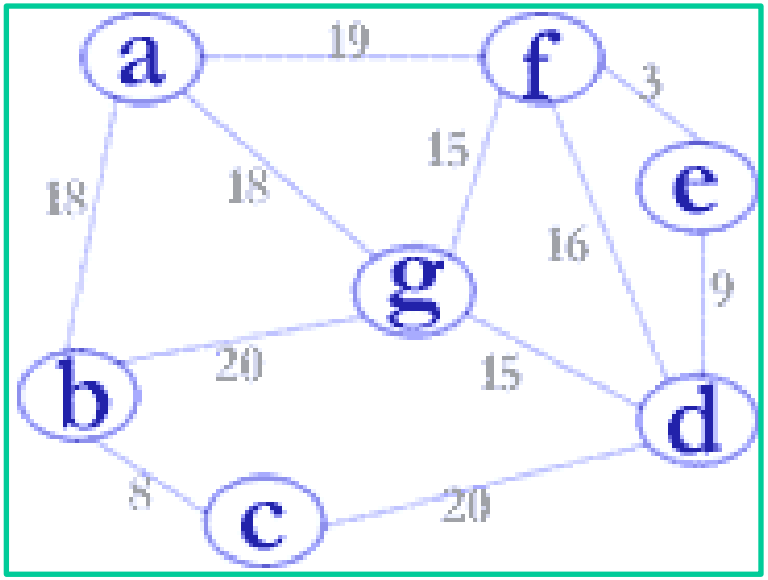
普里姆（Prim）算法

- **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∈V)，TE={ }开始，重复执行下述操作：
    - 在所有u∈U，v∈V-U的边(u,v)∈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

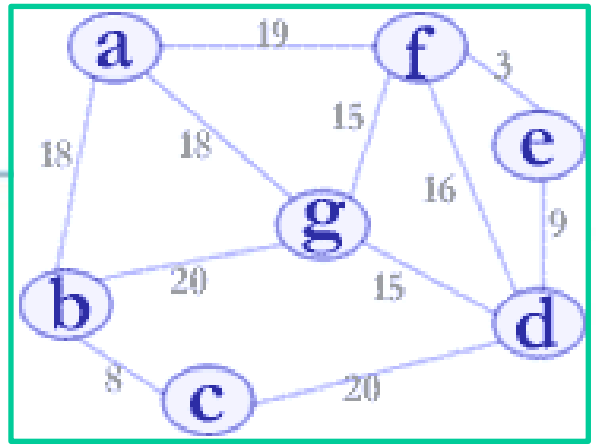
- 如右所示连通网G5
  - G5.vexs = [a,b,c,d,e,f,g]

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



连通网G5

普里姆算法



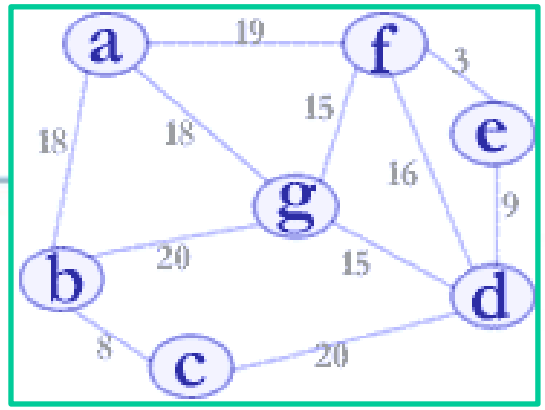
连通网G5

- 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flash\chap07\7-4-2.swf

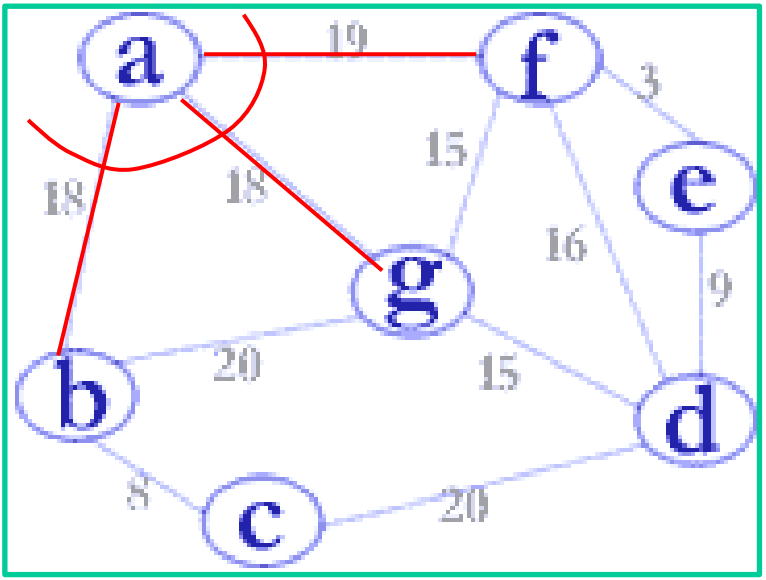
普里姆算法



- 从上述生成树的构造过程中可以发现，
- **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 加入到生成树上之后，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flash\chap07\7-4-3.swf

普里姆算法



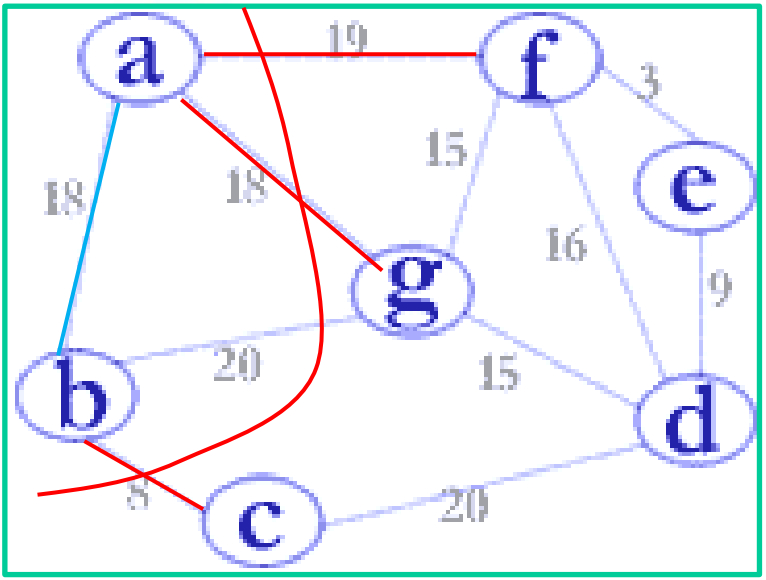
辅助数组closedge记录从U到V-U具有最小代价的边。对于每一个在V-U中的顶点v_i，closedge[i].lowcost为U中顶点到v_i的最小代价， closedge[i].adjvex为与v_i有最小代价的U中的顶点。

1.初始状态，U={a}，V-U={b,c,d,e,f,g},第一行记录从集合U中顶点到V-U中对应顶点的最小代价边。

可知，到b的最小代价边是(a,b)，权值为18，关联的顶点为a。类似地，得出到顶点f，g的最小代价边和权值信息。特别地，若不存在从U到V-U中顶点的最小代价边，则应设置为机器允许的最大值，方便起见，这里留空。所有从U到V-U的边，代价最小的边是(a,b)。

closedge	1(b)	2(c)	3(d)	4(e)	5(f)	6(g)
adjvex(cos)	a(18)				a(19)	a(18)

普里姆算法

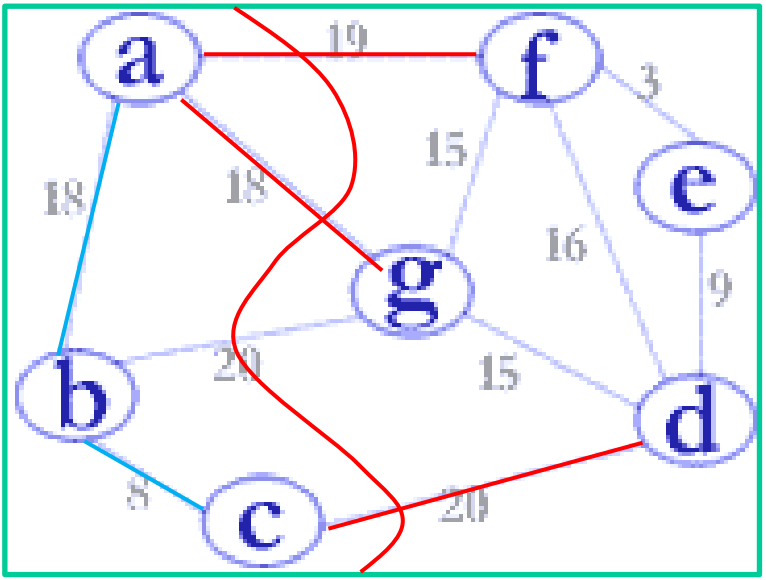


2. 将最小代价边(a,b)依附的顶点b加入集合U，其对应closedge[1]信息置零。V-U变为{c,d,e,f,g}。

- 顶点b的加入，可能使从U到V-U的最小代价边变化。因此，需要比较原最小代价边，以及从顶点b到V-U中各顶点的边，并用较小者作为新的最小代价边。
- 本轮中，(b,c)权值为8，代替原来不存在（权值较大）的a和c；(b,g)权值大于(a,g)，因此最小代价边不更新，仍然是(a,g)。
- 更新后，挑选最小代价边，将其依附的顶点加入集合U。挑最小代价边时，closedge[i].lowcost为0的边不需要考虑。本轮所有从U到V-U的边，代价最小的边是(b,c)。

closedge	1(b)	2(c)	3(d)	4(e)	5(f)	6(g)
adjvex(cos)	a(18)				a(19)	a(18)
adjvex(cos)	0	b(8)			a(19)	a(18)

普里姆算法

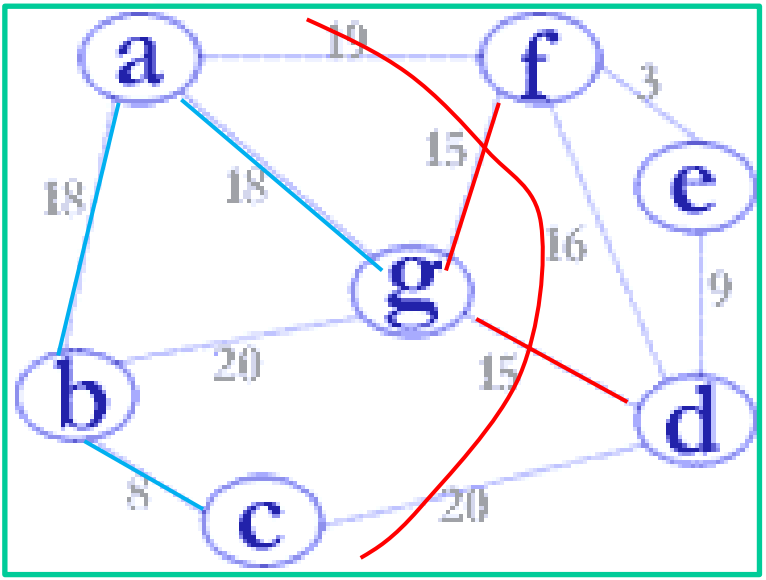


3. 将最小代价边(b,c)依附的顶点c加入集合U，其对应closedge[2]信息置零。V-U变为{d,e,f,g}。

- 顶点c的加入，可能使从U到V-U的最小代价边变化。因此，需要比较原最小代价边，以及从顶点c到V-U中各顶点的边，并用较小者作为新的最小代价边。
- 本轮中，(c,d)权值为20，代替原来不存在（权值较大）的边。
- 更新后，挑选最小代价边，将其依附的顶点加入集合U。挑最小代价边时，closedge[i].lowcost为0的边不需要考虑。本轮所有从U到V-U的边，代价最小的边是(a,g)。

closedge	1(b)	2(c)	3(d)	4(e)	5(f)	6(g)
adjvex(cos)	a(18)				a(19)	a(18)
adjvex(cos)	0	b(8)			a(19)	a(18)
adjvex(cos)	0	0	c(20)		a(19)	a(18)

普里姆算法

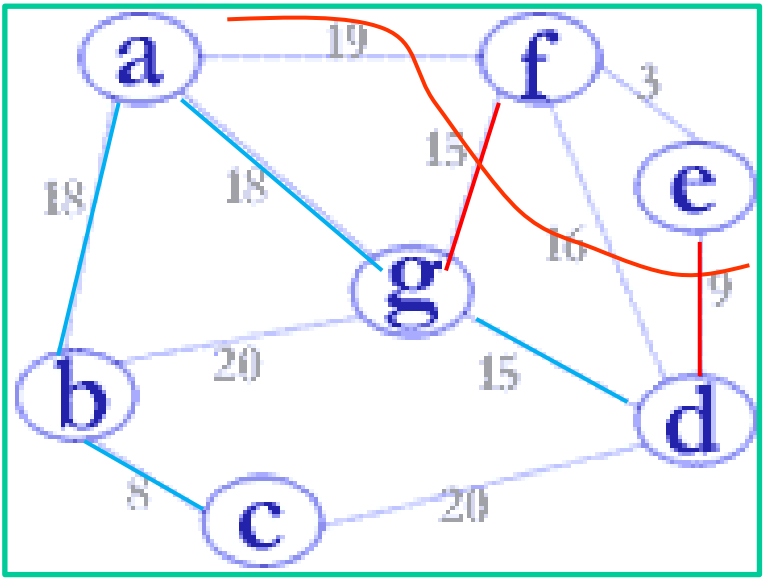


3. 将最小代价边(a,g)依附的顶点g加入集合U，其对应closedge[6]信息置零。V-U变为{d,e,f}。

- 顶点g的加入，可能使从U到V-U的最小代价边变化。因此，需要比较原最小代价边，以及从顶点g到V-U中各顶点的边，并用较小者作为新的最小代价边。
- 本轮中，(g,d)权值为15，代替(c,d)；(g,f)权值为15，代替(a,f)；。
- 更新后，挑选最小代价边，将其依附的顶点加入集合U。挑最小代价边时，closedge[i].lowcost为0的边不需要考虑。本轮所有从U到V-U的边，代价最小的边是(g,d)。

closedge	1(b)	2(c)	3(d)	4(e)	5(f)	6(g)
adjvex(cos)	a(18)				a(19)	a(18)
adjvex(cos)	0	b(8)			a(19)	a(18)
adjvex(cos)	0	0	c(20)		a(19)	a(18)
adjvex(cos)	0	0	g(15)		g(15)	0

普里姆算法

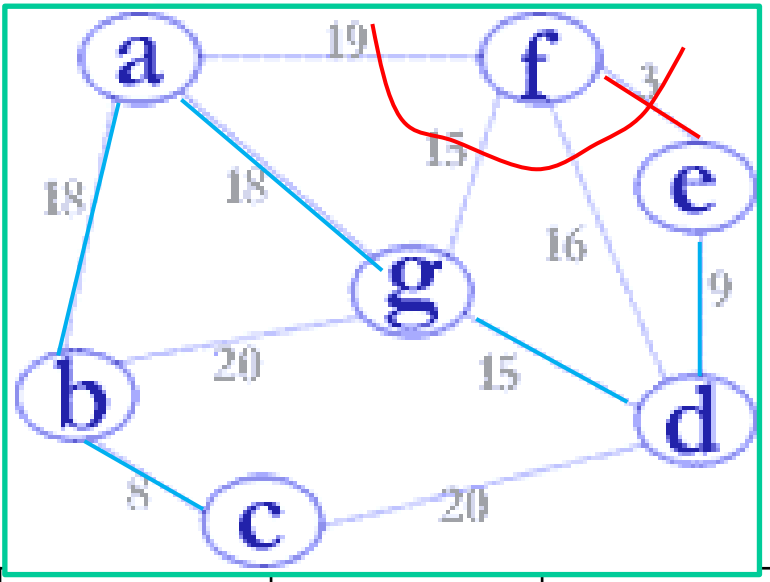


3. 将最小代价边(g,d)依附的顶点d加入集合U，其对应closedge[3]信息置零。V-U变为{e,f}。

- 顶点d的加入，可能使从U到V-U的最小代价边变化。因此，需要比较原最小代价边，以及从顶点g到V-U中各顶点的边，并用较小者作为新的最小代价边。
- 本轮中，(d,e)权值为9，代替原来不存在的边（权值较大）。
- 更新后，挑选最小代价边，将其依附的顶点加入集合U。挑最小代价边时，closedge[i].lowcost为0的边不需要考虑。本轮所有从U到V-U的边，代价最小的边是(d,e)。

closedge	1(b)	2(c)	3(d)	4(e)	5(f)	6(g)
adjvex(cos)	a(18)				a(19)	a(18)
adjvex(cos)	0	b(8)			a(19)	a(18)
adjvex(cos)	0	0	c(20)		a(19)	a(18)
adjvex(cos)	0	0	g(15)		g(15)	0
adjvex(cos)	0	0	0	d(9)	g(15)	0

普里姆算法



3. 将最小代价边(d,e)依附的顶点e加入集合U，其对应closedge[4]信息置零。V-U变为{f}。

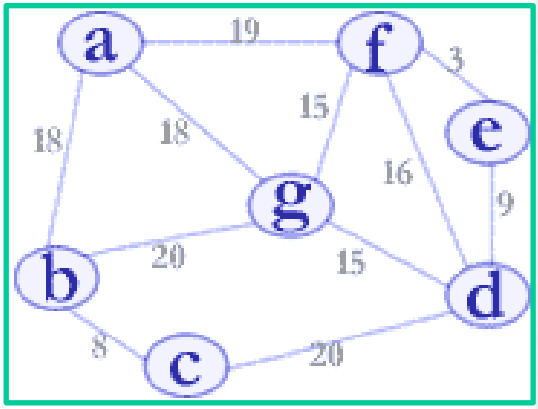
- 顶点e的加入，可能使从U到V-U的最小代价边变化。因此，需要比较原最小代价边，以及从顶点g到V-U中各顶点的边，并用较小者作为新的最小代价边。
- 本轮中，(e,f)权值为3，代替(g,f)。
- 至此，顶点f也加入集合U，最小代价边也确定，最小生成树构造完毕。

closedge	1(b)	2(c)	3(d)	4(e)	5(f)	6(g)
adjvex(cos)	a(18)				a(19)	a(18)
adjvex(cos)	0	b(8)			a(19)	a(18)
adjvex(cos)	0	0	c(20)		a(19)	a(18)
adjvex(cos)	0	0	g(15)		g(15)	0
adjvex(cos)	0	0	0	d(9)	g(15)	0
adjvex(cos)	0	0	0	0	e(3)	0

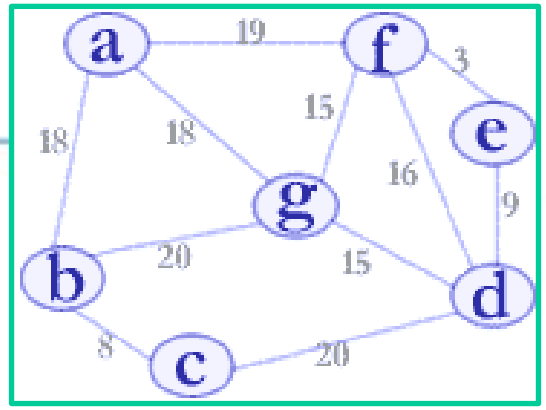
普里姆算法

● 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）
    // }closedge[MAX_VERTEX_NUM];
    k = LocateVex ( G, u );
    for ( j=0; j<G.vexnum; ++j )        // 辅助数组初始化，U{u}到V-U的最近距离
        if (j!=k) closedge[j] = { u, G.arcs[k][j].adj };    //{adjvex,lowcost}
    closedge[k].lowcost = 0;            // 初始状态，置lowcost=0，标识顶点u 属于U
    for (i=1; i<G.vexnum; ++i)
    {    // 选择其余 G.vexnum-1 个顶点
        k = minimum(closedge);   // 求出T的下一个结点(图中第k顶点)
    //用lowcost=0标识对应顶点在集合U中，此时应找到集合V-U的最小权重边，即
       closedge[k].lowcost = Min{ closedge[v_i].lowcost |
          closedge[v_i].lowcost>0, v_i∈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后更新到达V-U中顶点的
//最小边，因为U中顶点的lowcost为0，所以更新不会影响到U中顶点间的边
            closedge[j] = { G.vexs[k]， G.arcs[k][j].adj };
} // for
} // MiniSpanTree
```

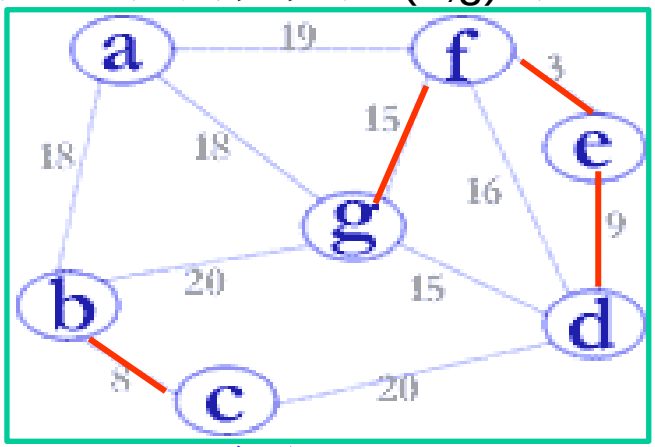
- 算法分析
  - 假设网中有n个顶点，则第一个进行初始化的循环语句的频度为n，第二个循环语句的频度为n-1。
  - 其中有两个内循环：其一是在closedge[v].lowcost中求最小值，其频度为n-1；其二是更新U中顶点到V-U中顶点的最小代价的边，其频度为n。
  - 由此，普里姆算法的时间复杂度为$O(n^2)$，与网中的边数无关，因此适用于求边稠密的网的最小生成树。

克鲁斯卡尔（Kruskal）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（Kruskal）算法

- **生成树上不允许有回路，因此并非每一条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flash\chap07\7-4-1.swf



连通网G5

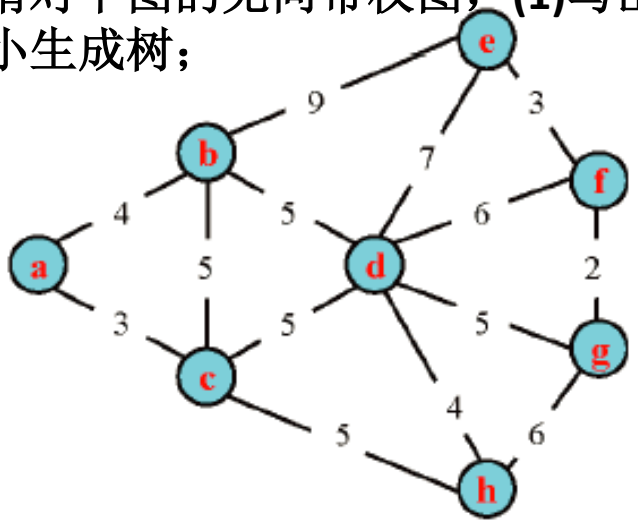
- **思考：在算法中如何判别当前权值最小边的两个顶点之间是否已经连通？**
  - 分析：从生成树的构造过程可见，初始态为 n 个顶点分属 n 棵树，互不连通，每加入一条边，就将两棵树合并为一棵树，在同一棵树上的两个顶点之间自然相连通。由此**判别当前权值最小边是否可取只要判别它的两个顶点是否在同一棵树上即可**。

克鲁斯卡尔（Kruskal）算法

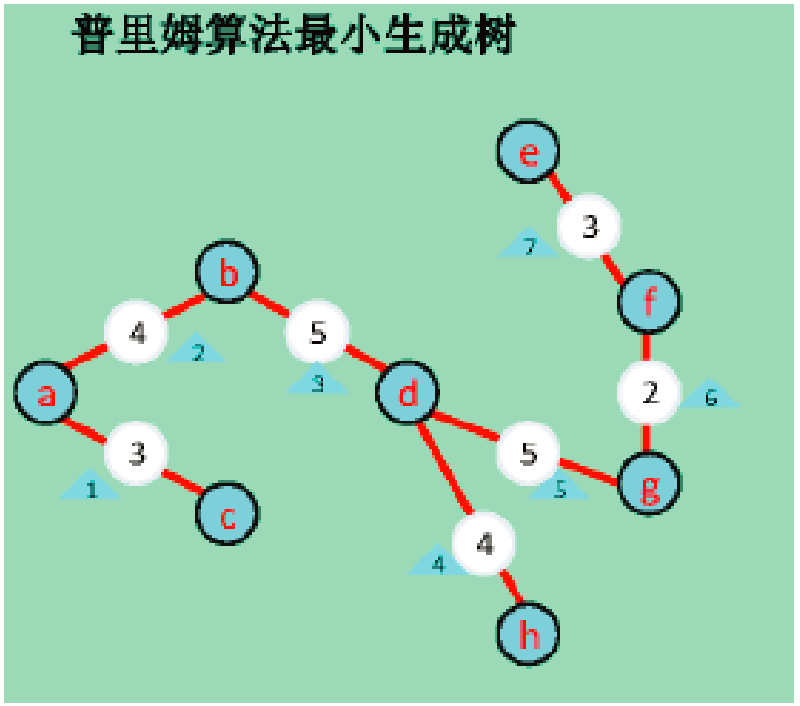
- 如果**不考虑建立图的存储结构所需时间**，则克鲁斯卡尔算法的时间复杂度为***O* (eloge)** (e为网中边的数目)，因此它相对于普里姆算法而言，**适合于求边稀疏的网的最小生成树**。

最小生成树算法练习

请对下图的无向带权图，(1)写出它的邻接矩阵，并按普里姆算法从顶点a起始，求其最小生成树；

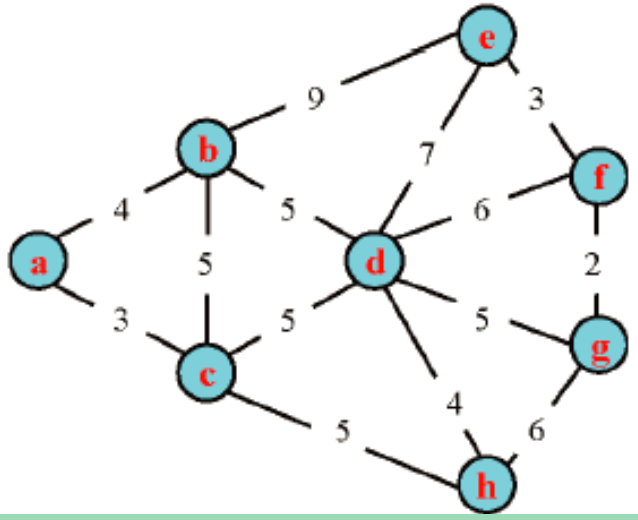


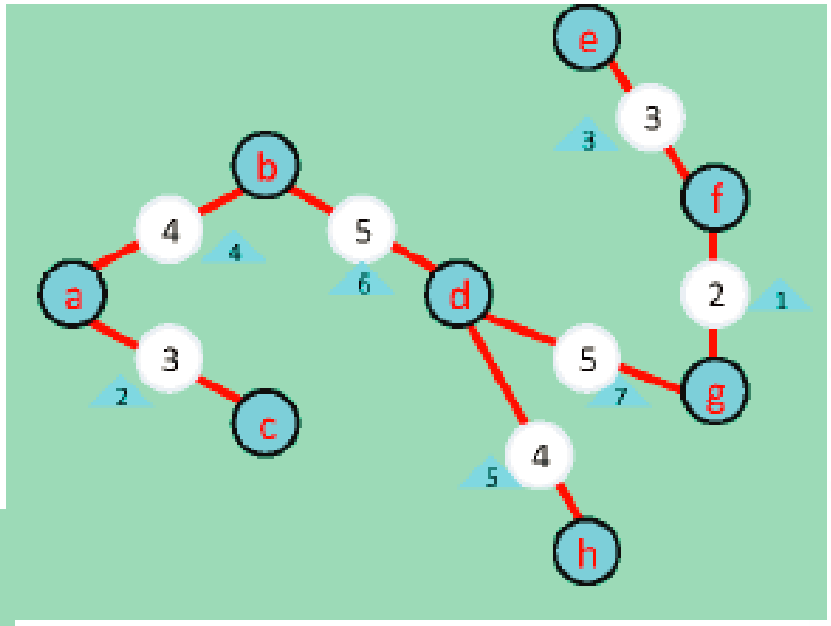
	a	b	c	d	e	f	g	h
a	∞	4	3	∞	∞	∞	∞	∞
b	4	∞	5	5	9	∞	∞	∞
C	3	5	∞	5	∞	∞	∞	5
d	∞	5	5	∞	7	6	5	4
e	∞	9	∞	7	∞	3	∞	∞
f	∞	∞	∞	6	3	∞	2	∞
g	∞	∞	∞	5	∞	2	∞	6
h	∞	∞	5	4	∞	∞	6	∞



最小生成树算法练习

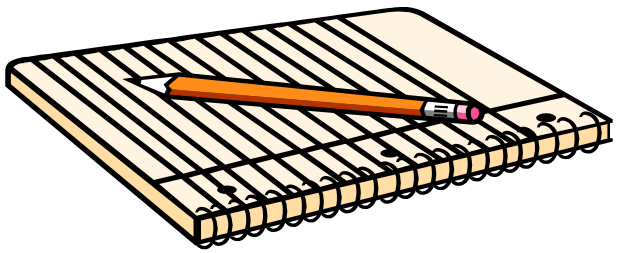
请对下图的无向带权图，(2)写出它的邻接表，并按克鲁斯卡尔算法求其最小生成树。





1	a	2	3	∧				
2	b	1	3	4	5	∧		
3	c	1	2	4	8	∧		
4	d	2	3	5	6	7	8	∧
5	e	2	4	6	∧			
6	f	4	5	7	∧			
7	g	4	6	8	∧			
8	h	3	4	7	∧			

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

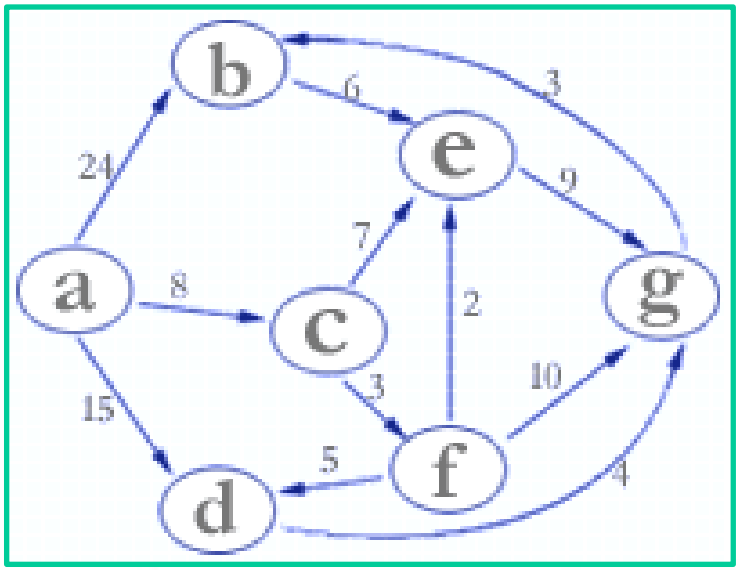
- 图或网还经常用于**描述一个城市或城市间的交通运输网络**，以图中顶点表示一个城市或某个交通枢纽，以边或弧表示两地之间的交通状况，边或弧上的权值可表示各种相关信息，例如两地之间的路程长度，交通费用或行程时间等等。
- 对于网来说，两个顶点之间的路径长度是路径中**“弧的权值之和”**。当两个顶点之间存在多条路径时，必然存在一条**“最短路径”**。
- 考虑到交通图的有向性，本节将讨论**带权有向图(有向网)**，并称路径中的第一个顶点为**“源点”**，路径中的最后一个顶点为**“终点”**。以下讨论两种最常见的路径问题。

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，

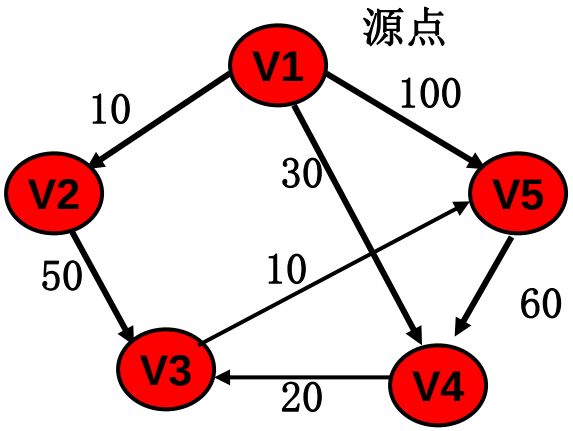
路径{a, d}的长度为15，

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



源点	终点	最短路径	路径长度
V_1	V_2	(V_1,V_2)	10
	V_3	(V_1,V_4,V_3)	50
	V_4	(V_1,V_4)	30
	V_5	(V_1,V_4,V_3,V_5)	60

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

从v1出发，到达各顶点的路径长度按短到长依次排序为：

- (v1,v2)，长度为10
- (v1,v4)，长度为30
- (v1,v4,v3)，长度为50
- (v1,v4,v3,v5)，长度为60，

注意：这里0、1、2、3、4用于标识结点 V1、V2、V3、V4、V5。

最短路径

- 如何求得这些最短路径？**迪杰斯特拉（Dijkstra）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**。
  - 引用辅助向量D，其每个分量D[i]表示，当前从始点v到每个终点v_i的最短路径。用V表示始点以外所有顶点集合。
  - 初始时，若v到v_i的有弧，则$D[i]$为若v到v_i的弧权值。
  - 显然，第一条最短路径为$D[j]=\min_i\{D[i]|v_i\in V\}$；
  - 第二条最短路径，即次短路径，只有两种可能，要么经由第一条最短路径，要么从始点直达；因此两种方式中，较短的路径为第二条最短路径；
  - 照此方式依次确定最短路径。

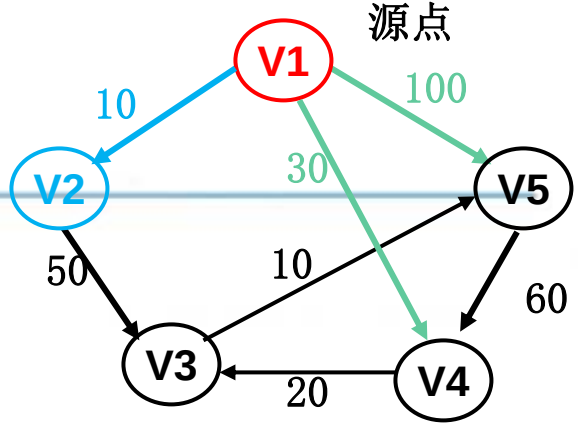
最短路径

- 现在证明迪杰斯特拉思路：**按长度递增次序确定最短路径，下一次最短路径，要么由始点直达，要么经由已确定最短路径的顶点到达**。
- 证明：
  - 不失一般性，设始点为v_0，用$D(v_i, v_j)$表示从顶点v_i到v_j的最短路径长度。已经确定了的前N条最短路径，其终点，按路径长度由短到长排列为，$v_{s_1}, v_{s_2}, \cdots, v_{s_N}$，即，$D(v_0, v_{s_1}) \leq D(v_0, v_{s_2}) \leq \cdots \leq D(v_0, v_{s_N})$，下一条最短路径到达顶点$v_k$，即$D(v_0, v_{s_N}) \leq D(v_0, v_k) \leq D(v_0, v_i), v_i \in V - \{v_{s_1}, v_{s_2}, \cdots, v_{s_N}, v_k\}$。那么：
  - $D(v_0, v_k) = \min\left(D\left(v_0, v_{s_j}\right) + arcs[s_j][k], arcs[0][k]\right)$，即$v_0$到$v_k$的最短路径要么从始点$v_0$直达，长度即为$v_0$到$v_k$的弧权重$arcs[0][k]$；要么经由已确定最短路径的顶点$v_{s_j}$（$v_{s_j} \in \{v_{s_1}, v_{s_2}, \cdots, v_{s_N}\}$）到达，长度$D\left(v_0, v_{s_j}\right) + arcs[s_j][k]$。

最短路径

- 现在证明迪杰斯特拉思路：**按长度递增次序确定最短路径，下一次最短路径，要么由始点直达，要么经由已确定最短路径的顶点到达**。
- 证明：
  - $D(v_0,v_k)=\min\left(D\left(v_0,v_{s_j}\right)+arcs[s_j][k],arcs[0][k]\right)$，即$v_0$到$v_k$的最短路径要么从始点$v_0$直达，长度即为$v_0$到$v_k$的弧权重$arcs[0][k]$；要么经由已确定最短路径的顶点$v_{s_j}$（$v_{s_j}\in\{v_{s_1},v_{s_2},\cdots,v_{s_N}\}$）到达，长度$D\left(v_0,v_{s_j}\right)+arcs[s_j][k]$。
  - 到达v_k的最短路径不存在第三种可能。若有，则存在顶点v_j（$v_j\notin\{v_{s_1},v_{s_2},\cdots,v_{s_N}\}$），且$D(v_0,v_k)=D(v_0,v_j)+arcs[j][k]$这与$D(v_0,v_{s_N})\leq D(v_0,v_k)\leq D(v_0,v_i)$（$v_i\in V-\{v_{s_1},v_{s_2},\cdots,v_{s_N},v_k\}$）矛盾，因此不成立。

最短路径



$(V_1,V_2)10$, $(V_1,V_4)30$， $(V_1,V_4,V_3)50$, $(V_1,V_4,V_3,V_5)60$,

- 迪杰斯特拉（Dijkstra）算法示例。
  - 从始点v_1出发
  - 初始时，若v到v_i的有弧，则$D[i]$为若v到v_i的弧权值。
  - 显然，第一条最短路径为$D[j]=\min_i\{D[i]|v_i\in V\}$;

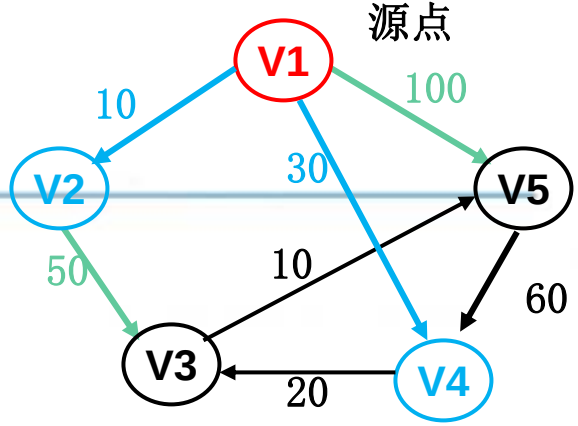
轮次	0（初始）	1	2	3	4
v_2	(V1 ,V2)10				
v_3	（）Inf				
v_4	(V1 ,V4)30				
v_5	(V1 ,V5)100				
v_j	V2				
S	{V1,V2}				

v_j：下一最短路径到达顶点

可见，第一条最短路径为(V1 ,V2)，将其所关联的顶点V2加入S，S变为{V1,V2}

S：已确定的最短路径集合

最短路径



$(V_1,V_2)10$, $(V_1,V_4)30$， $(V_1,V_4,V_3)50$, $(V_1,V_4,V_3,V_5)60$,

- 迪杰斯特拉（Dijkstra）算法示例。
  - 下一条最短路径要么经由S（已确定的最短路径），要么从始点直达。因为D中已保存了当前到达各个顶点的最短路径了，所以只需要比较**经由最新进入S的顶点v_j的路径**是否更短即可。

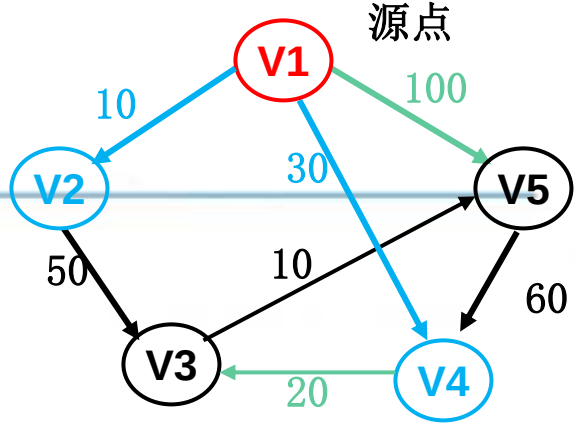
轮次	0（初始）	1
v_2	(V1 ,V2)10	
v_3	（）Inf	(V1 ,V2,V3)60
v_4	(V1 ,V4)30	(V1 ,V4)30
v_5	(V1 ,V5)100	(V1 ,V5)100
v_j	V2	V4
S	{V1,V2}	{V1,V2,V4}

v_j：下一最短路径到达顶点

S：已确定的最短路径集合

当前轮次：
1.到达V3的最短路径，比较了经由V2的路径(V1 ,V2,V3)和V1V3之间的弧。
2.到达顶点V4,V5的路径也做了类似比较。
3.此时的最短路径（V1,V4）为第二条最短路径，V4加入集合S。

最短路径



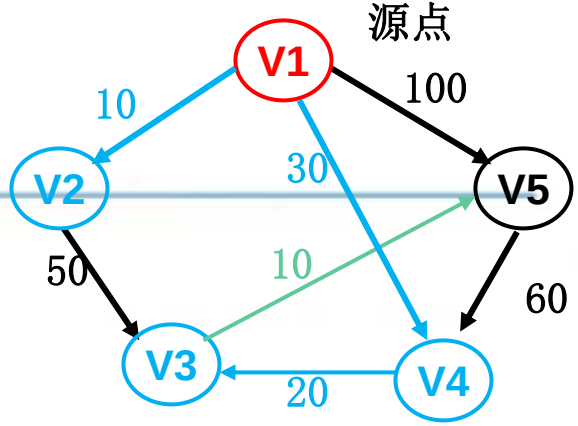
$(V_1,V_2)10$, $(V_1,V_4)30$， $(V_1,V_4,V_3)50$, $(V_1,V_4,V_3,V_5)60$,

- **迪杰斯特拉（Dijkstra）算法示例。**
  - 下一条最短路径要么经由S（已确定的最短路径），要么从始点直达。因为D中已保存了当前到达各个顶点的最短路径了，所以只需要比较经由最新进入S的顶点v_j的路径是否更短即可。

轮次	0（初始）	1	2
v_2	(V1 ,V2)10		
v_3	（） Inf	(V1 ,V2,V3)60	(V1 ,V4,V3)50
v_4	(V1 ,V4)30	(V1 ,V4)30	
v_5	(V1 ,V5)100	(V1 ,V5)100	(V1 ,V5)100
v_j	V2	V4	V3
S	{V1,V2}	{V1,V2,V4}	{V1,V2,V4,V3}

当前轮次：到达各顶点的最短路径，已经与经由V2的路径以及直达路径比较过了，所以只需关注顶点V4，比较当前最短，以及经由V4的路径长度即可。
1.到达V3的最短路径，经由V4的路径(V1 ,V4,V3)更短。
2.到达顶点V5的路径也做了类似比较。
3.此时的最短路径（V1,V4,V3）为第三条最短路径，V3加入集合S

最短路径



$(V_1,V_2)10$, $(V_1,V_4)30$， $(V_1,V_4,V_3)50$, $(V_1,V_4,V_3,V_5)60$,

- 迪杰斯特拉（Dijkstra）算法示例。
  - 下一条最短路径要么经由S（已确定的最短路径），要么从始点直达。因为D中已保存了当前到达各个顶点的最短路径了，所以只需要比较经由最新进入S的顶点v_j的路径是否更短即可。

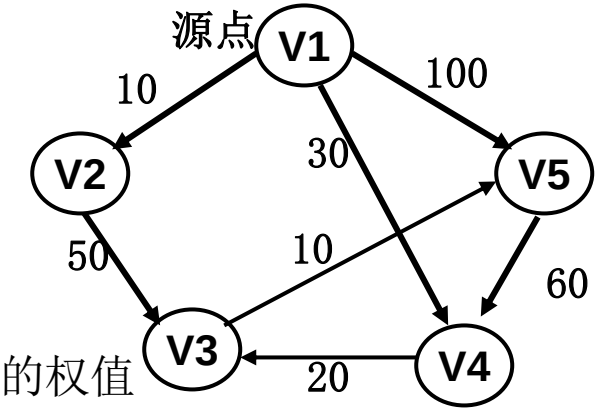
轮次	0（初始）	1	2	3
v_2	(V1 ,V2)10			
v_3	（） Inf	(V1 ,V2,V3)60	(V1 ,V4,V3)50	
v_4	(V1 ,V4)30	(V1 ,V4)30		
v_5	(V1 ,V5)100	(V1 ,V5)100	(V1 ,V5)100	(V1,V4,V3,V5) 60
v_j	V2	V4	V3	V5
S	{V1,V2}	{V1,V2,V4}	{V1,V2,V4,V3}	{V1,V2,V4,V3,V5}

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flash\chap07\7-5-0.swf
  - 从这个过程可见，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：

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

$$P[i]=\begin{cases}\text{空串} & \text{当}i=s\text{或}<v_s,v_i>\notin E\\ "v_sv_i" & \text{当}i\neq s\text{且}<v_s,v_i>\in E\end{cases}$$

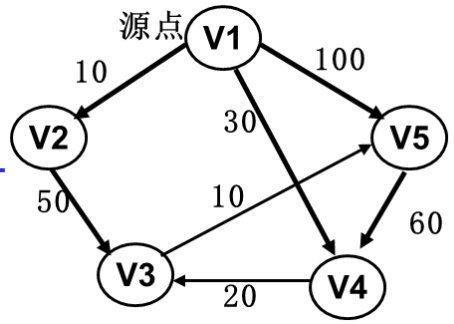
  - (2) 选择顶点v_j使得 D[j]=Min{D[k]| v_k∈V-S}，并将顶点v_j并入到集合S中；
  - (3) 对集合V-S中所有顶点v_k，若存在从v_j指向该顶点的弧，且D[j]+w_{jk}<D[k]，则更新 D[k]为D[j]+w_{jk} ,并更新 P[k]为P[j]U{k}。
  - (4) 重复(2)和(3)直至求得从源点到所有其它顶点的最短路径为止。

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∈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[v0][i];
  for(w=0;w<G.vexnum;++w)      P[v][w]=FALSE;   // 设空路径
                          //从v0到v当前最短路径包括顶点v0和v
  if (D[v] < INFINITY)  { P[v][v0]=TRUE;  P[v][v]=TRUE; }
 } // for
```

初始化

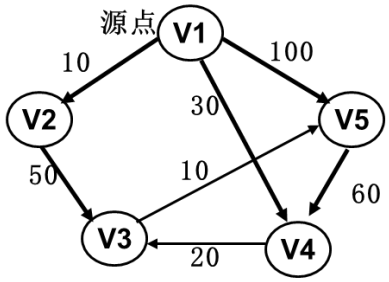



```
 D[v0] = 0; final[v0] = TRUE; // 初始化，顶点v0属于S集
// 开始主循环，每次求得v0到某个v顶点的最短路径，并加v到S集
for(i=1;i<G.vexnum;++i){          // 其余G.vexnum-1个顶点
  min  = INFINITY;              // 当前所知离v0 顶点的最近距离
  for(w=0;w<G.vexnum;++w)
        if(!final[w])                    // w顶点在V-S中
    if(D[w]<min)  {v=w;   min=D[w];}      // w顶点离v0顶点更近
   final[v]=TRUE;                        // 离v0顶点最近的v加入S集
```

从V_0到其他顶点的当前路径中，选择长度最短的路径。

迪杰斯特拉算法

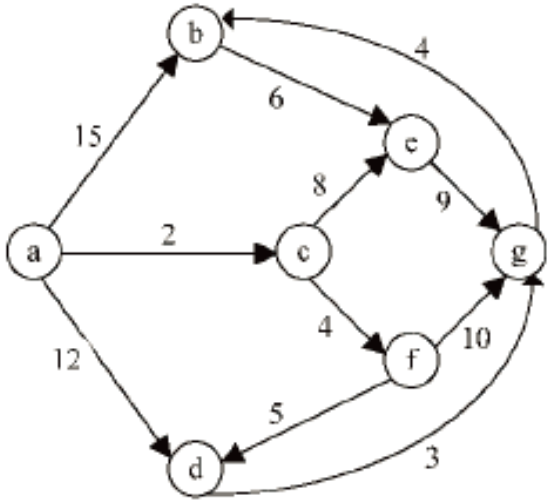
```
    for(w=0;w<G.vexnum;++w)        // 更新当前最短路径及距离
      if(!final[w]&&(min+G.arcs[v][w]<D[w])){  // 修改D[w]和P[w]，
                                                w∈V-S
            D[w]=min+G.arcs[v][w];
            P[w]=P[v];      P[w][w]=TRUE;     // P[w]=P[v]+[v][w]
      } // if
    } // for
}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

迪杰斯特拉算法练习

试利用Dijkstra算法求下图中从顶点a到其他各顶点间的最短路径，写出执行算法过程中各步的状态



终点 / Dist	b	c	d	e	f	g	S (终点集)
K=1	15 <a, b>	2 <a,c>	12 <a,d>	∞	∞	∞	{a, c}
K=2	15 <a,b>	∞	12 <a,d>	10 <a,c,e>	6 <a,c,f>	∞	{a, c, f}
K=3	15 <a, b>	∞	11 <a,c,f,d>	10 <a,c,e>	∞	16 <a,c,f,g>	{a, c, f, e}
K=4	15 <a,b>	∞	11 <a,c,f,d>	∞	∞	16 <a,c,f,g>	{a, c, f, e, d}
K=5	15 <a, b>	∞	∞	∞	∞	14 <a,c,f,d,g>	{a, c, f, e, d, g}
K=6	15 <a,b>	∞	∞	∞	∞	∞	{a, c, f, e, d, g, b}

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逐步试探，每次增加一个顶点进入中间顶点集合，计算此时的最短路径。算法求得一个 n 阶方阵的序列

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

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

- $D^{(-1)}[i][j]=G.arcs[i][j]$，
- $D^{(k)}[i][j]$ 则中间顶点集合为$\{v_0,v_1, \ldots ,v_k\}$时，从v_i到v_j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。

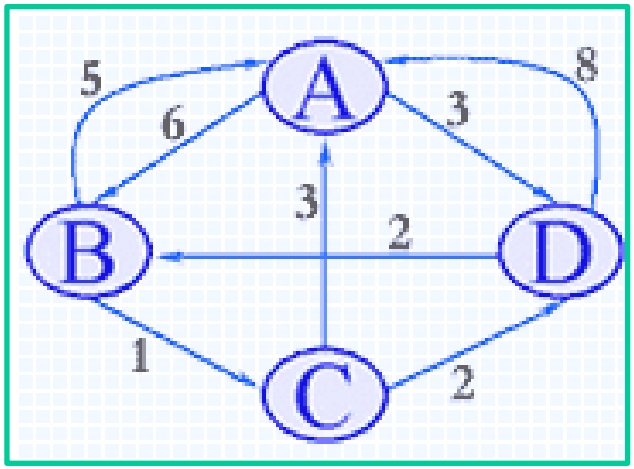
最短路径

- 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

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

- 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]
}
```

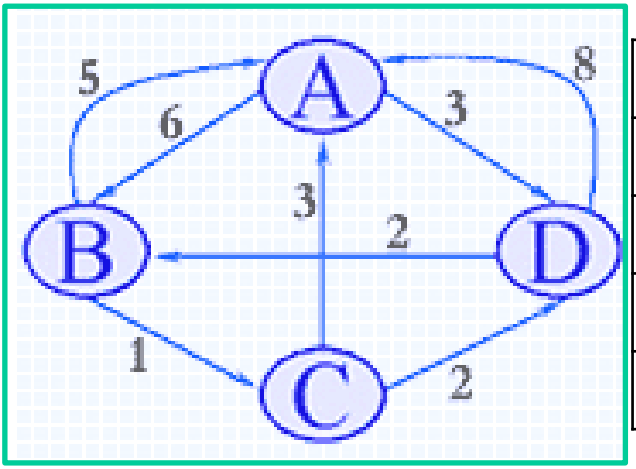


有向网G7

  - ➔ 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flash\chap07\7-5-1.swf

最短路径



$D^{(-1)}$

	A	B	C	D
A	0	6(AB)	inf	3(AD)
B	5(BA)	0	1(BC)	Inf
C	3(CA)	inf	0	2(CD)
D	8(DA)	2(DB)	inf	0

$D^{(0)}$：加入顶点A，从A出发的最短路径（第1行），和到达A的最短路径（第1列）已经包含顶点A了，因此路径不会变化，考查其他位置即可。

	A	B	C	D
A	0	6(AB)	inf	3(AD)
B	5(BA)	0	1(BC)	Inf 8(BAD)
C	3(CA)	Inf 9(CAB)	0	2(CD)
D	8(DA)	2(DB)	inf	0

$D^{(-1)}(BA)+D^{(-1)}(AC)=5+inf>D^{(-1)}(BC)$,因此，$D^{(0)}(BC)=D^{(-1)}(BC)$

$D^{(-1)}(BA)+D^{(-1)}(AD)=5+3<D^{(-1)}(BD)$,因此，$D^{(0)}(BD)=8$

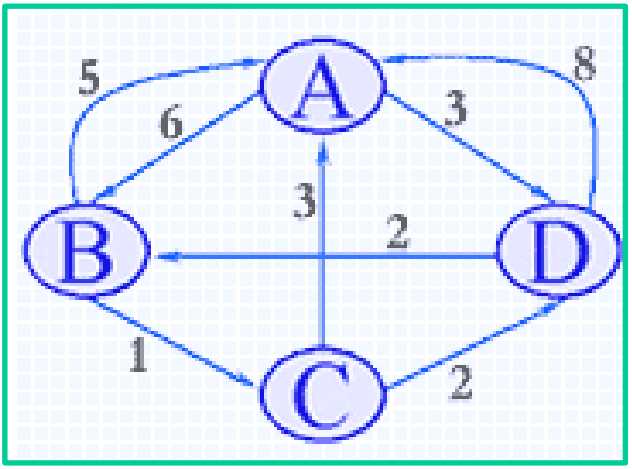
$D^{(-1)}(CA)+D^{(-1)}(AB)=3+6<D^{(-1)}(CB)$,因此，$D^{(0)}(CB)=9$

$D^{(-1)}(CA)+D^{(-1)}(AD)=3+3>D^{(-1)}(CD)$,因此，$D^{(0)}(CD)=D^{(-1)}(CD)$

$D^{(-1)}(DA)+D^{(-1)}(AB)=8+6>D^{(-1)}(DB)$,因此，$D^{(0)}(DB)=D^{(-1)}(DB)$

$D^{(-1)}(DA)+D^{(-1)}(AC)=8+inf==D^{(-1)}(DC)$,因此，$D^{(0)}(DC)=D^{(-1)}(DC)$

最短路径



$D^{(0)}$

	A	B	C	D
A	0	6(AB)	inf	3(AD)
B	5(BA)	0	1(BC)	8(BAD)
C	3(CA)	9(CAB)	0	2(CD)
D	8(DA)	2(DB)	inf	0

$D^{(1)}$：加入顶点B，从B出发的最短路径（第2行），和到达B的最短路径（第2列）已经包含顶点B了，因此路径不会变化，考查其他位置即可。

	A	B	C	D
A	0	6(AB)	Inf 7(ABC)	3(AD)
B	5(BA)	0	1(BC)	8(BAD)
C	3(CA)	9(CAB)	0	2(CD)
D	8(DA) 7(DBA)	2(DB)	Inf 3(DBC)	0

$D^{(0)}(AB)+D^{(0)}(BC)=6+1<D^{(0)}(AC)$,因此，$D^{(1)}(AC)=7$

$D^{(0)}(AB)+D^{(0)}(BD)=6+8>D^{(0)}(AD)$,因此，$D^{(1)}(AD)=D^{(0)}(AD)$

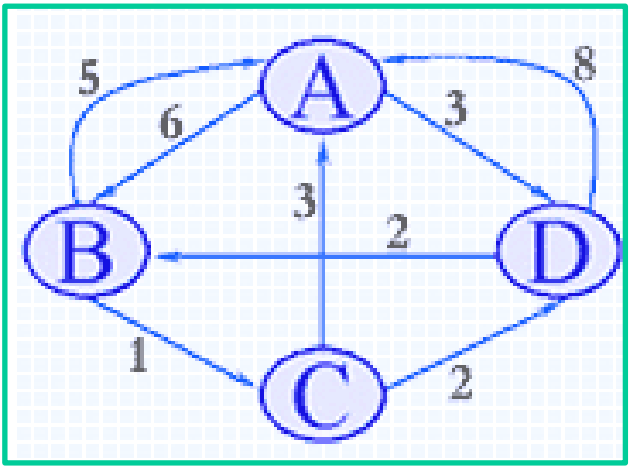
$D^{(0)}(CB)+D^{(0)}(BA)=9+5>D^{(0)}(CA)$,因此，$D^{(1)}(CA)=D^{(0)}(CA)$

$D^{(0)}(CB)+D^{(0)}(BD)=9+8>D^{(0)}(CD)$,因此，$D^{(1)}(CD)=D^{(0)}(CD)$

$D^{(0)}(DB)+D^{(0)}(BA)=2+5<D^{(0)}(DA)$,因此，$D^{(1)}(DA)=7$

$D^{(0)}(DB)+D^{(0)}(BC)=2+1<D^{(0)}(DC)$,因此，$D^{(1)}(DC)=3$

最短路径



$D^{(1)}$

	A	B	C	D
A	0	6(AB)	7(ABC)	3(AD)
B	5(BA)	0	1(BC)	8(BAD)
C	3(CA)	9(CAB)	0	2(CD)
D	7(DBA)	2(DB)	3(DBC)	0

$D^{(2)}$：加入顶点C，从C出发的最短路径（第3行），和到达C的最短路径（第3列）已经包含顶点C了，因此路径不会变化，考查其他位置即可。

	A	B	C	D
A	0	6(AB)	7(ABC)	3(AD)
B	5(BA) 4(BCA)	0	1(BC)	8(BAD) 3(BCD)
C	3(CA)	9(CAB)	0	2(CD)
D	7(DBA) 6(DBCA)	2(DB)	3(DBC)	0

$D^{(1)}(AC)+D^{(1)}(CB)=7+9>D^{(1)}(AB)$,因此，$D^{(2)}(AB)=D^{(1)}(AB)$

$D^{(1)}(AC)+D^{(1)}(CD)=7+2>D^{(1)}(AD)$,因此，$D^{(2)}(AD)=D^{(1)}(AD)$

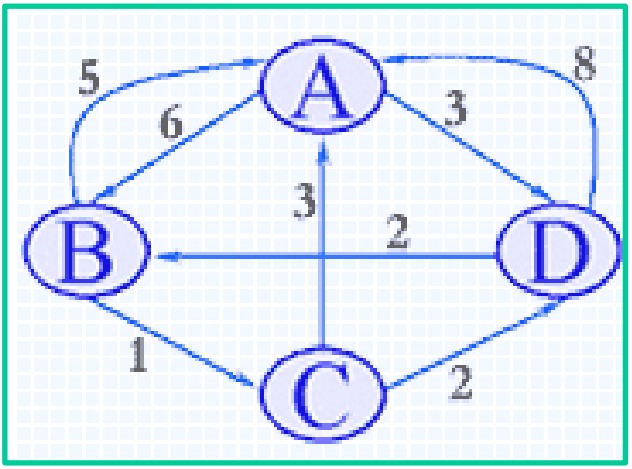
$D^{(1)}(BC)+D^{(1)}(CA)=1+3<D^{(1)}(BA)$,因此，$D^{(2)}(BA)=4$

$D^{(1)}(BC)+D^{(1)}(CD)=1+2<D^{(1)}(BD)$,因此，$D^{(2)}(BD)=3$

$D^{(1)}(DC)+D^{(1)}(CA)=3+3<D^{(1)}(DA)$,因此，$D^{(2)}(DA)=6$

$D^{(1)}(DC)+D^{(1)}(CB)=3+9>D^{(1)}(DB)$,因此，$D^{(2)}(DB)=D^{(1)}(DB)$

最短路径



$D^{(2)}$

	A	B	C	D
A	0	6(AB)	7(ABC)	3(AD)
B	4(BCA)	0	1(BC)	3(BCD)
C	3(CA)	9(CAB)	0	2(CD)
D	6(DBCA)	2(DB)	3(DBC)	0

$D^{(3)}$：加入顶点D，从D出发的最短路径（第4行），和到达D的最短路径（第4列）已经包含顶点D了，因此路径不会变化，考查其他位置即可。

	A	B	C	D
A	0	6(AB) 5(ADB)	7(ABC) 6(ADBC)	3(AD)
B	4(BCA)	0	1(BC)	3(BCD)
C	3(CA)	9(CAB) 4(CDB)	0	2(CD)
D	6(DBCA)	2(DB)	3(DBC)	0

$D^{(2)}(AD)+D^{(2)}(DB)=3+2<D^{(2)}(AB)$,因此，$D^{(3)}(AB)=5$

$D^{(2)}(AD)+D^{(2)}(DC)=3+3<D^{(2)}(AC)$,因此，$D^{(3)}(AC)=6$

$D^{(2)}(BD)+D^{(2)}(DA)=3+6>D^{(2)}(BA)$,因此，$D^{(3)}(BA)=D^{(2)}(BA)$

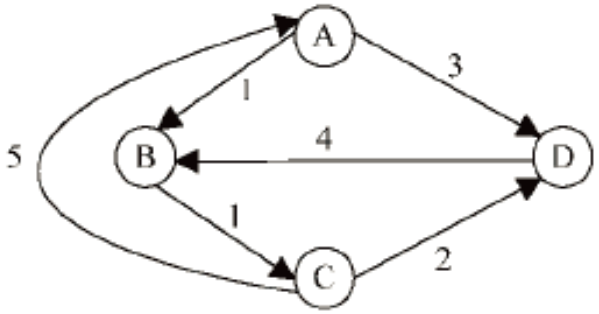
$D^{(2)}(BD)+D^{(2)}(DC)=3+3>D^{(2)}(BC)$,因此，$D^{(3)}(BC)=D^{(2)}(BC)$

$D^{(2)}(CD)+D^{(2)}(DA)=2+6>D^{(2)}(CA)$,因此，$D^{(3)}(CA)=D^{(2)}(CA)$

$D^{(2)}(CD)+D^{(2)}(DB)=2+2<D^{(2)}(CB)$,因此，$D^{(3)}(CB)=4$

Floyd算法练习

试利用Floyd算法求下图所示有向图中各对顶点之间的最短路径



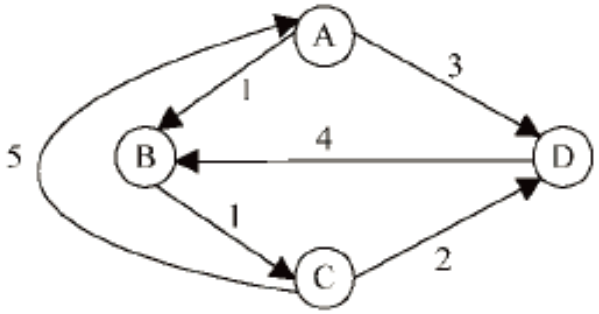
D(-1)	A	B	C	D
A	0	1(AB)	INF	3(AD)
B	INF	0	1(BC)	INF
C	5(CA)	INF	0	2(CD)
D	INF	4(DB)	INF	0

D(0)	A	B	C	D
A	0	1(AB)	INF	3(AD)
B	INF	0	1(BC)	INF
C	5(CA)	6(CAB)	0	2(CD)
D	INF	4(DB)	INF	0

D(1)	A	B	C	D
A	0	1(AB)	2(ABC)	3(AD)
B	INF	0	1(BC)	INF
C	5(CA)	6(CAB)	0	2(CD)
D	INF	4(DB)	5(DBC)	0

Floyd算法练习

试利用Floyd算法求下图所示有向图中各对顶点之间的最短路径



D(1)	A	B	C	D
A	0	1(AB)	2(ABC)	3(AD)
B	INF	0	1(BC)	INF
C	5(CA)	6(CAB)	0	2(CD)
D	INF	4(DB)	5(DBC)	0

D(2)	A	B	C	D
A	0	1(AB)	2(ABC)	3(AD)
B	6(BCA)	0	1(BC)	3(BCD)
C	5(CA)	6(CAB)	0	2(CD)
D	10(DBCA)	4(DB)	5(DBC)	0

D(3)	A	B	C	D
A	0	1(AB)	2(ABC)	3(AD)
B	6(BCA)	0	1(BC)	3(BCD)
C	5(CA)	6(CAB)	0	2(CD)
D	10(DBCA)	4(DB)	5(DBC)	0

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

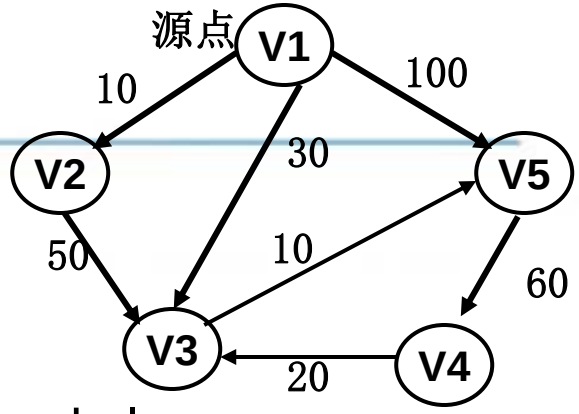
- **知识点**

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。

本章练习



- 已知有向网如右图所示，请：

1. 若将所有弧改为边，请分别使用Prim和Kruskal算法找到最小生成树，并逐一写出步骤。
2. 若源点为V1，请使用Dijkstra算法，找出V1到其他顶点的最短路径，并逐一写出步骤。
3. 请使用Floyd算法，找出有向网各个顶点之间的最短路径，并逐一写出步骤。