第七章　图(Graph)

7.1 图的定义和术语

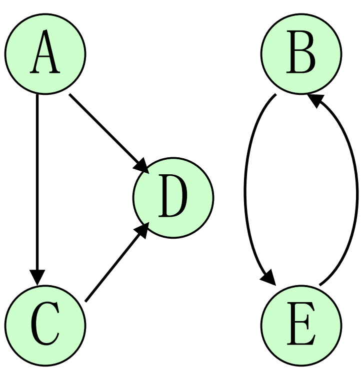
- **图**

图G由顶点集V和关系集E组成,记为:

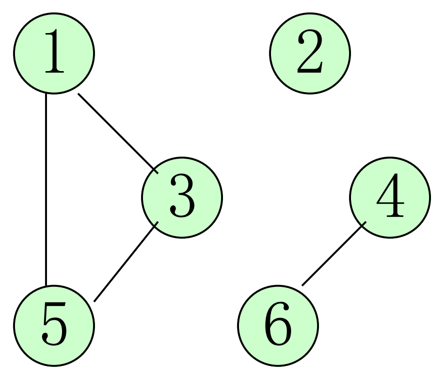
G=(V,E)

V是顶点(元素)的有穷非空集,

E是两个顶点之间的关系的集合。



G1



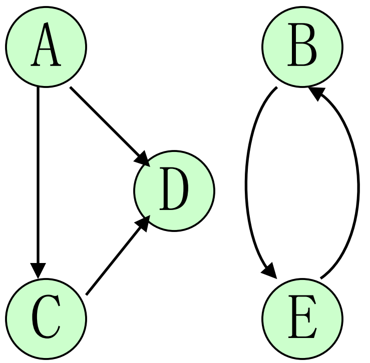
G2

● 若图G任意两顶点a, b之间的关系为有序对<a, b>, **<a, b>**∈E， 则称<a, b>为从a到b的一条**弧/有向边**；

其中： a是<a, b>的弧尾，

b是<a, b>的弧头；

称该图G是**有向图**。



G1

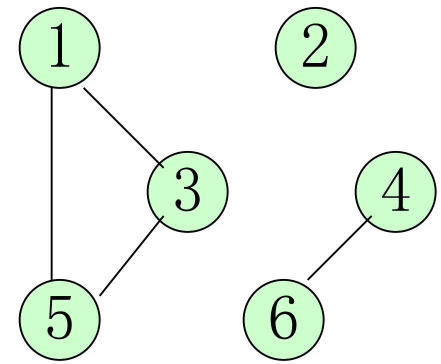
例 G1={V1, E1}， V1={A, B, C, D, E}

E1={<A, C>, <A, D>, <C, D>,

<B, E>, <E, B>}

- 若图G的任意两顶点a, b之间的关系为无序对(a, b)， 则称(a, b)为**无向边（边）**，称该图G是无向图。 无向图可简称为图。

(a, b)**依附**于a和b， (a, b)与a和b**相关联**



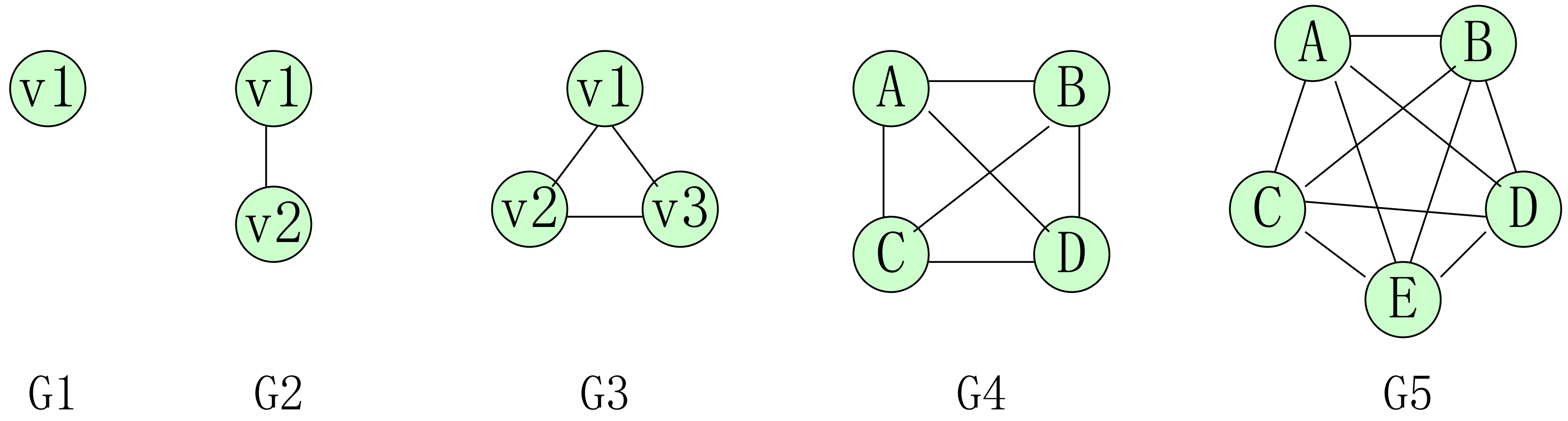
G2

例　G2={V2, E2},

V2={1, 2, 3, 4, 5, 6},

E2={(1, 3), (1, 5), (3, 5), (4, 6)}

- **完全图**----有n个顶点和**n(n-1)/2**条边的无向图



e=1(1-1)/2 =0

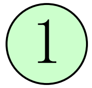
e=2(2-1)/2 =1

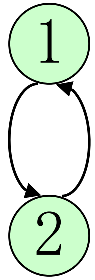
e=3(3-1)/2 =3

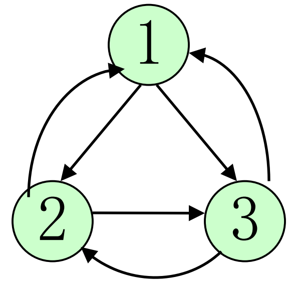
e=4(4-1)/2 =6

e=5(5-1)/2 =10

- **有向完全图**----有n个顶点和n(n-1)条弧的有向图。







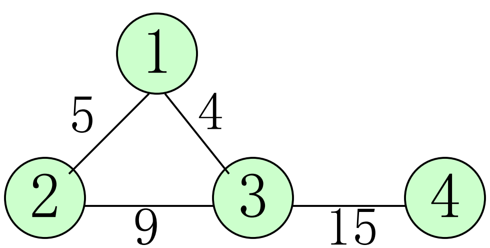
G1　　G2　　G3

e=1(1-1)
=0

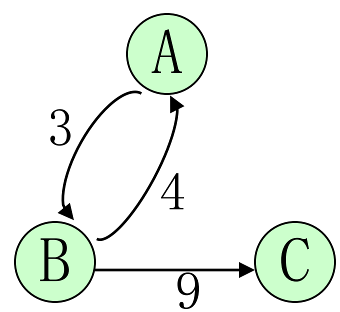
e=2(2-1)
=2

e=3(3-1)
=6

- **网**(Network)----边(弧)上加**权**(weight)的图。

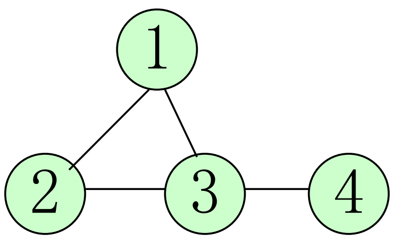


无向网G1

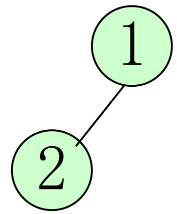


有向网G2

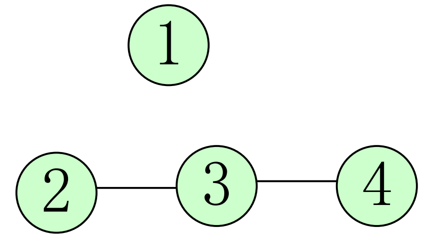
- 对图 G=(V, E) 和G’=(V’, E’)，
  若V’⊆V 且 E’⊆E，则称G’是G的一个**子图**



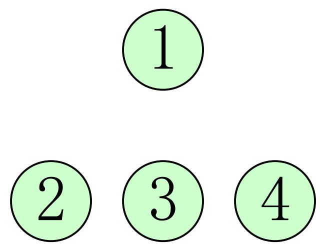
G



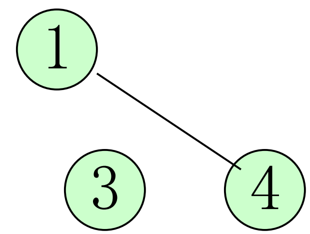
G1



G2



G3

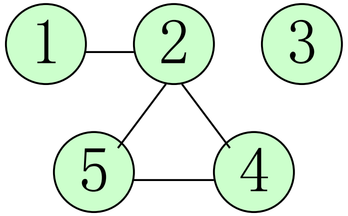


G4

G1, G2, G3是G的子图　G4不是G的子图

- 与顶点x相关联的边(x, y)的数目，称为x的**度**，
  记作TD(x) 或D(x)，

TD(1)=1
TD(2)=3
TD(3)=0



G1

- 以顶点x为弧尾的弧<x, y>的数目，称为x的**出度**，记作OD(x)。

  OD(A)=1

  OD(B)=2

  OD(C)=0

- 以顶点x为弧头的弧<y, x>的数目，称为x的**入度**，记作ID(x)。

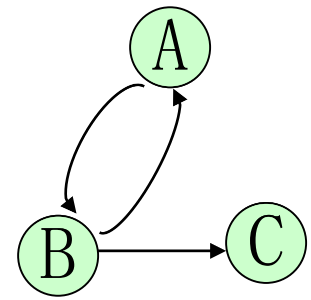
  ID(A)=1

  ID(B)=1

  ID(C)=1

  TD(A)=OD(A)+ID(A)=2

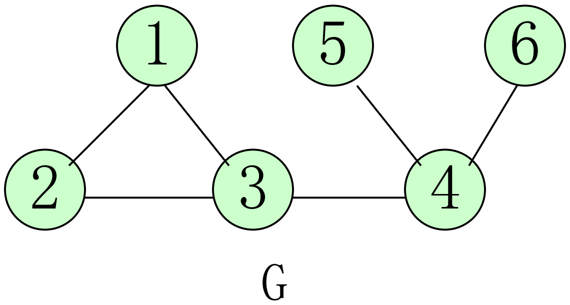
  TD(B)=OD(B)+ID(B)=3



G2

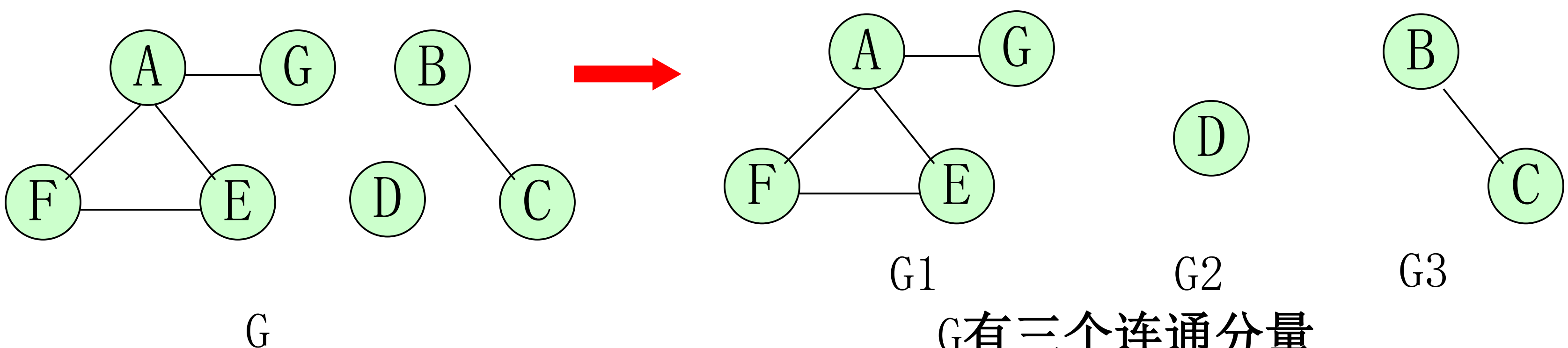
对无向图G：

- 若从顶点vi到vj有路径，则称vi和vj是**连通**的。
- 若图G中任意两顶点是连通的，则称G是**连通图**。



G

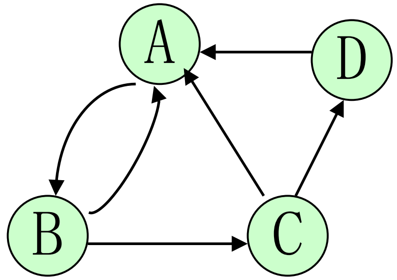
- 若图G’是G的一个**极大连通子图**，则称G’是G的一个**连通分量**。**（连通图的连通分量是自身）**



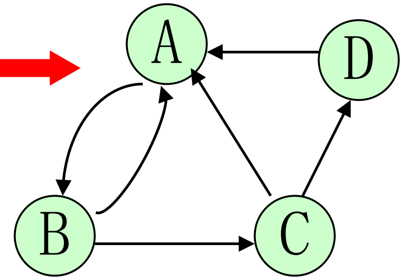
对有向图G

- 若在图G中,每对顶点vi和vj之间，从vi到vj,且从vj到vi都存在路径,则称G是**强连通图**。

- 若图G’是G的一个**极大强连通子图**,则称G’是G的一个**强连通分量**。(**强连通图的强连通分量是自身**)

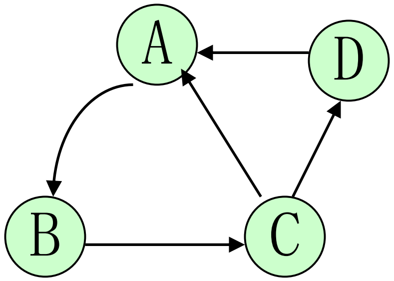


G1



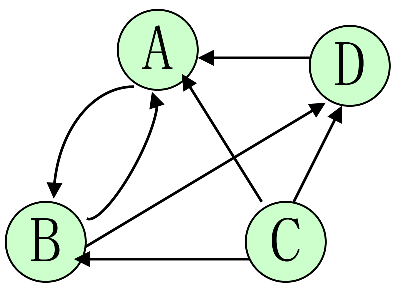
G11

是G1的强连通分量

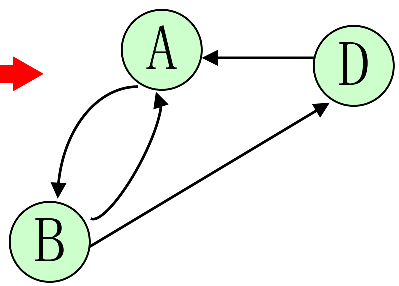


G12

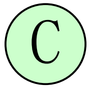
不是G1的强连通分量



G2



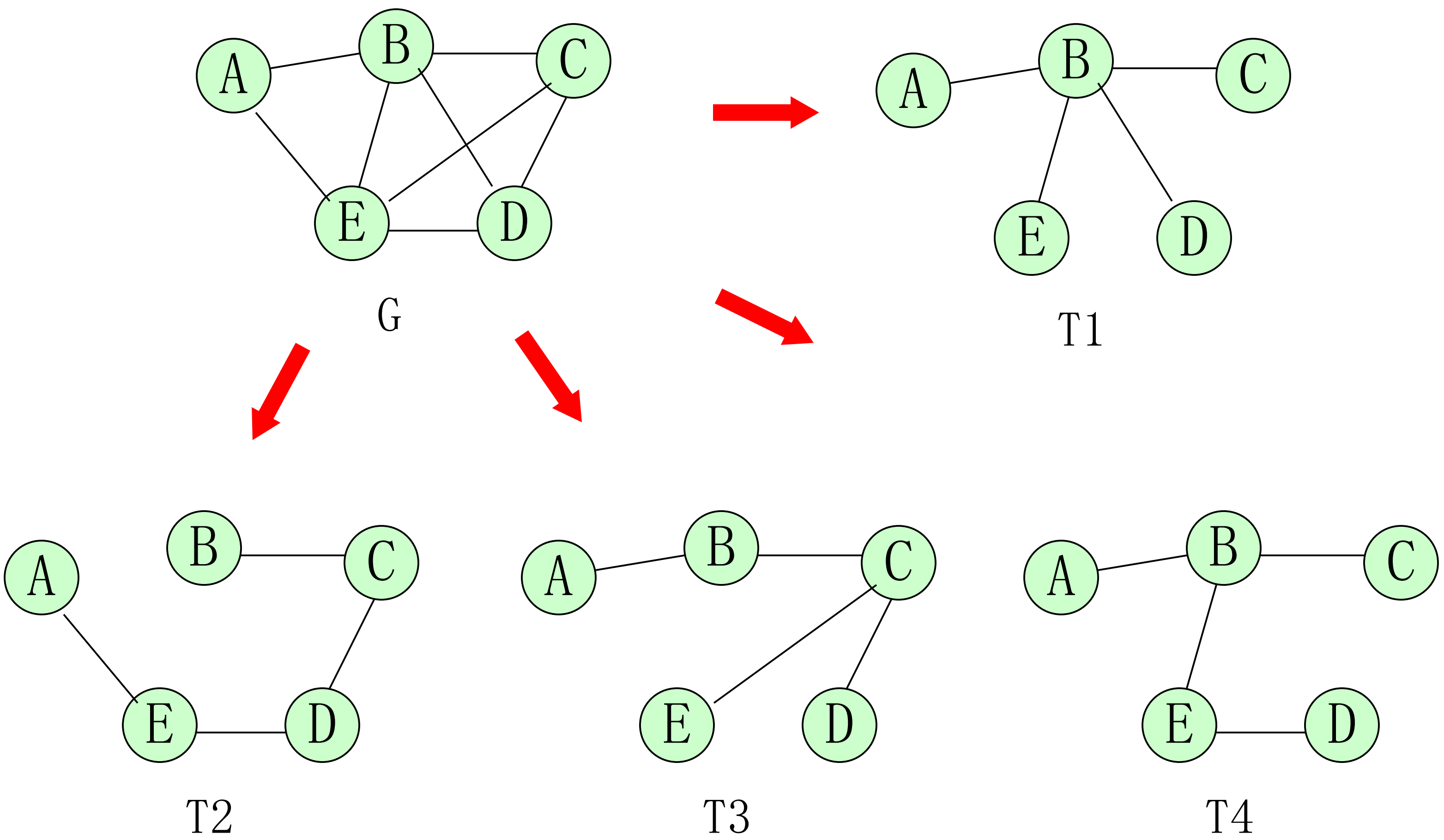
G21



G22

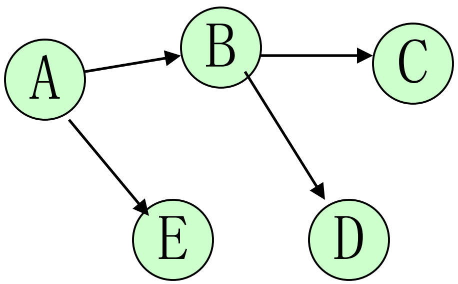
G2有两个强连通分量

- 设G=(V,E),G'=(V',E'),V=V',若G是连通图,G'是G的一个**极小连通子图**,则G'是G的一棵**生成树**。

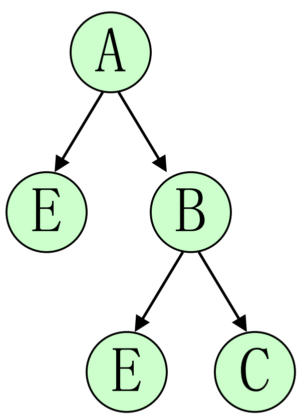


G的多棵生成树

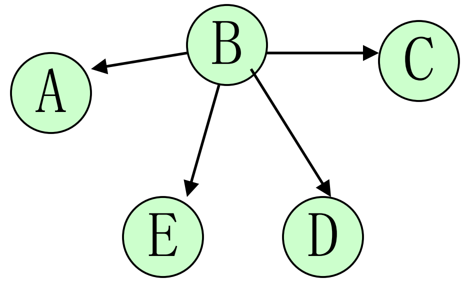
- 若有向图G有且仅有一个顶点的入度为0，其余顶点的入度为1，则G是一棵**有向树**。



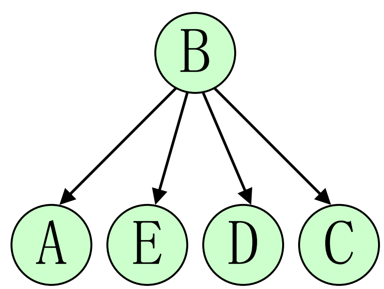
T1



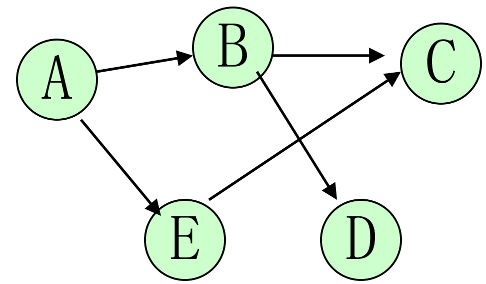
T3



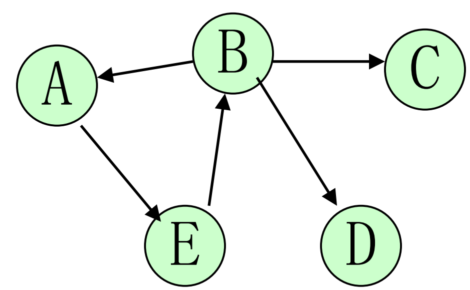
T2



T4



T5

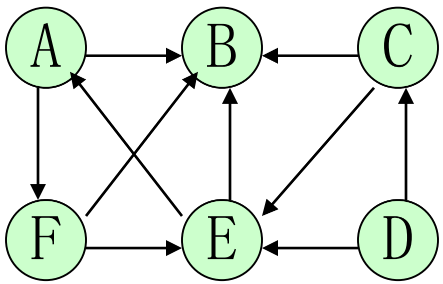


T6

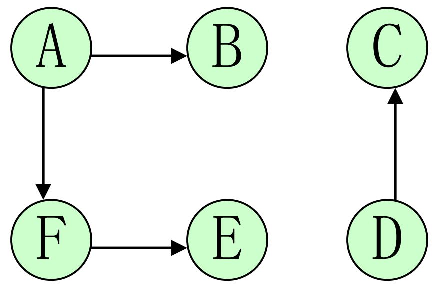
T1, T2, T3, T4是**有向树**

T5, T6不是**有向树**

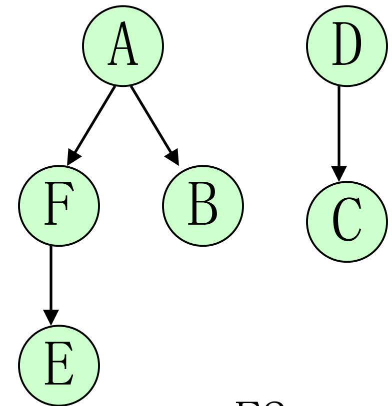
- 有向图的生成树/生成森林。



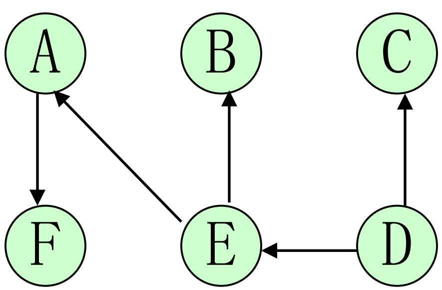
G



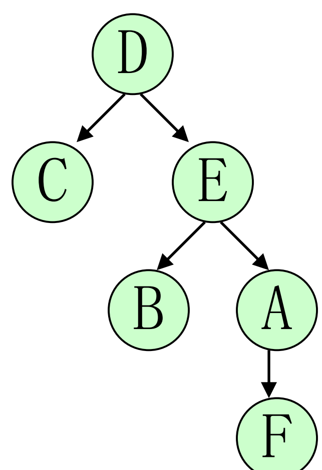
F1



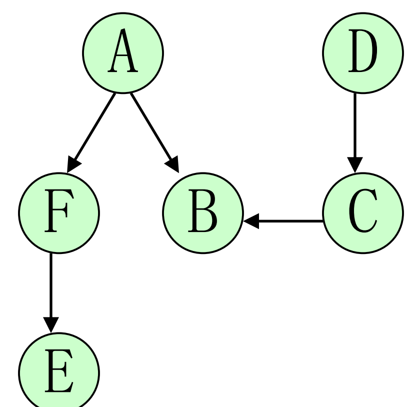
F2



T1



T2



T3

- 图的操作
  - 生成/消除一个图
  - 加入一个顶点/边(弧)
  - 遍历图
  - 求生成树

  ……

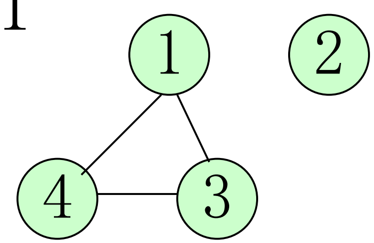
7.2 图的存储结构

7.2.1 数组表示法/邻接矩阵

顶点数组---用一维数组存储顶点(元素)

邻接矩阵---用二维数组存储顶点(元素)之间的关系(边或弧)

例1



G

V[1..4]

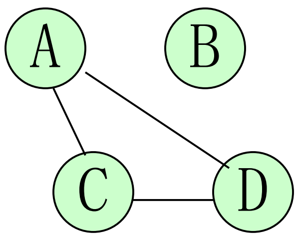
1	2	3	4
1	2	3	4

顶点数组

$$
M=\begin{array}{c c}
 & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}
\end{array}
$$

邻接矩阵

例2



G

V[1..4]

A	B	C	D
1	2	3	4

顶点数组

$$M=\begin{array}{c} \\ A \\ B \\ C \\ D \end{array}\begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

邻接矩阵

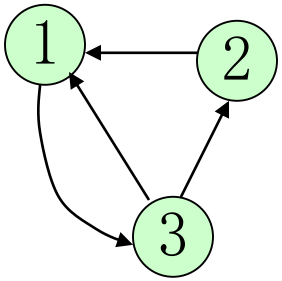
顶点vi的 **TD**(vi)=M中第i**行**元素之和

$$= \sum_{j=1}^{n} M[i][j]$$

顶点vi的 **TD**(vi)=M中第i**列**元素之和

$$= \sum_{j=1}^{n} M[j][i]$$

例3



$$M=\begin{array}{c|ccc} & 1 & 2 & 3 \\ \hline 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \end{array}$$

邻接矩阵

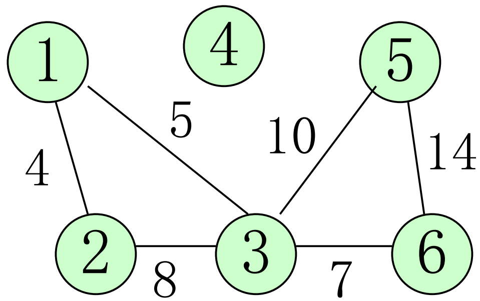
顶点vi的 **ID**(vi)=M中第i**列**元素之和

$$= \sum_{j=1}^{n} M[j][i]$$

顶点vi的 **OD**(vi)=M中第i**行**元素之和

$$= \sum_{j=1}^{n} M[i][j]$$

例4



网G

$$
M=\begin{array}{c c}
 & \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\
\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} &
\left(\begin{array}{cccccc}
\infty & 4 & 5 & \infty & \infty & \infty \\
4 & \infty & 8 & \infty & \infty & \infty \\
5 & 8 & \infty & \infty & 10 & 7 \\
\infty & \infty & \infty & \infty & \infty & \infty \\
\infty & \infty & 10 & \infty & \infty & 14 \\
\infty & \infty & 7 & \infty & 14 & \infty
\end{array}\right)
\end{array}
$$

邻接矩阵

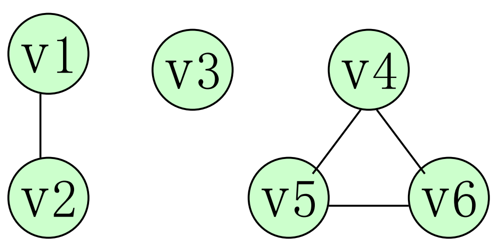
思考题:

1. 如何求每个顶点的度 **D(vi)**?　　$1 \leqslant i \leqslant n$
2. 如何求每个顶点的出度 **OD(vi)**?　　$1 \leqslant i \leqslant n$
3. 如何求每个顶点的入度 **ID(vi)**?　　$1 \leqslant i \leqslant n$

7.2.2 **邻接表、逆邻接表** ： 链式存储结构。

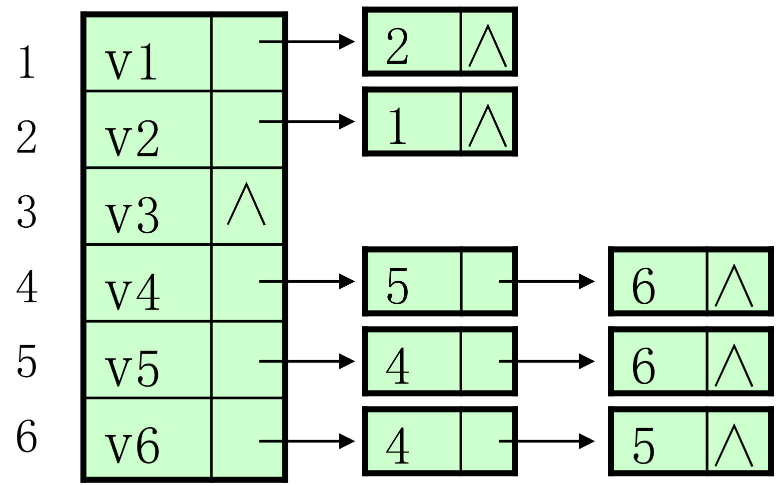
（1） 无向图的**邻接表** ：

为图G的每个顶点建立一个单链表，第i个单链表中的结点表示依附于顶点vi的边。



图G

序号 **头结点**数组 **表结点**单链表

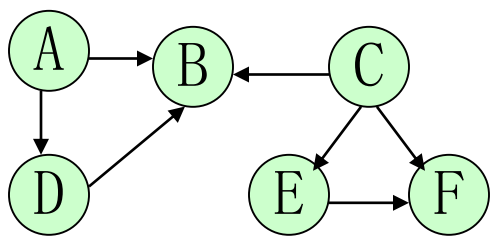


图G邻接表

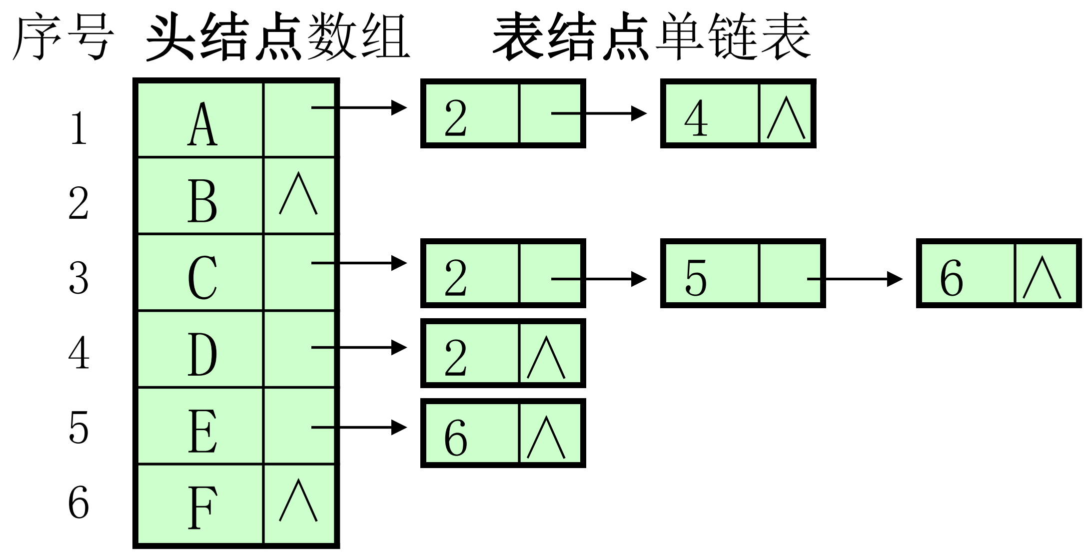
- 若无向图G有n个顶点和e条边，需n个表头结点和2e个表结点。
- 无向图G的邻接表，顶点vi的度为第i个单链表的长度。

（2）有向图的**邻接表** ：

第i个单链表中的表结点，表示以顶点vi为尾的弧(vi,vj)的弧头。



有向图G

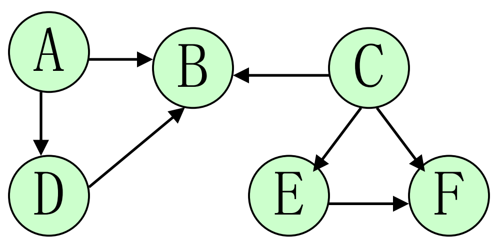


图G的**邻接表**

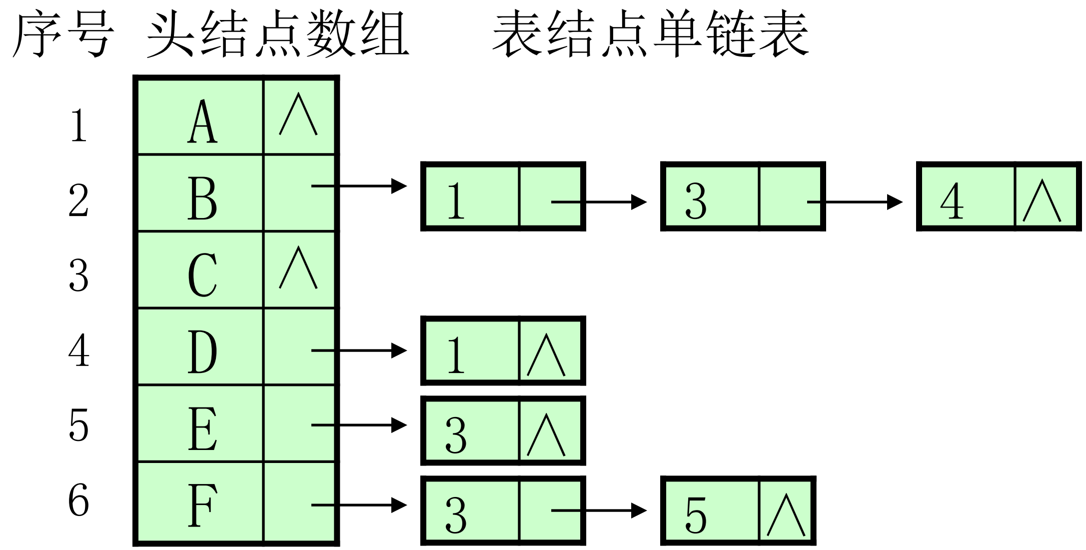
- 若有向图G有n个顶点和e条弧，则需n个表头结点和e个表结点。
- 有向图G的邻接表，顶点vi的**出度**为第i个单链表的长度。
- 求顶点vi的**入度**需遍历全部单链表，统计结点值为i的结点数。

（3）有向图的**逆邻接表**

第i个单链表中的表结点，表示以顶点vi为头的弧(vi,vj)的弧尾。



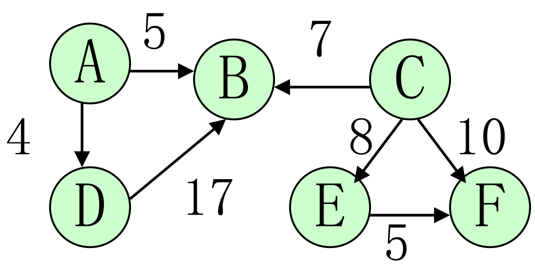
有向图G



图G的**逆邻接表**

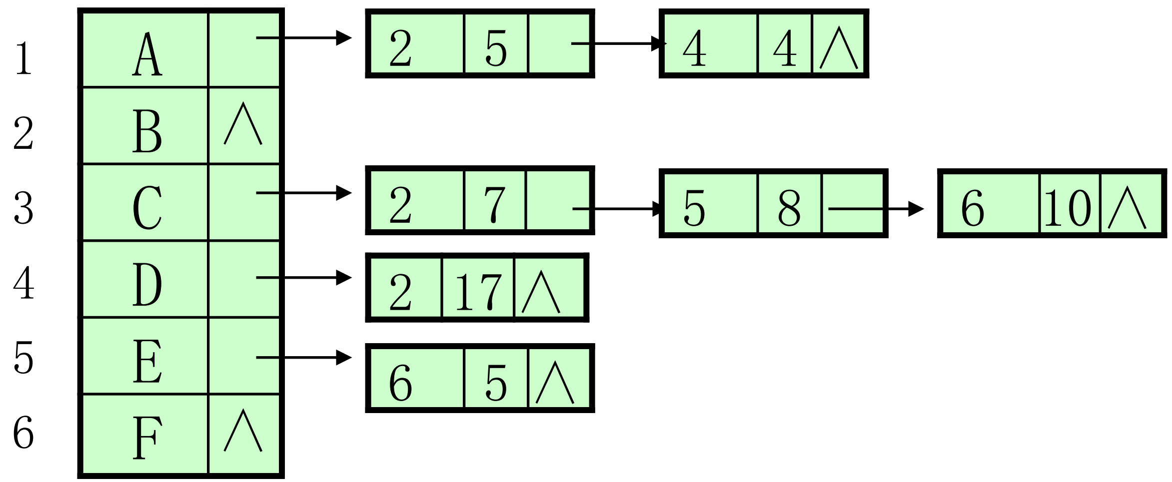
- 若有向图G有n个顶点e条弧，则需n个表头结点和e个表结点。
- 有向图G的逆邻接表，顶点vi的**入度**为第i个单链表的长度。
- 求顶点vi的**出度**需遍历全部单链表，统计结点值为i的结点数。

（4）有向网的**邻接表**



有向网G

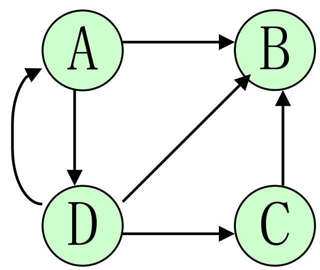
序号 **头结点**数组 **表结点**单链表



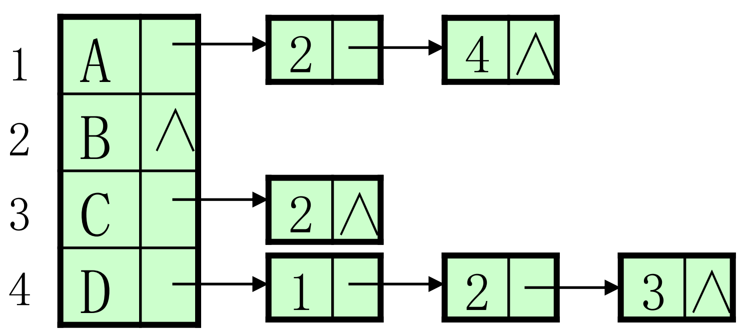
有向网G的**邻接表**

7.2.3 有向图的十字链表

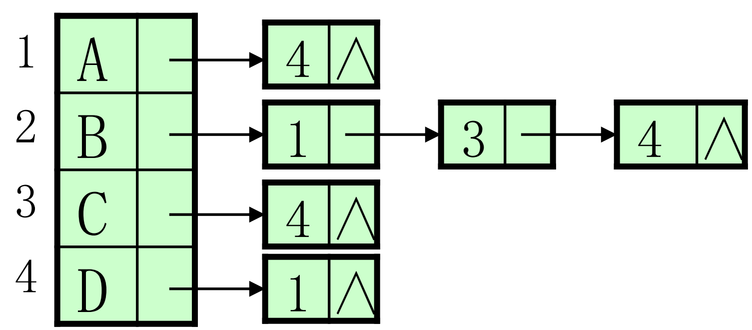
将邻接表和逆邻接表合并而成的链接表。



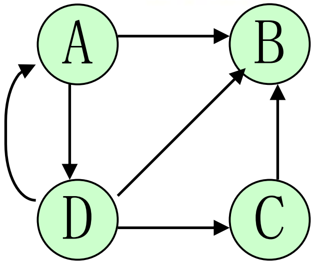
G



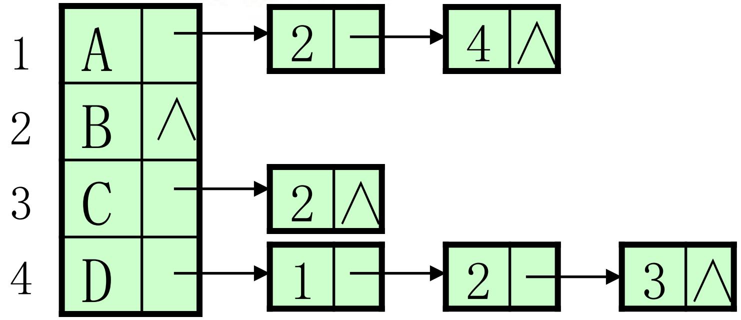
G的邻接表



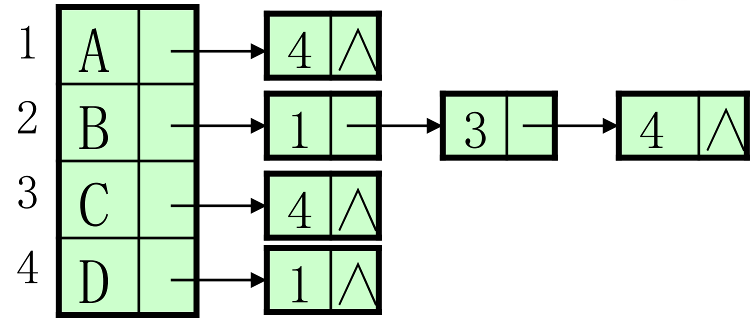
G的逆邻接表



G

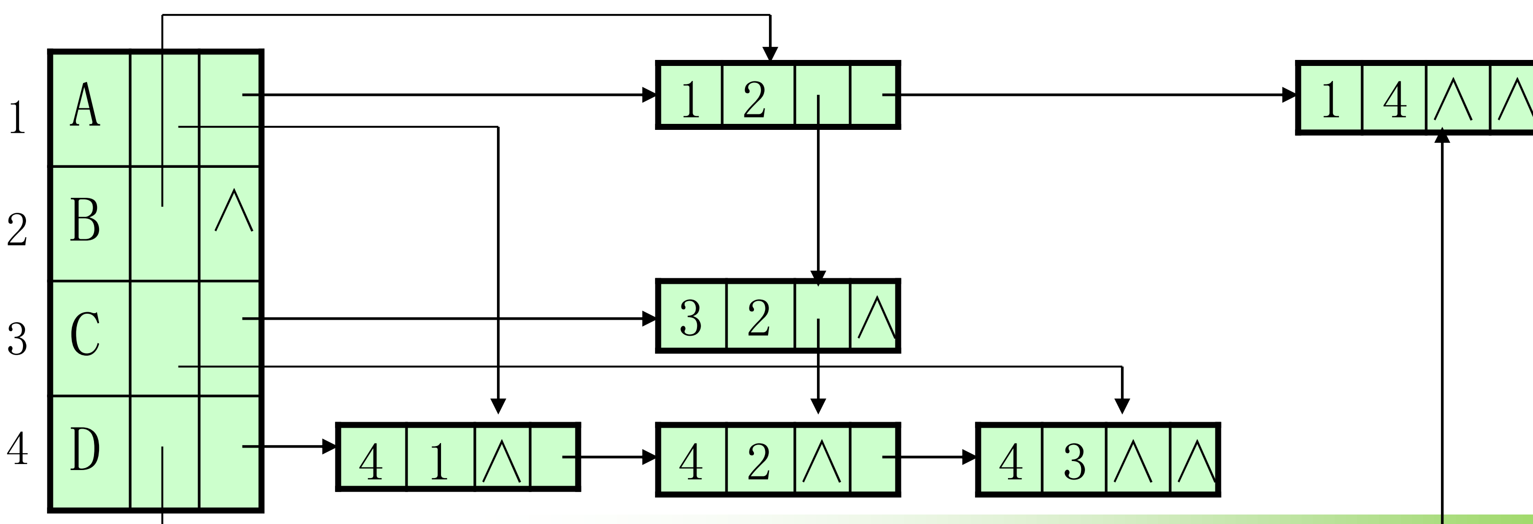


G的**邻接表**



G的**逆邻接表**

- 以邻接表为基础，扩展结点属性成起止结点序号
- 再添加逆邻接表信息



G的十字链表

7.2.4 邻接多重表

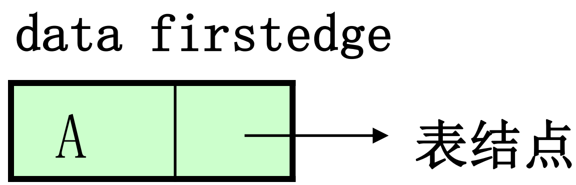
（无向图的）的另一种链式存储结构

（1）图的一个顶点用一个“头结点”表示，

其中：

data域 存储和该顶点相关的信息，

firstedge域 存储第一条依附于该顶点的边。



表示顶点A的**头结点**

（2）图的一条边用一个“**结点**”表示，

其中：

mark----标志域，可用以标记该条边是否被搜索过；

vi和vj----该条边依附的两个顶点在图中的位置；

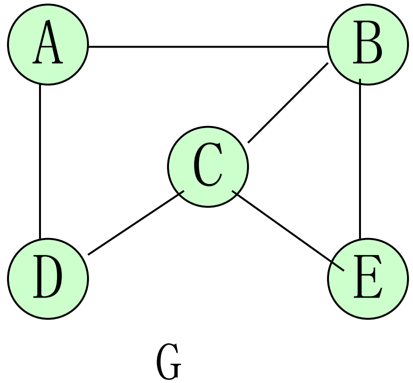
vilink----指向下一条依附于顶点vi的边；

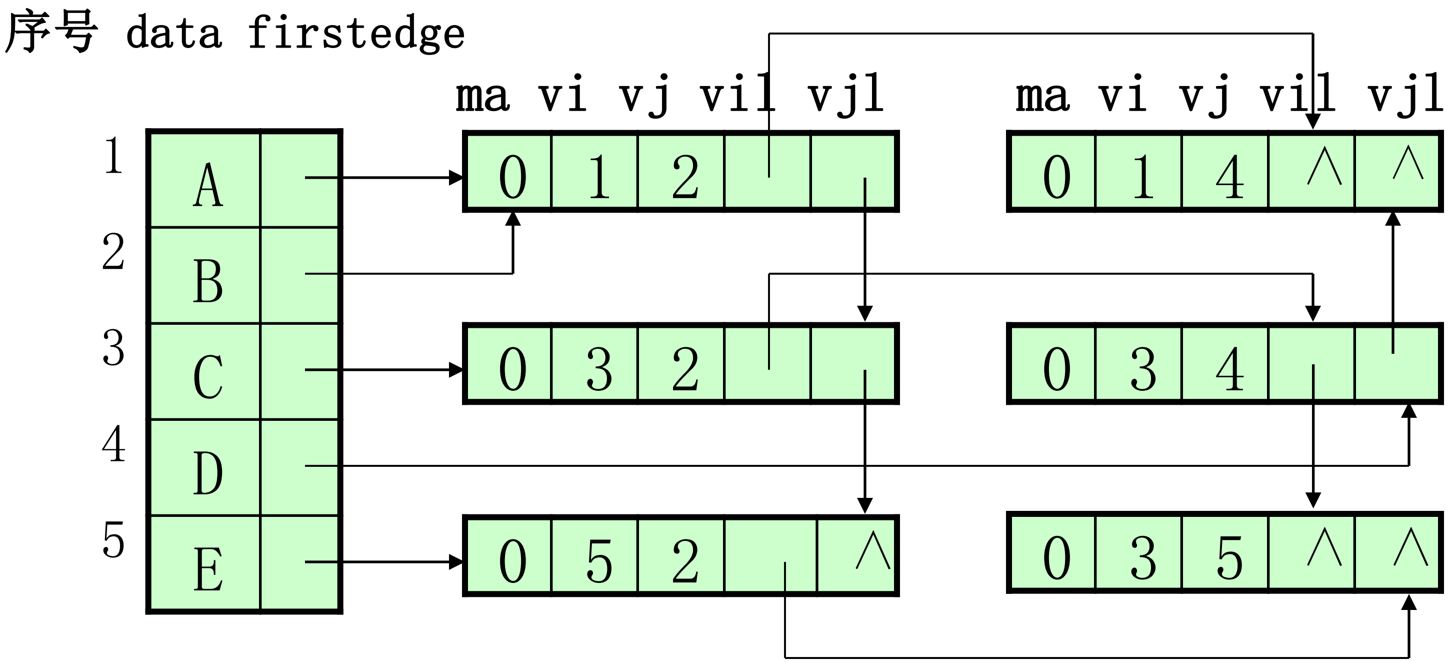
vjlink----指向下一条依附于顶点vj的边。

避免了无向图邻接表的一条边用两个结点。

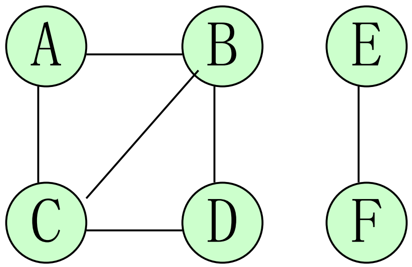
mark	vi	vj	vilink	vjlink
0	1	2		

7.2.4 邻接多重表（续）

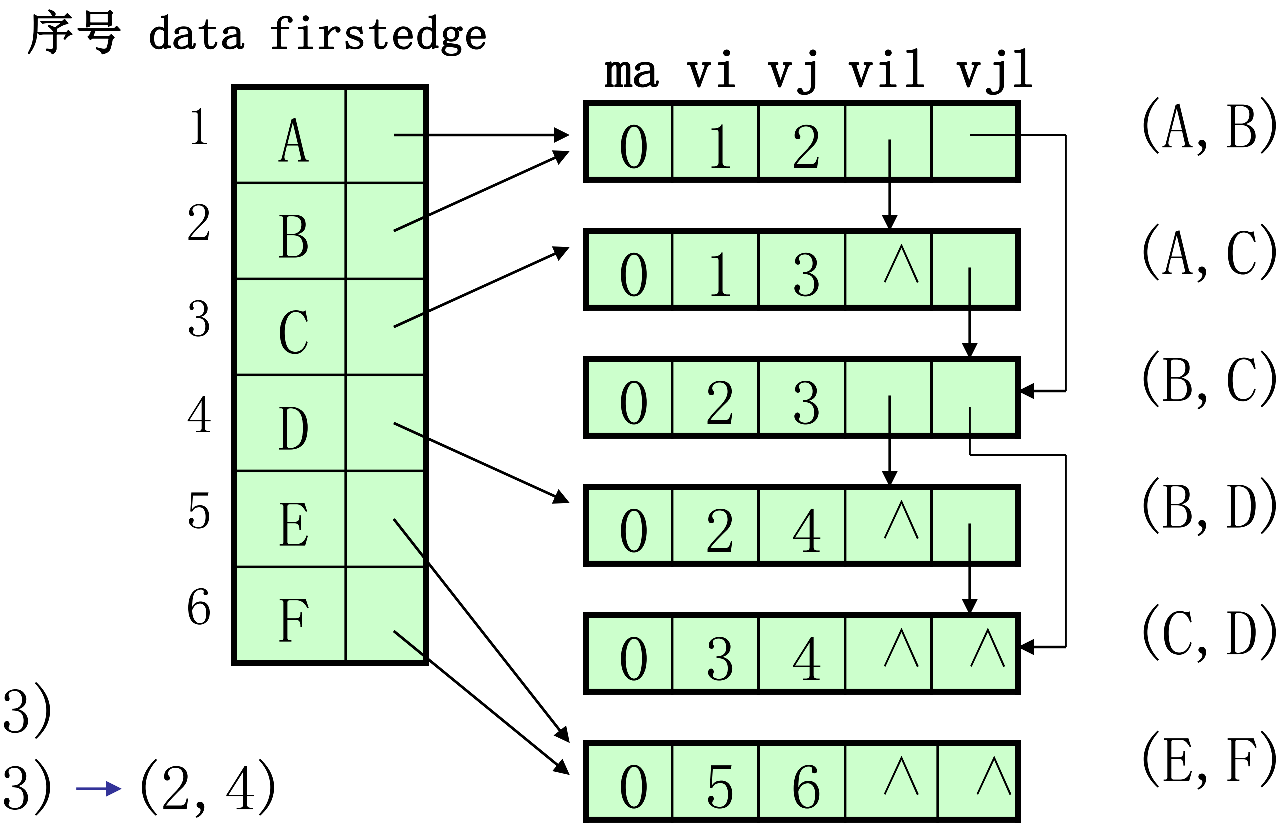

A
B
C
D
E
G


序号 data firstedge
ma vi vj vil vjl
ma vi vj vil vjl
1 A
2 B
3 C
4 D
5 E
0 1 2
0 1 4 ∧ ∧
0 3 2
0 3 4
0 5 2 ∧
0 3 5 ∧ ∧

7.2.4 邻接多重表（续）



G



隐含的链接表：

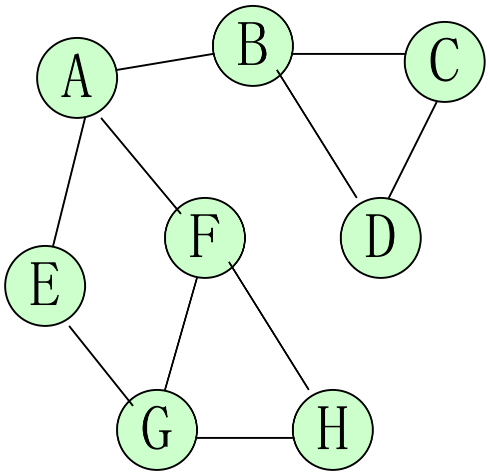
- A → (1, 2) → (1, 3)
- B → (1, 2) → (2, 3) → (2, 4)
- C → (1, 3) → (2, 3) → (3, 4)
- D → (2, 4) → (3, 4)
- E → (5, 6)
- F → (5, 6)

7.3 图的遍历

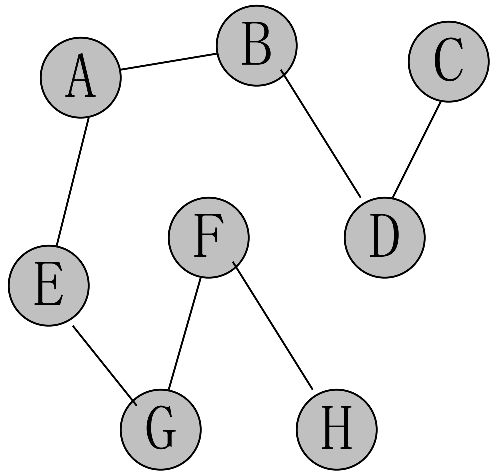
从图G的某定点vi出发，访问G的每个顶点一次且一次的过程。

7.3.1 图的深度优先搜索

DFS---Depth First Search



图G



假定从 A 出发遍历图G：

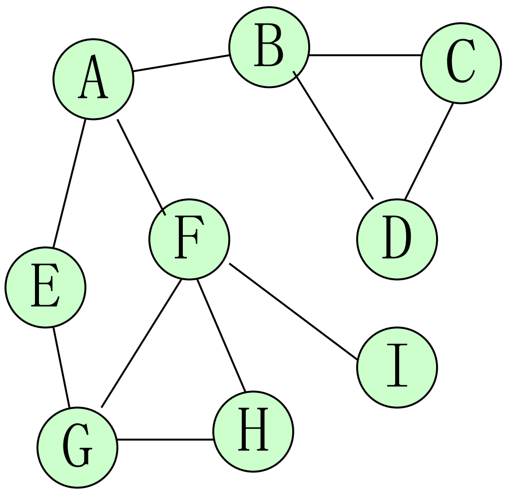
- A, E, G, F, H, B, D, C
- A, B, D, C, E, G, H, F
- A, B, F, G, H, E, C, D
- A, E, F, H, G, B, C, D
- A, F, G, H, E, B, D, C

假定从G出发遍历图G：

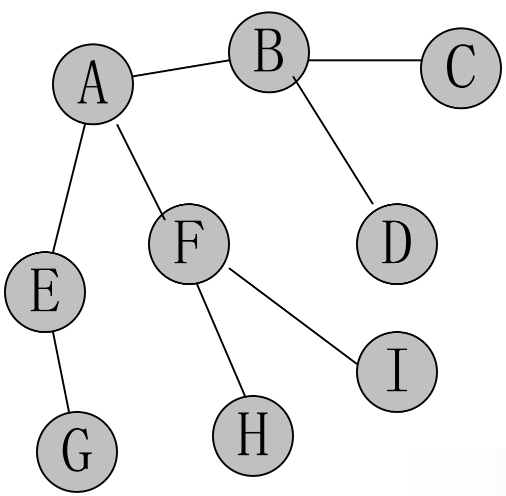
- G, F, A, B, D, C, E, H
- G, H, F, A, E, B, D, C
- G, E, A, H, F, B, C, D

7.3.2 图的广(宽)度优先搜索

BFS---Breadth First Search



图G



假定从 A 出发遍历图G:

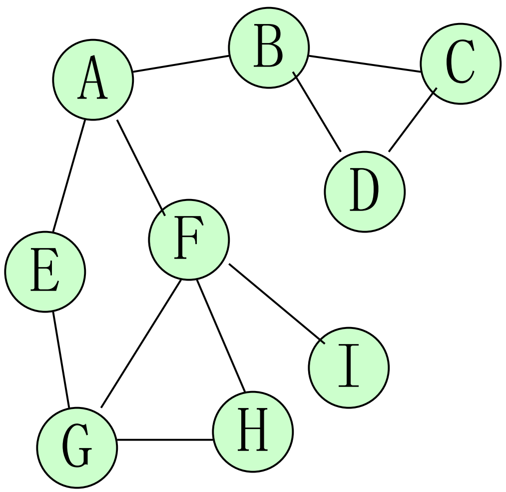
- A, E, F, B, G, H, I, D, C
- A, B, F, E, D, C, I, H, G
- A, F, E, G, H, I, B, D, C ?
- A, E, B, F, I, H, G, D, C ?

假定从 G 出发遍历G:

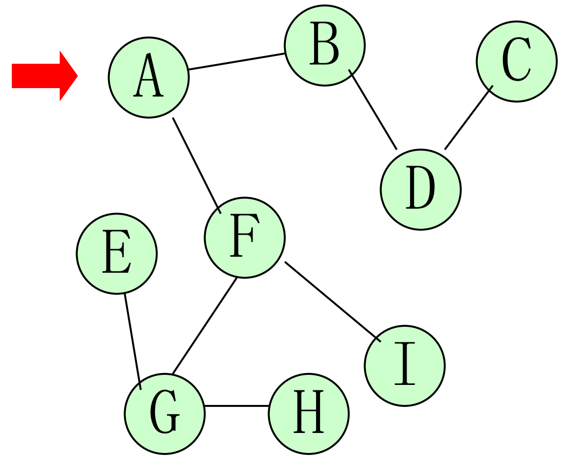
- G, F, E, H, A, I, B, C, D
- G, H, F, E, I, A, B, C, D
- G, E, F, H, I, A, B, C, D ?

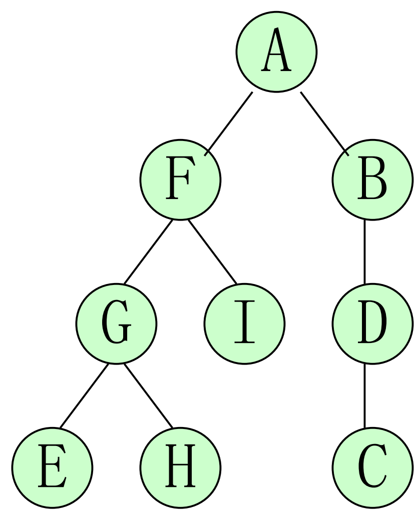
7.4 图的连通性问题

7.4.1 DFS生成树

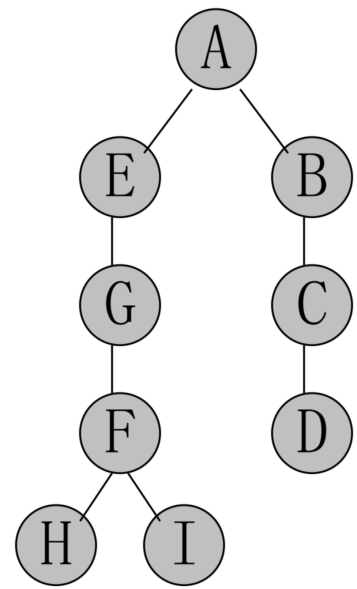


图G





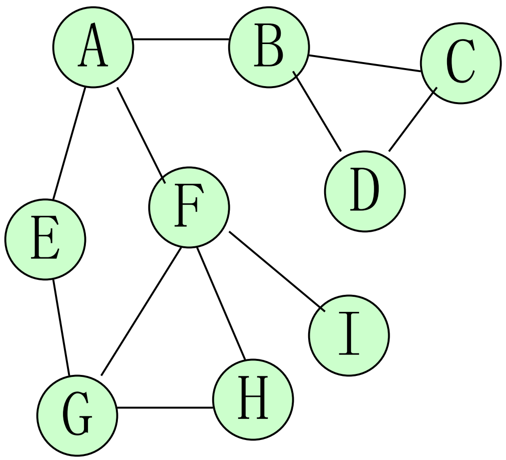
DFS生成树T1



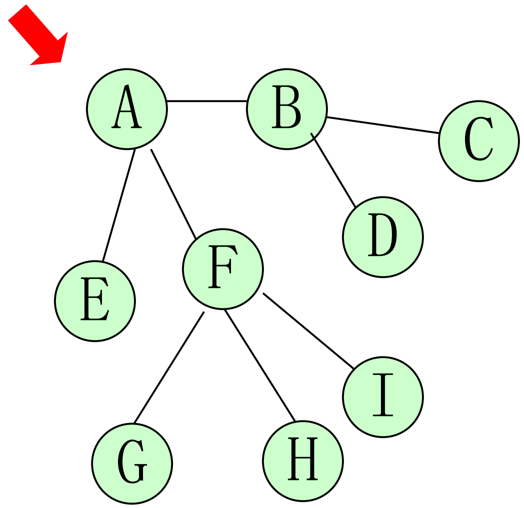
DFS生成树T2

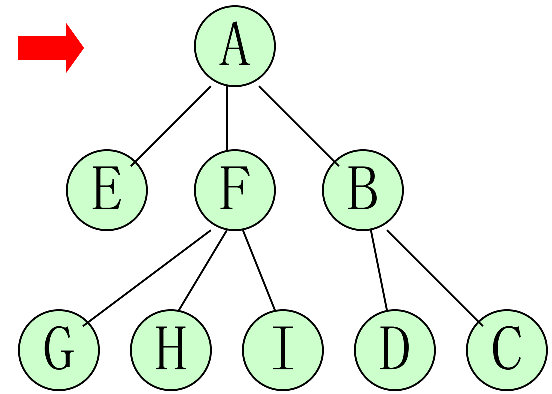
7.4.2 BFS生成树

假定从A出发BFS遍历图G：

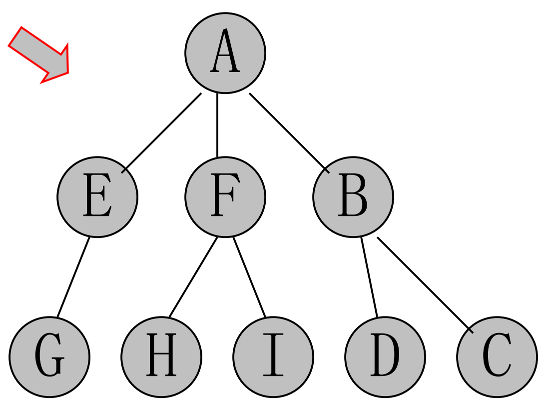


图G



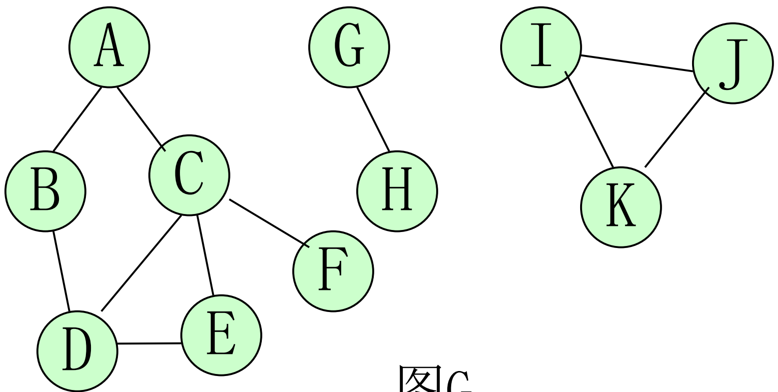


BFS生成树T1



BFS生成树T2

7.4.3 DFS生成森林

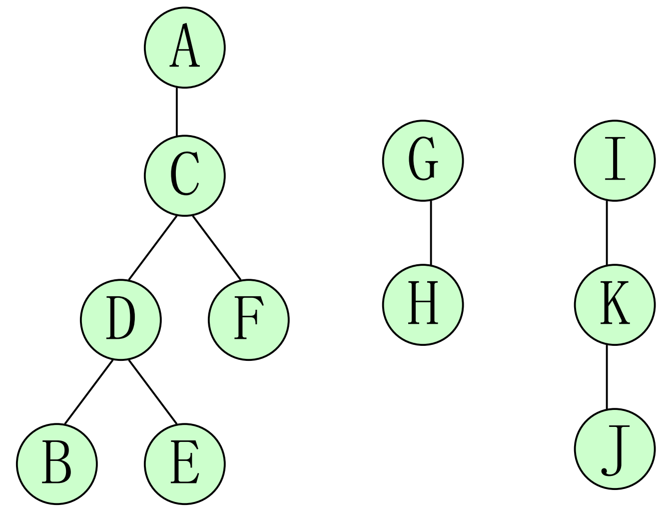


图G

从**A**出发，得树T1：

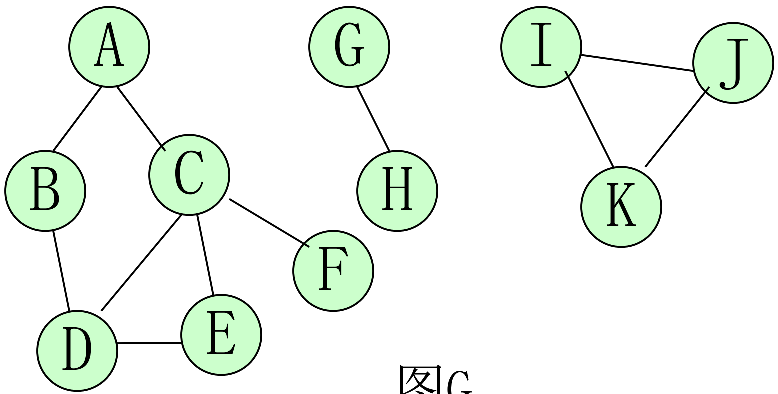
从**G**出发,得树T2：

从**I**出发,得树T3：

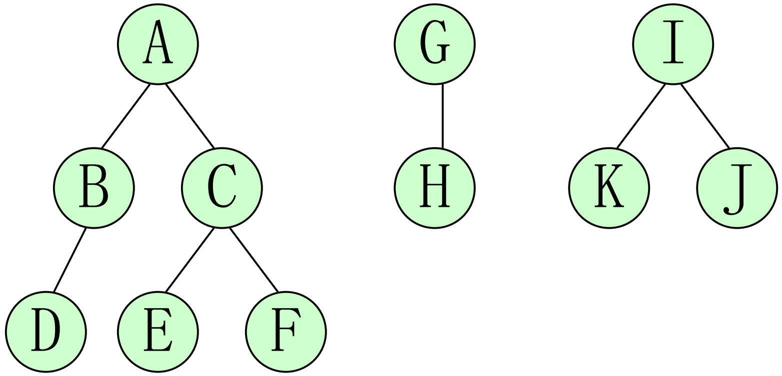


T1　　T2　　T3

7.4.4 BFS生成森林



图G



从A出发，得树T1：

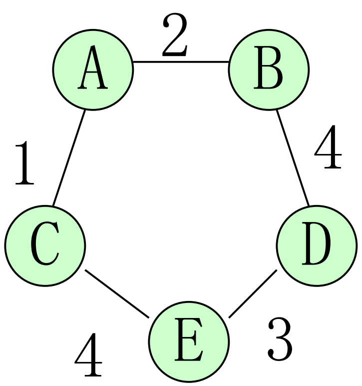
从G出发,得树T2：

从I出发,得树T3：

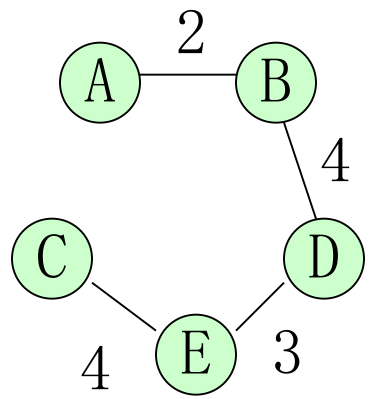
T1 T2 T3

7.4.5 网的最小生成树:

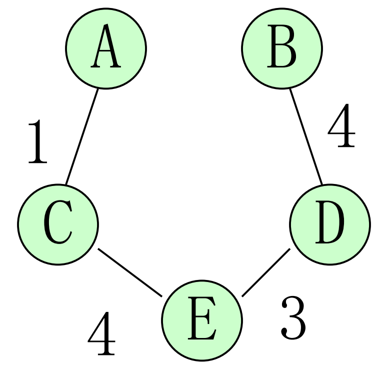
在网G的各生成树中，其中各边的权之和最小的生成树称为G的**最小生成树**



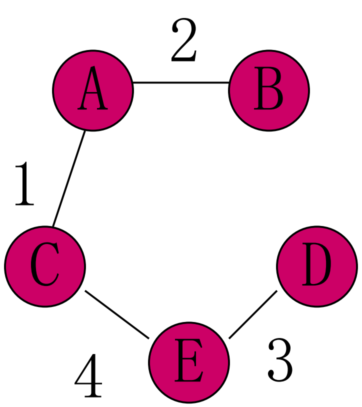
网G



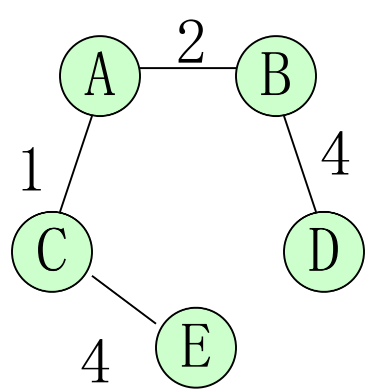
生成树1（13）



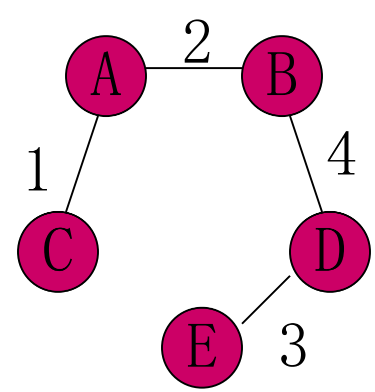
生成树2（12）



生成树3（**10**）

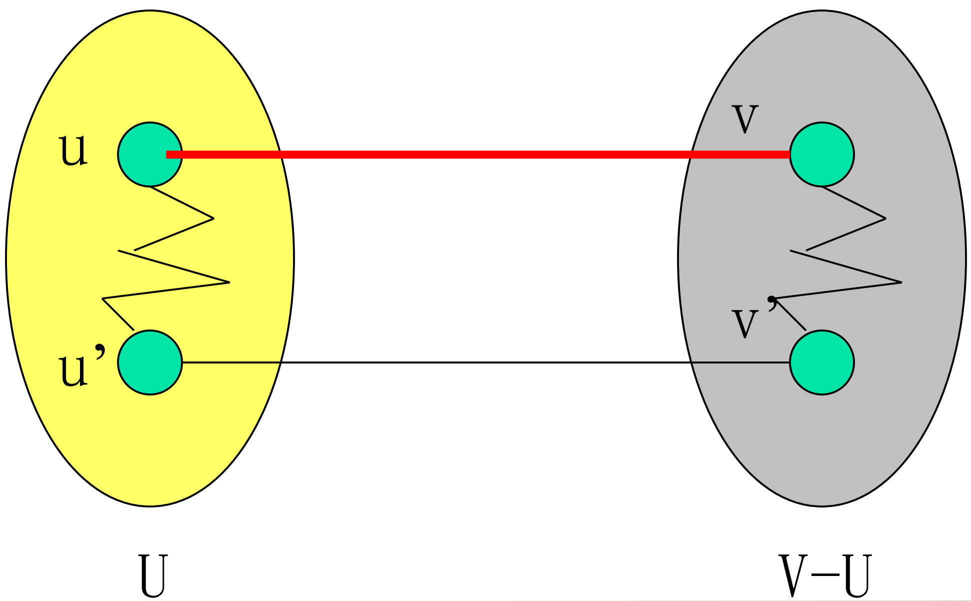


生成树4（11）



生成树5（**10**）

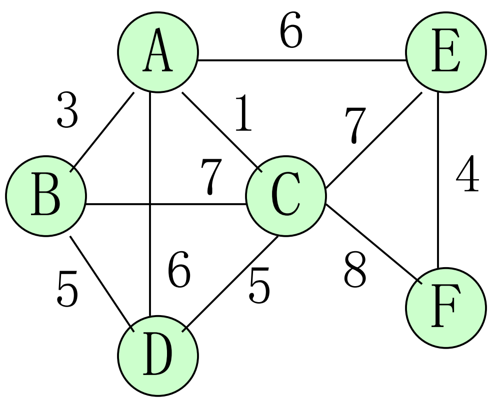
MST性质：设G=（V，E）是一个连通图，通过某种算法构造其最小生成树，T=（U，TE）是正在构造的最小生成树。如果边（u,v)是G中所有一端在U中（即u∈U）而另一端在V-U中（即v∈V-U）具有最小值的一条边，则必存在一棵包含边（u,v)的最小生成树。



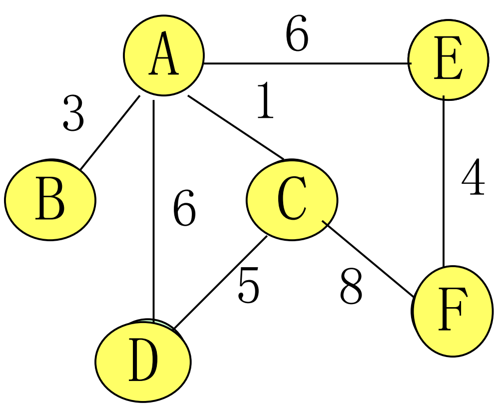
1. **普里姆(prim)**算法　　**以选顶点为主**

　　对n个顶点的连通网，初始时， T=（U，TE），U为一个开始顶点，TE=φ，以后根据MST性质，每次增加一个顶点和一条边，重复n-1次。U不断增大，V－U不断减小直到为空。

例：从A出发

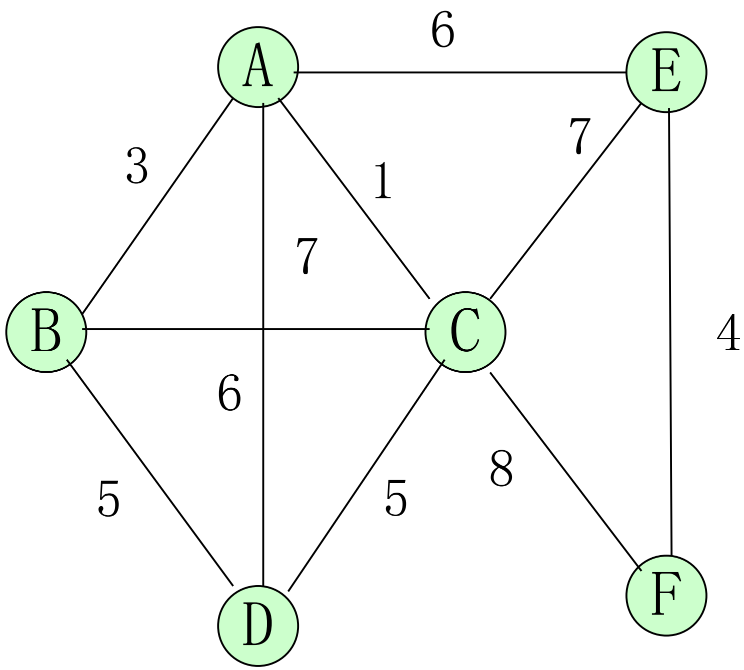


网G

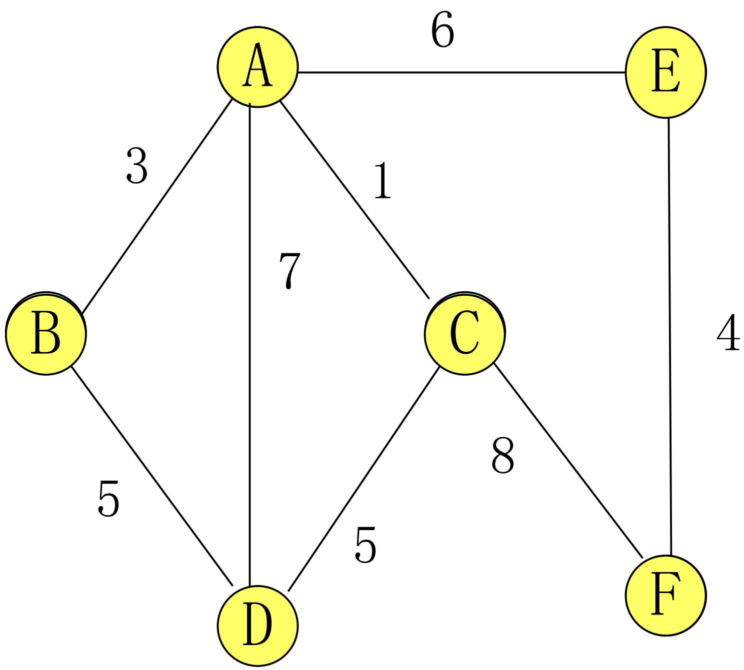


最小生成树T

1. **普里姆(prim)**算法　另一棵最小生成树：从D出发

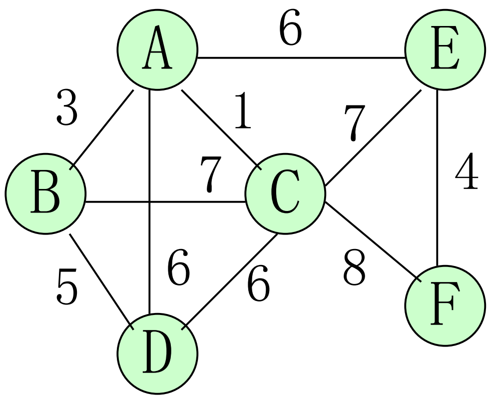


网G



最小生成树T

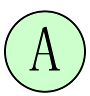
Prim算法思想：



图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0

初始化



V-U中各顶点到U的最短直接路径：

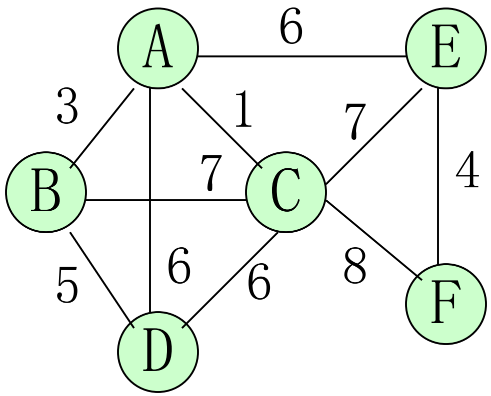
A	B	C	D	E	F
0	3	1	6	6	∞

相邻顶点：

A	B	C	D	E	F
A	A	A	A	A	A

黄色表示U的顶点，其他为V-U的顶点

Prim算法:



图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0

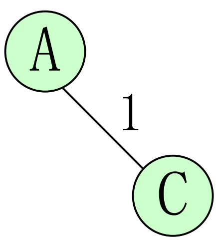
比较大小

V-U中各顶点到顶点集U的最短直接路径:

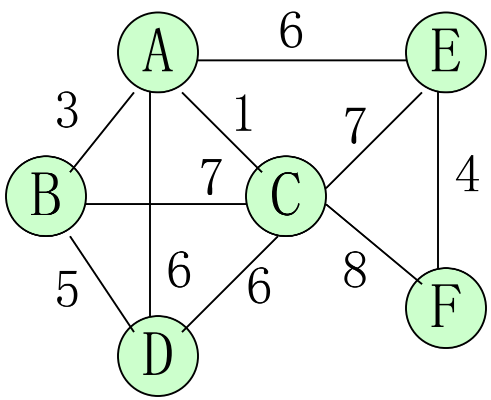
0	3	1	6	6	∞
0	3	1	6	6	8
A	B	C	D	E	F

相邻顶点:

A	A	A	A	A	A
A	A	A	A	A	C



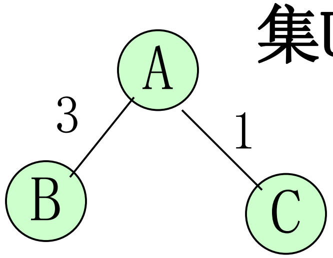
Prim算法:



图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0

比较大小



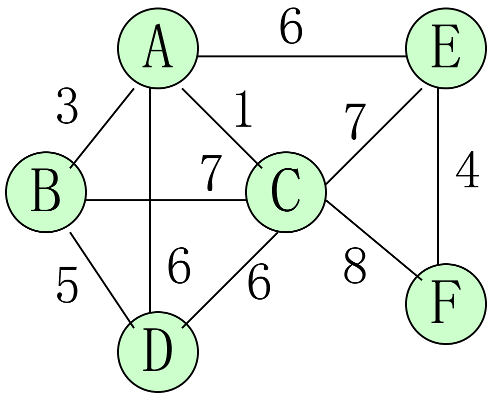
V-U中各顶点到顶点集U的最短直接路径:

0	3	1	6	6	8
0	3	1	5	6	8
A	B	C	D	E	F

相邻顶点:

A	A	A	A	A	C
A	A	A	B	A	C

Prim算法:



图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0

比较大小

V-U中各顶点到顶点集U的最短直接路径:

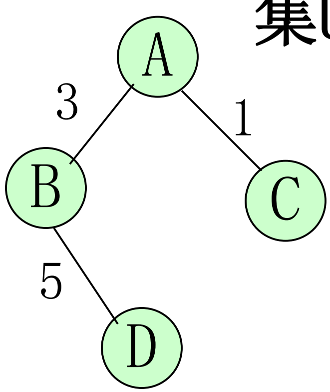
0	3	1	5	6	8

A	B	C	D	E	F
0	3	1	5	6	8

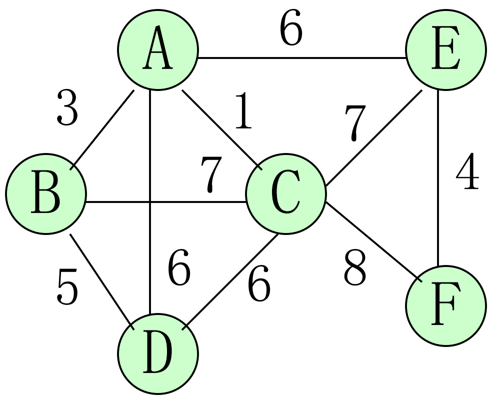
相邻顶点:

A	A	A	B	A	C

A	A	A	B	A	C



Prim算法：



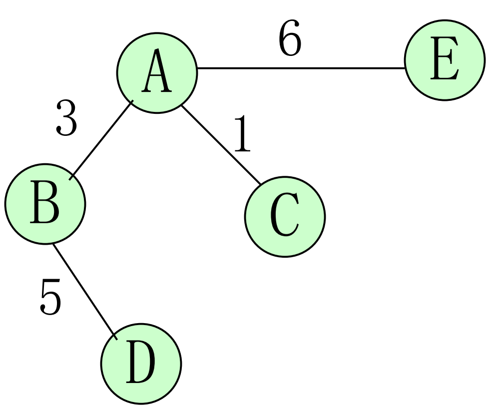
图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	**4**
F	∞	∞	8	∞	4	0

比较大小

V-U中各顶点到顶点集U的最短直接路径：

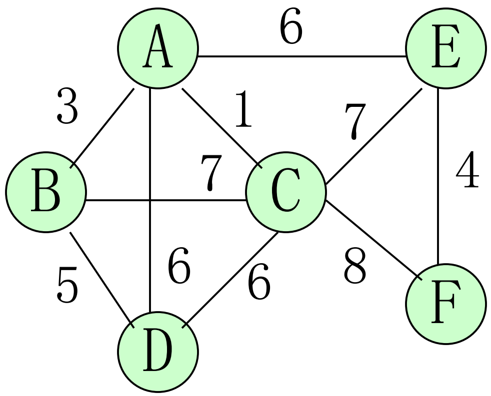
A	B	C	D	E	F
0	3	1	5	6	8
0	3	1	5	6	4



相邻顶点：

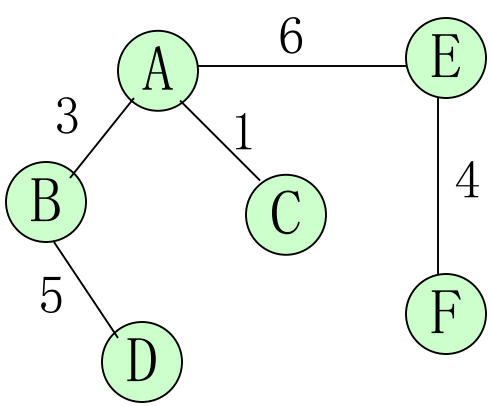
A	B	C	D	E	F
A	A	A	B	A	C
A	A	A	B	A	E

Prim算法:



图G

	A	B	C	D	E	F
A	0	3	1	6	6	∞
B	3	0	7	5	∞	∞
C	1	7	0	6	7	8
D	6	5	6	0	∞	∞
E	6	∞	7	∞	0	4
F	∞	∞	8	∞	4	0



U 和 V-U 最短路径:

0	3	1	5	6	4

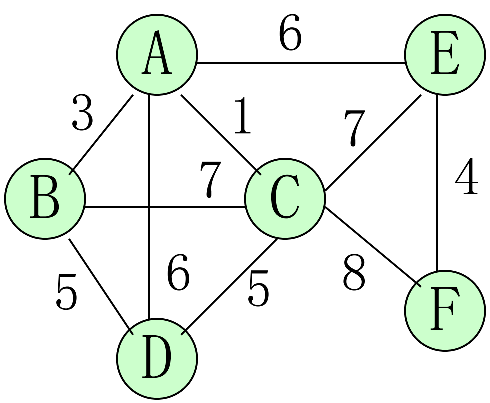
A	B	C	D	E	F
0	3	1	5	6	4

相邻顶点:

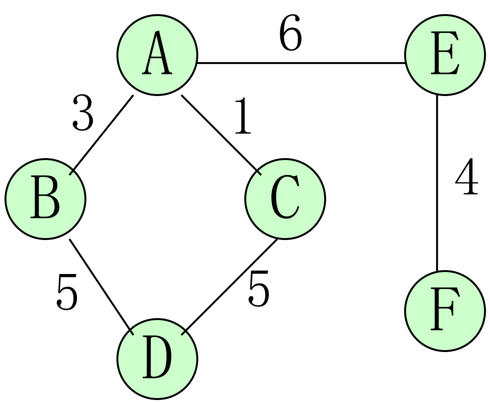
A	A	A	B	A	E
A	A	A	B	A	E

2. **克鲁斯卡尔（Kruskal)算法，以选边为主**

需要将边按递增次序排列以供选择。

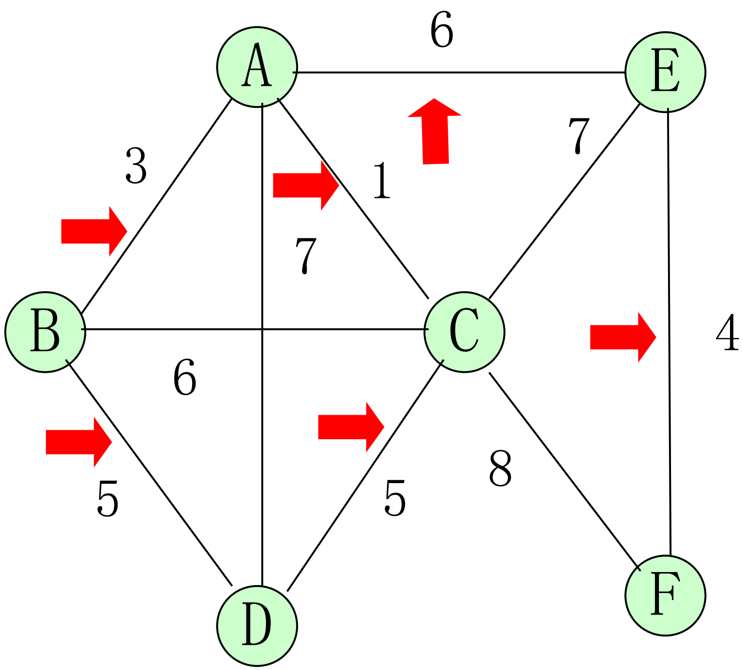


网G

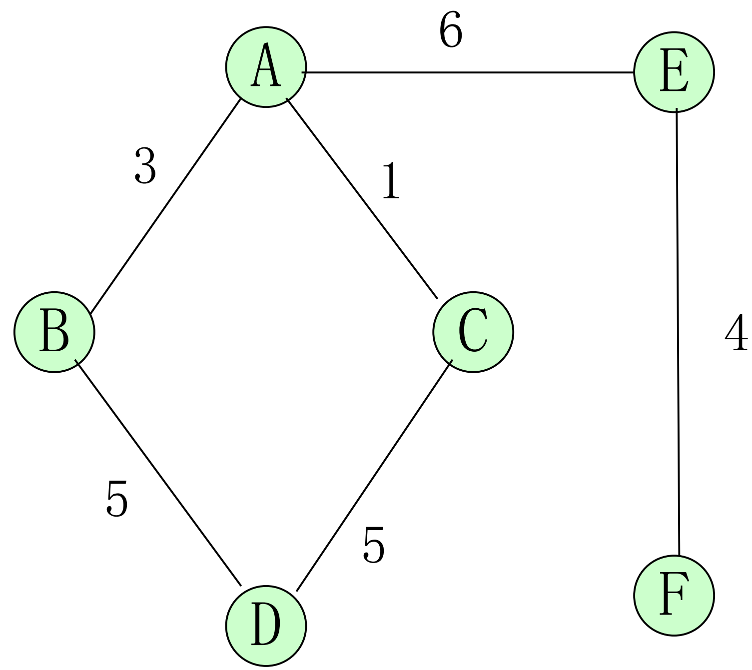


最小生成树T

克鲁斯卡尔（Kruskai)算法的另一最小生成树



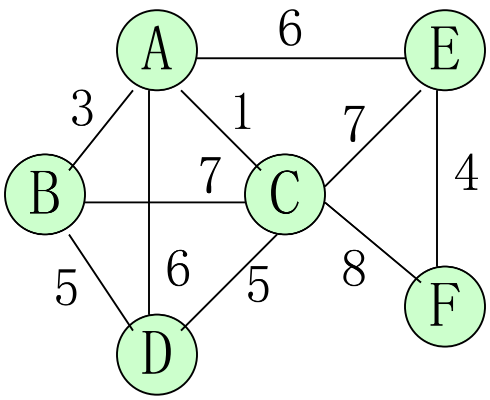
网G



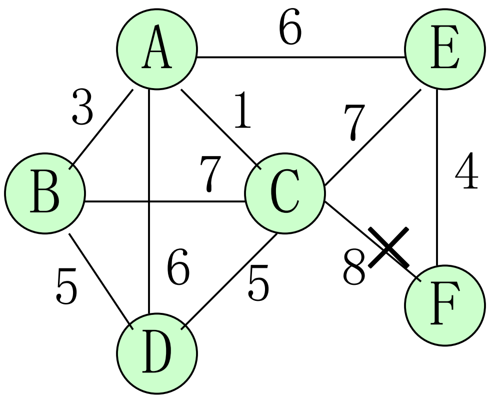
最小生成树T

7.4.5 网的最小生成树

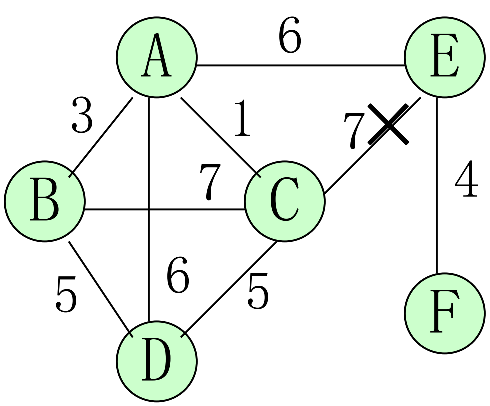
删除权最大的边，以不破坏连通性为标准，保留n-1条



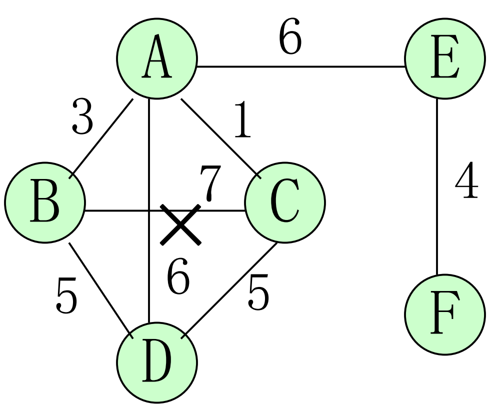
图G



图G’



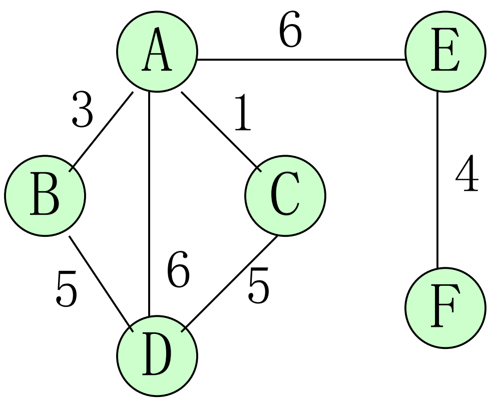
图G’



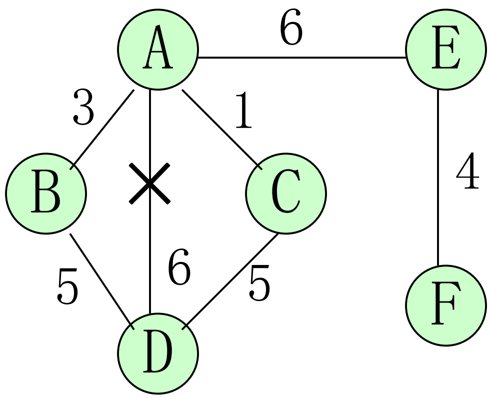
图G’

7.4.5 网的最小生成树

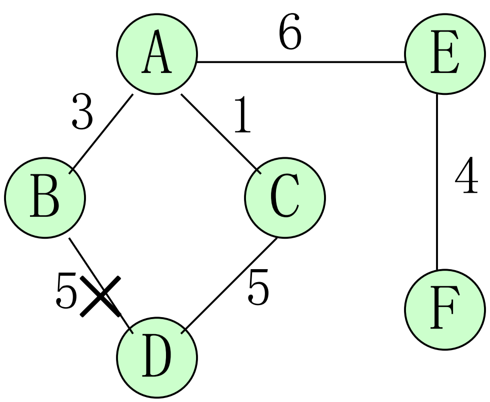
删除权最大的边，以不破坏连通性为标准，保留n-1条



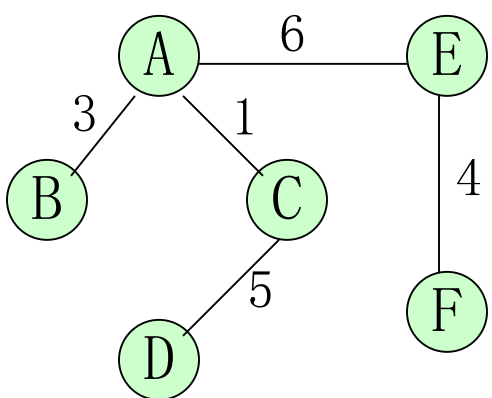
图G



图G’



图G’



T

7.5 有向无环图及其应用

一个无环的有向图称为**有向无环图**（directed acycline graph），简称DAG图。

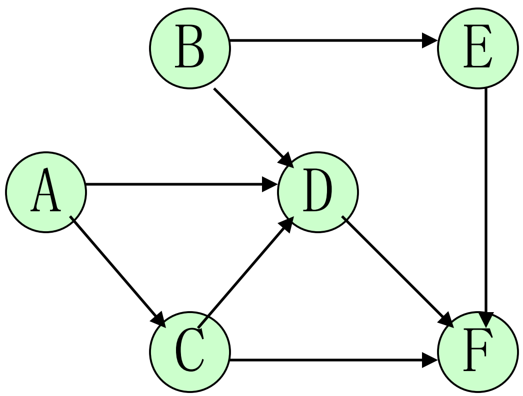


图1

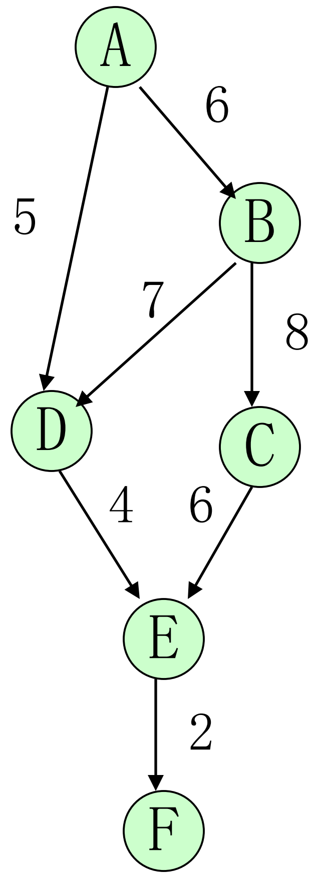


图2

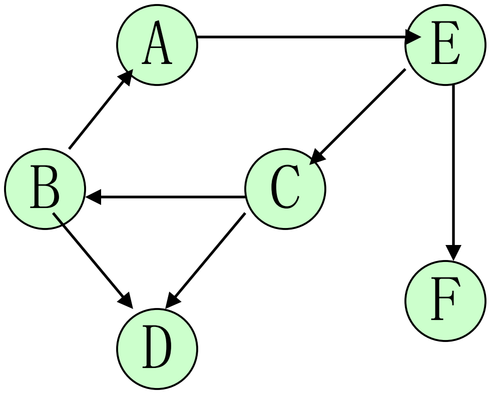


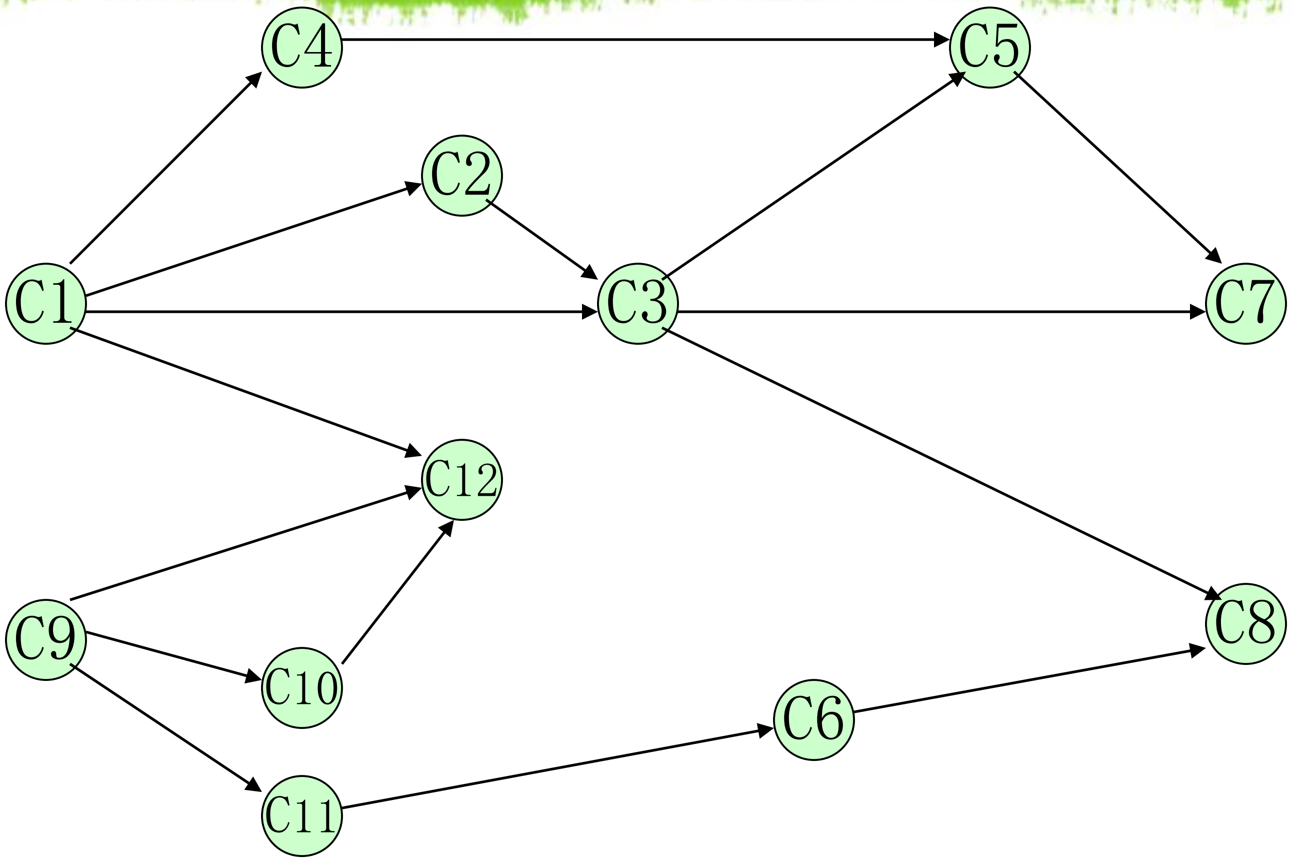
图3（非DAG）

7.5.1 拓扑排序

AOV网（Activity On Vertex network）：
以顶点表示活动，弧表示活动之间的优先关系的DAG图。

计算机软件专业课程

课程编号	课程名称	先决条件
C1	程序设计基础	无
C2	离散数学	C1
C3	数据结构	C1，C2
C4	汇编语言	C1
C5	语言的设计和分析	C3，C4
C6	计算机原理	C11
C7	编译原理	C5，C3
C8	操作系统	C3，C6
C9	高等数学	无
C10	线性代数	C9
C11	普通物理	C9
C12	数值分析	C9，C10，C1



表示课程间关系的有向图

拓扑排序:是有向图的全部顶点的一个线性序列，该序列保持了原有向图中各顶点间的相对次序。例：

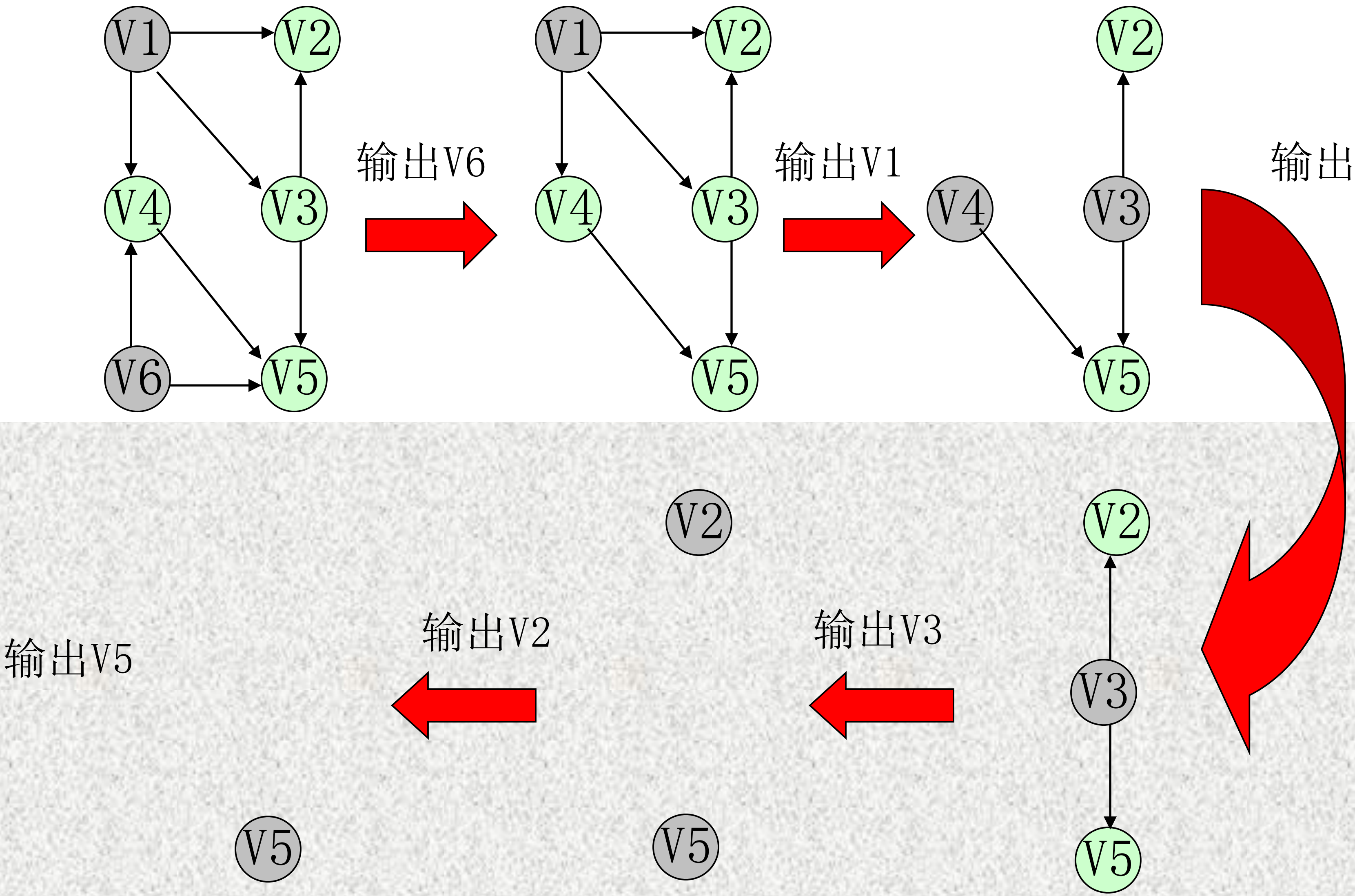
（C1, C2, C3, C4, C5, C7, C9, C10, C11, C6, C12, C8)

（C9, C10, C11, C6, C1, C12, C4, C2, C3, C5, C7, C8)

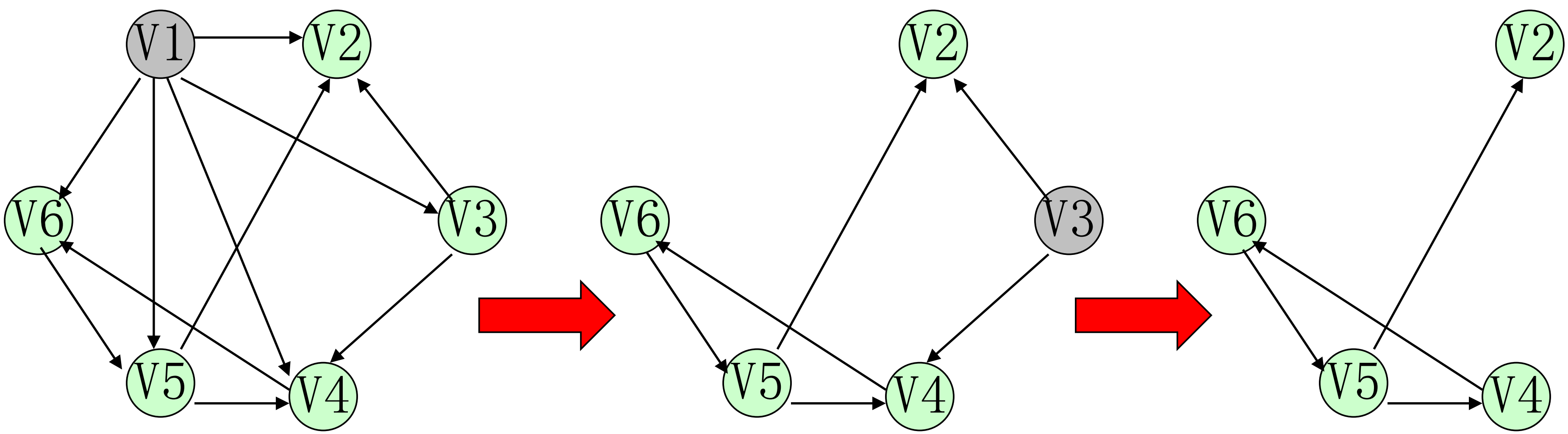
拓扑排序算法思想：重复下列操作，直到所有顶点输出完。

（1）在有向图中选一个没有前驱的顶点输出(选择入度为0的顶点)；

（2）从图中删除该顶点和所有以它为尾的弧(修改其它顶点入度）。



有回路的有向图不存在拓扑排序。



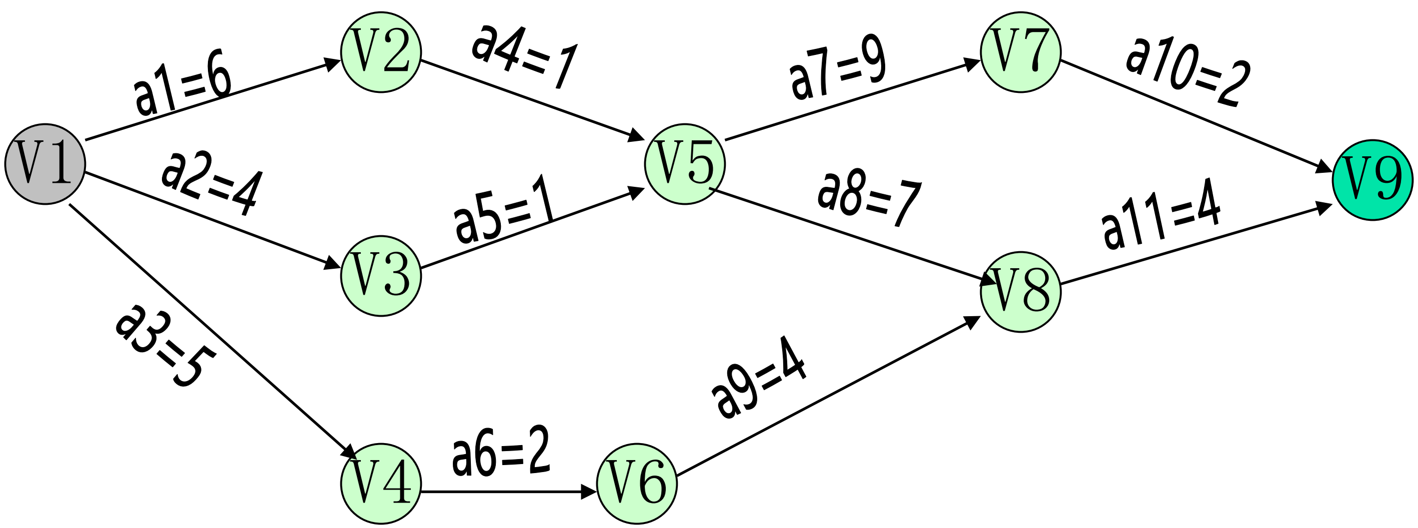
输出V1　　　　输出V3　　　　不能输出

7.5.2 关键路径

AOE网（Activity On Edge）：

是一个带权的有向无环图，其中以顶点表示事件，弧表示活动，权表示活动持续的时间。

当AOE网用来估算工程的完成时间时，只有一个开始点（入度为0,称为**源点**）和一个完成点（出度为0，称为**汇点**）



AOE网研究的问题：

（1） 完成整项工程至少需要多少时间；

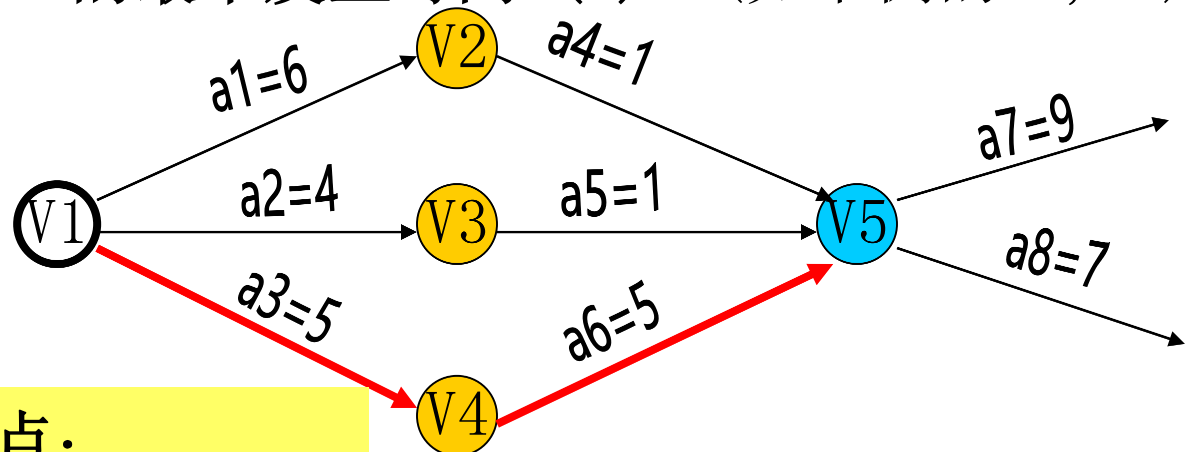
（2） 哪些活动是影响工程进度的关键。

在AOE网中，部分活动可并行进行，所以完成工程的最短时间是从开始点到完成点的最长路径长度。路径长度最长的路径称为**关键路径**（Critical Path）。

（顶点）事件vi的最早发生时间ve(vi)：

从开始点到vi的最长路径长度。（ve(v1)=0）

既表示事件vi的最早发生时间，也表示所有以vi为尾的弧所表示的**活动ak的最早发生时间e(k)**。（如下例的a7，a8）



仅有一个前驱顶点：

ve(v2)=ve(v1)+6=0+6=6

ve(v3)=ve(v1)+4=0+6=4

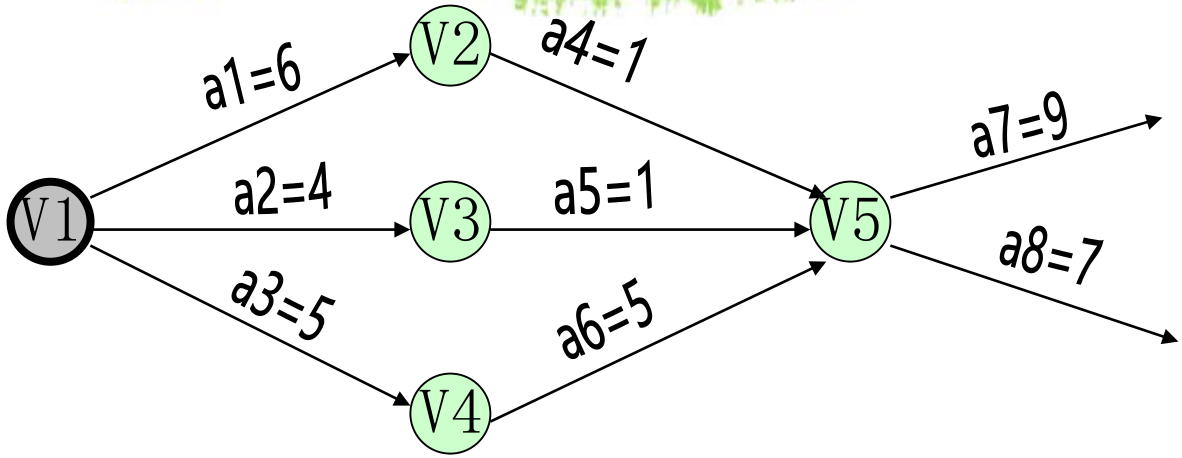
ve(v4)=ve(v1)+6=0+5=5

有多个前驱顶点：

ve(v5)=max{ve(前驱顶点)+前驱活动时间}

=max{6+1，4+1，5+5}=10

完成点（汇点）的ve(vn)为工程完成所需要的时间。



各顶点事件最早开始时间：

ve(v1)=0

ve(v2)=6

ve(v3)=4

ve(v4)=5

ve(v5)=10

各活动最早开始时间：

e(a1)= e(a2)=e(a3)=ve(v1)=0

e(a4)=ve(v2)=6

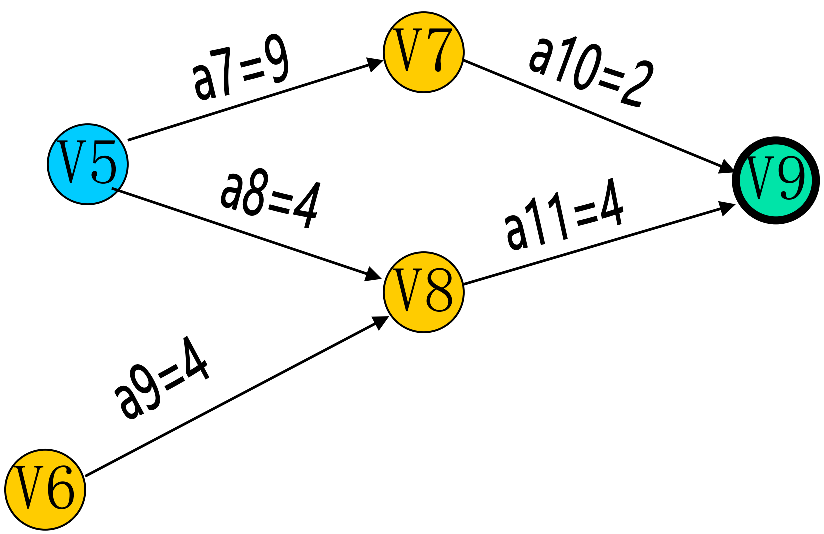
e(a5)=ve(v3)=4

e(a6)=ve(v4)=5

e(a7)=e(a8)=ve(v5)=10

不推迟整个工程完成的前提下，**（顶点）事件vi允许的最迟开始时间vl(vi)** = 完成点（汇点）v_n的**的最早发生时间ve(vn)**减去vi到vn的最长路径长度。

（v_n的**的最早发生时间ve(vn)**等于**最迟开始时间vl(vn)**）。



仅有一个后继顶点：

假定工程18天完成(ve(v9)=18),则：

vl(v9)=18

vl(v7)= vl(v9)-2=16

vl(v8)= vl(v9)-4=14

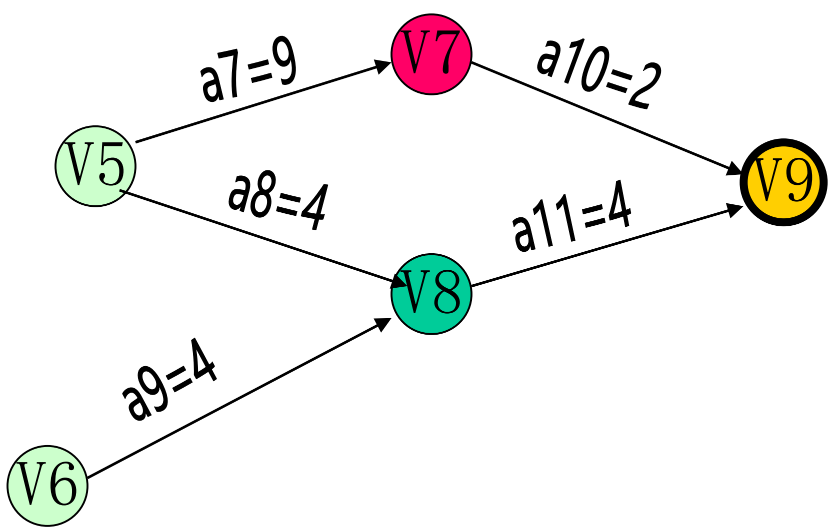
vl(v6)= vl(v9)-8=10

有多个后继顶点：

vl(v5)= min{vl(v7)-9, vl(v8)-4}=min{7,10}=7

确定了顶点vi的最迟开始时间后，确定所有以vi为弧头的**活动ak的最迟开始时间l(k)**：表示在不推迟整个工程完成的前提下，活动ak最迟必须开始的时间。

l(ak)=vl(ak弧头对应顶点)-活动ak的持续时间



vl(v7)= vl(v9)-2=16

vl(v8)= vl(v9)-4=14

vl(v9)=18

l(a11)=vl(v9)-4=18-4=14

l(a10)=vl(v9)-2=18-2=16

l(a9)=vl(v8)-4=14-4=10

l(a8)=vl(v8)-4=14-4=10

l(a7)=vl(v7)-9=**16**-9=7

l(i)-e(i)意味着完成活动ai的**时间余量**。

关键活动： l(i)=e(i)的活动。

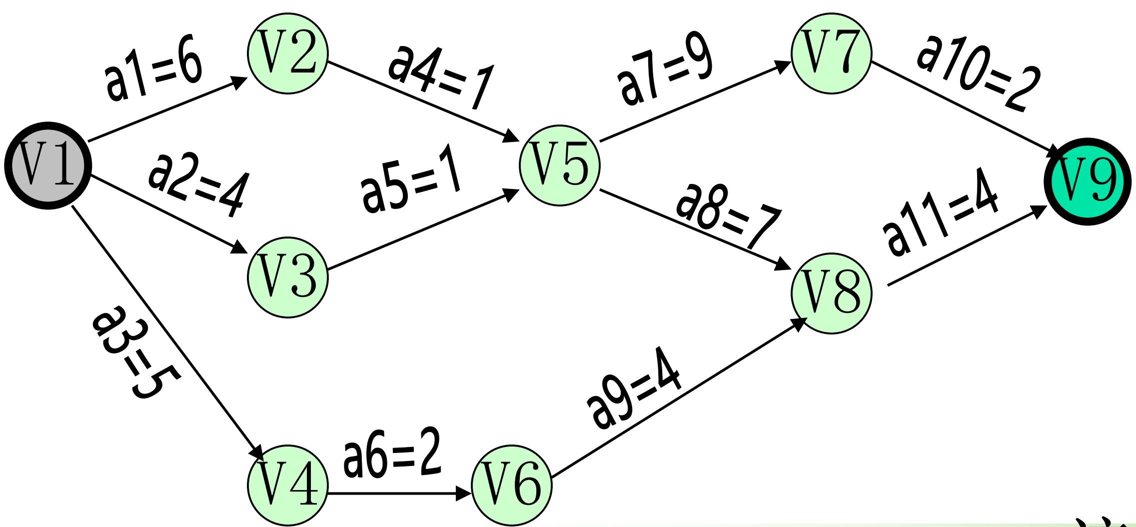
关键路径算法步骤：

（1）从开始点v1出发，令ve(1)=0,按拓朴排序序列求其它各顶点的最早发生时间

$ve(k)=max\{ve(j)+dut(\langle j,k\rangle)\}$

(vj为以顶点vk为弧头的所有弧的弧尾对应的顶点集合)

顶点	ve(i)	vl(i)
v_1	0	
v_2	6	
v_3	4	
v_4	5	
v_5	**7,** 5	
v_6	7	
v_7	16	
v_8	**14,** 11	
v_9	**18,18**	



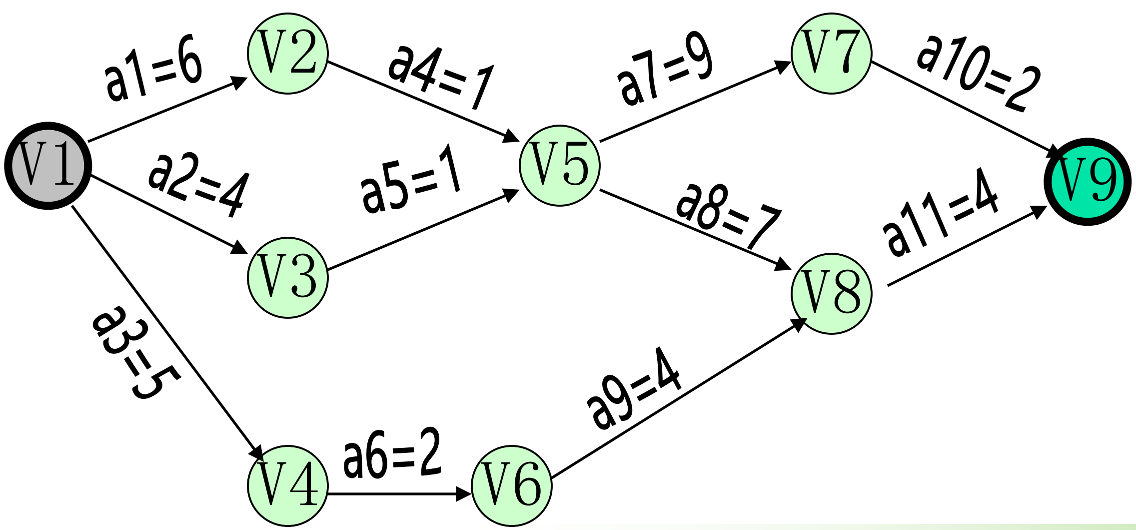
该表次序为一拓扑排序序列

关键路径算法步骤：

（2）从完成点v_n出发，令vl(n)=ve(n),按逆拓朴排序序列求其它各顶点的最迟发生时间

vl(j)=min{vl(k)-dut(<j,k>)}

(vk为以顶点vj为弧尾的所有弧的弧头对应的顶点集合)



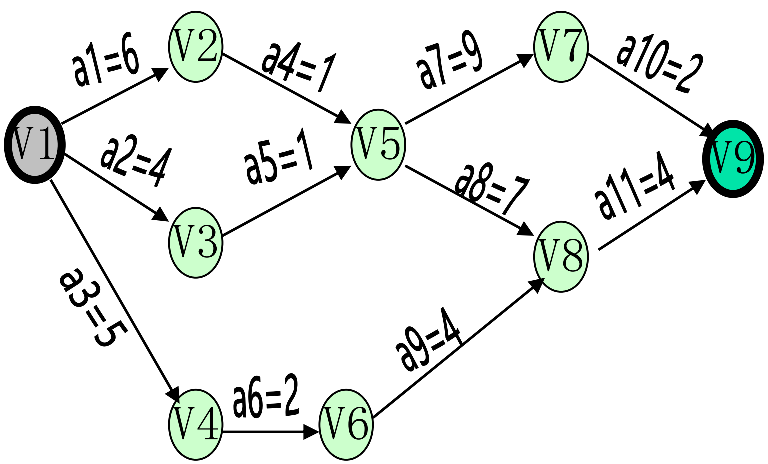
顶点	ve(i)	vl(i)
v_1	0	0，2，3
v_2	6	6
v_3	4	6
v_4	5	8
v_5	7	7，7
v_6	7	10
v_7	16	16
v_8	14	14
v_9	18	18

关键路径算法步骤：

（3）求每一项活动ai(vj, vk)：

e(i)=ve(vj)

l(i)=vl(vk)-dut(ai)



顶点	ve(i)	vl(i)
v_1	0	0
v_2	6	6
v_3	4	6
v_4	5	8
v_5	7	7
v_6	7	10
v_7	16	16
v_8	14	14
v_9	18	18

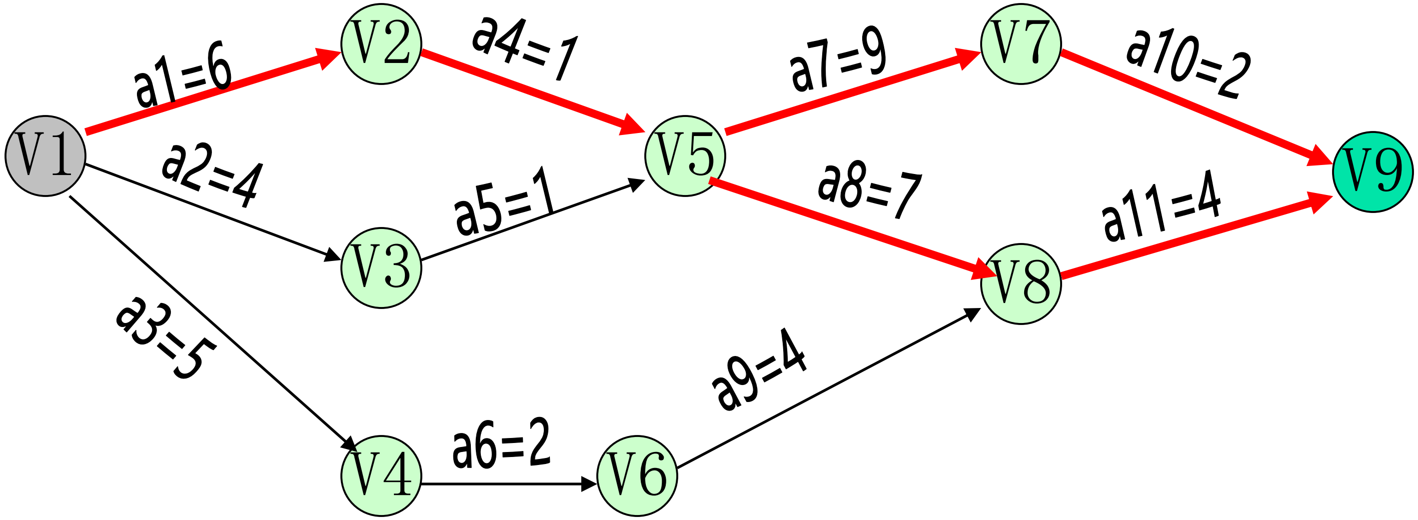
活动	e(i)	l(i)	l(i)-e(i)
a_1	0	0	0
a_2	0	2	2
a_3	0	3	3
a_4	6	6	0
a_5	4	6	2
a_6	5	8	3
a_7	7	7	0
a_8	7	7	0
a_9	7	10	3
a_{10}	16	16	0
a_{11}	14	14	0

关键活动： 选取e(i)=l(i)的活动。

关键路径：

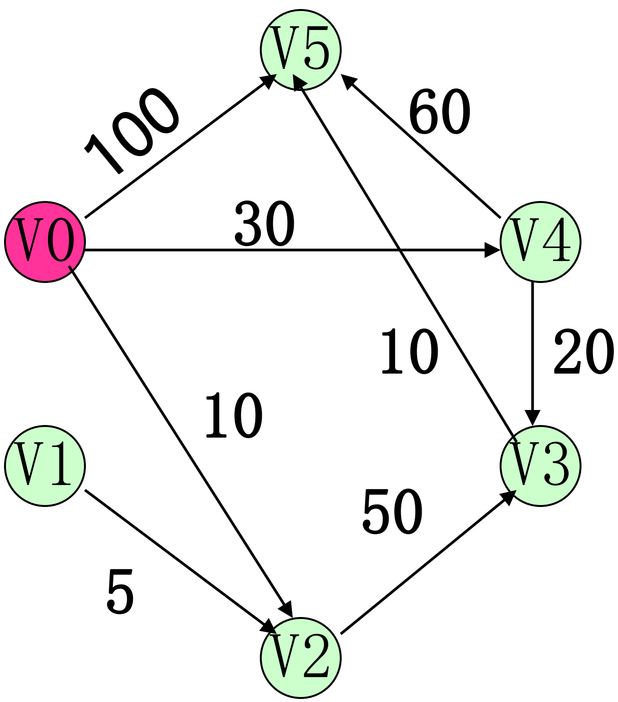
（1） v1→ v2 → v5 → v7 → v9

（2） v1→ v2 → v5 → v8 → v9



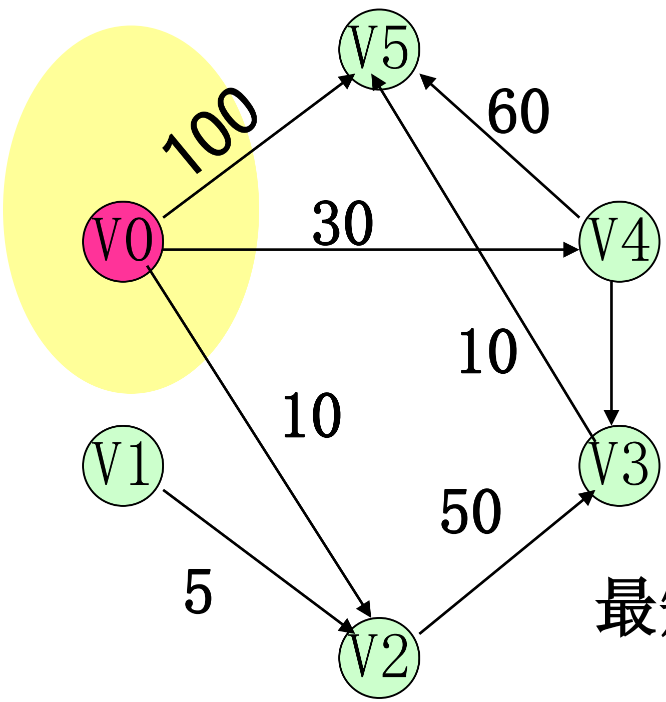
7.6 最短路径

7.6.1　从某个源点到其余各顶点的最短路径



始点	终点	最短路径	路径长度
v0	v1	无	
	v2	v0,v2	10
	v3	v0,v4,v3	50
	v4	v0,v4	30
	v5	v0,v4,v3,v5	60

Dijkstra的路径长度递增次序产生最短路径法：



	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

初始化

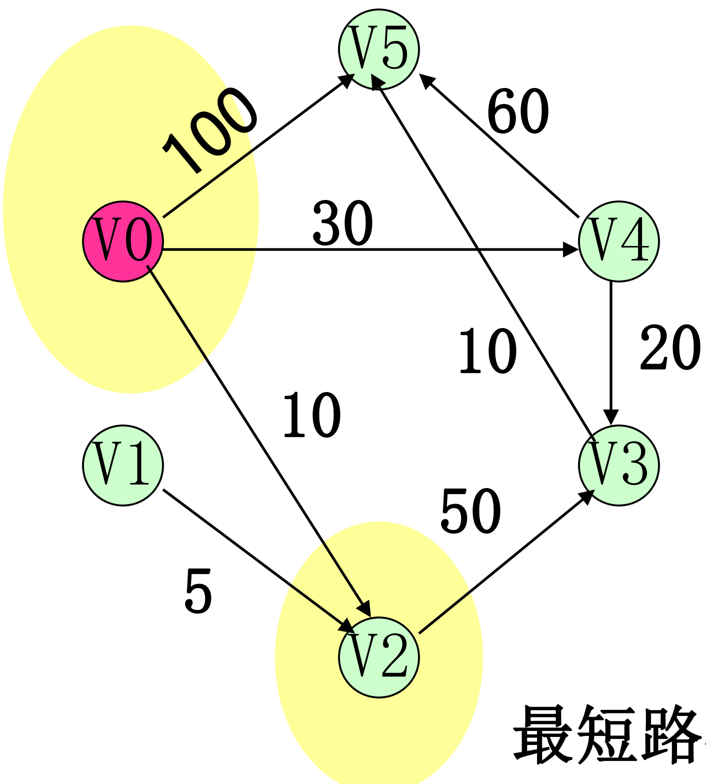
最短路径：

∞	10	∞	30	100

1	2	3	4	5

前驱顶点：

V0	V0	V0	V0	V0



	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

+10

比较大小

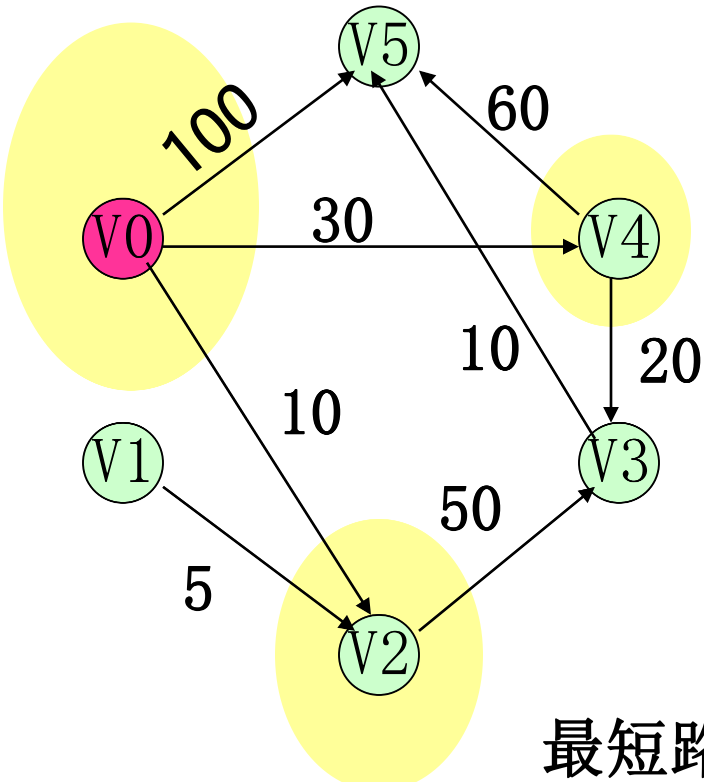
∞	10	∞	30	100

最短路径：

∞	10	60	30	100

前驱顶点：

1	2	3	4	5
V0	V0	V2	V0	V0



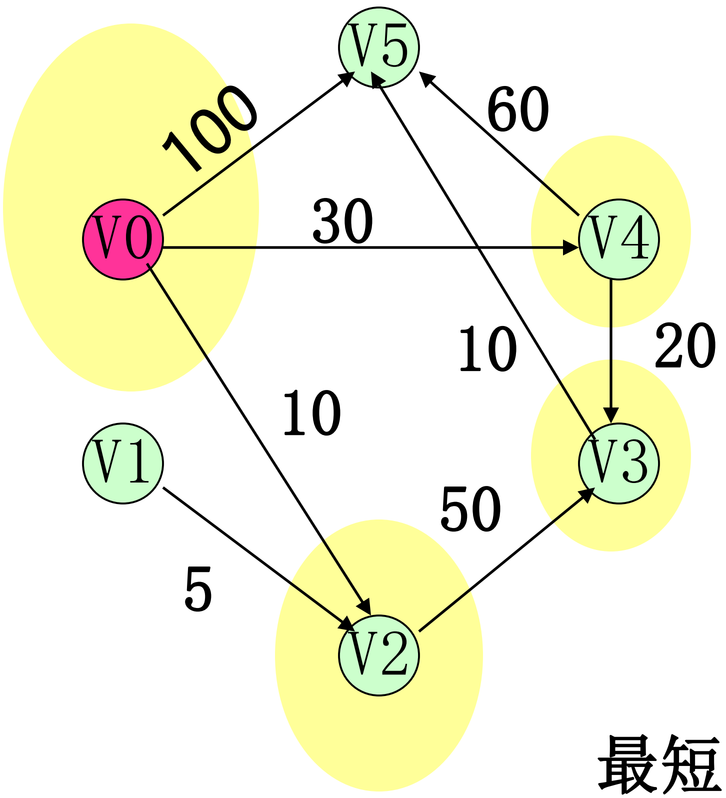
	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

+30

比较大小

∞	10	60	30	100

最短路径：	∞	10	50	30	90
	1	2	3	4	5
前驱顶点：	V0	V0	V4	V0	V4



	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

+50

比较大小

∞	10	50	30	90

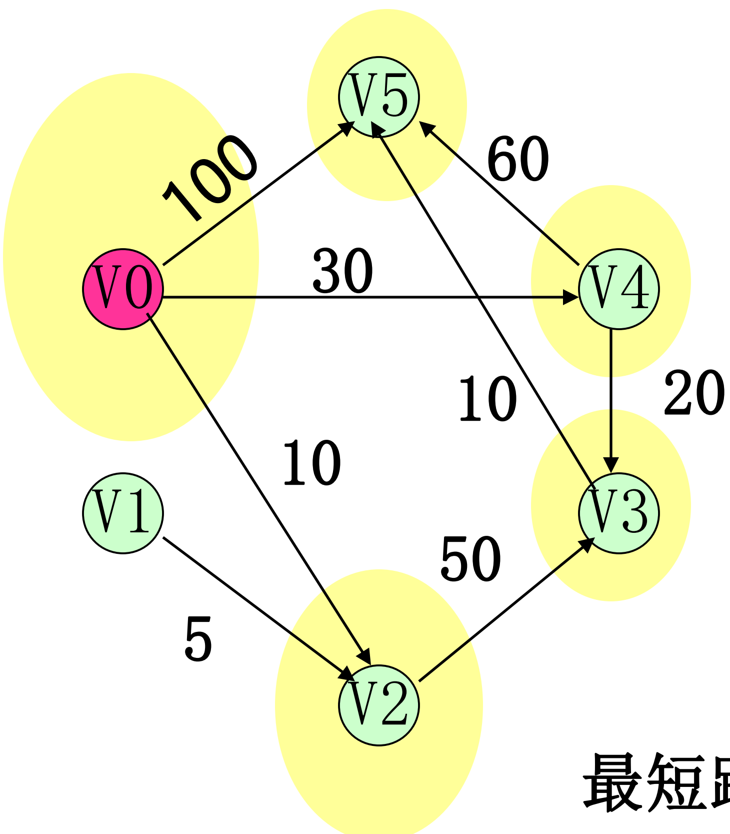
最短路径：

∞	10	50	30	60

1	2	3	4	5

前驱顶点：

V0	V0	V4	V0	V3



	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

+60

比较大小

∞	10	50	30	60

最短路径：

∞	10	50	30	60

前驱顶点：

1	2	3	4	5
V0	V0	V4	V0	V3

		1	2	3	4	5
最短路径：		∞	10	50	30	60
前驱序列：		V1	V2	V3	V4	V5
前驱顶点：		V0	V0	V4	V0	V3

V1：	无路径		
V2：	10	V2←V0	V0→V2
V3：	50	V3←V4 ←V0	V0→V4→V3
V4：	30	V4←V0	V0→V4
V5：	60	V5←V3 ←V4 ←V0	V0→V4→V3→V5

7.6.1 每一对顶点之间的最短路径

算法1（Dijkstra算法）：

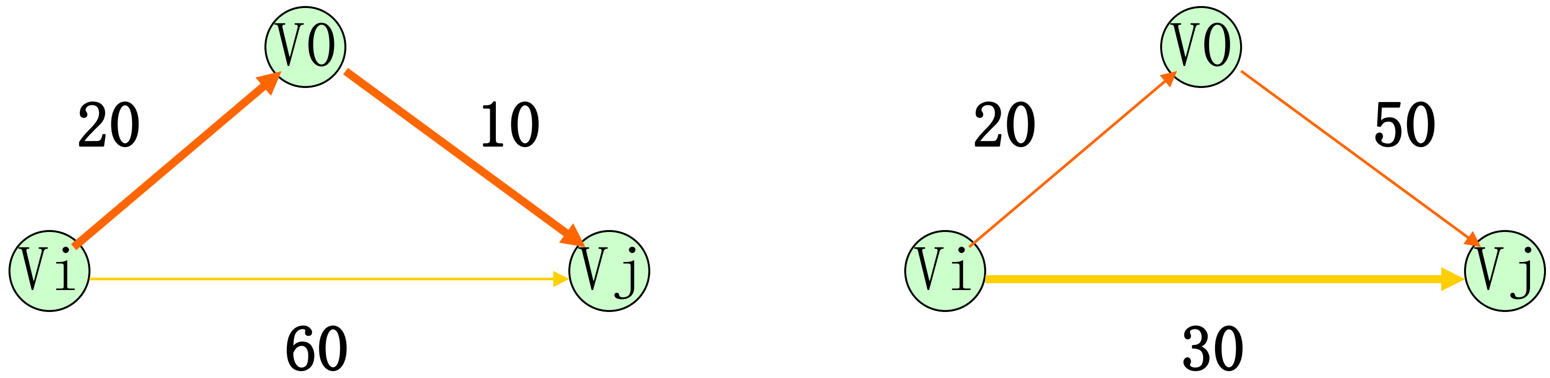
以每一个顶点为源点，重复执行Dijkstra算法n次，即可求出每一对顶点之间的最短路径。

算法2（Floyd算法）：

算法思想：

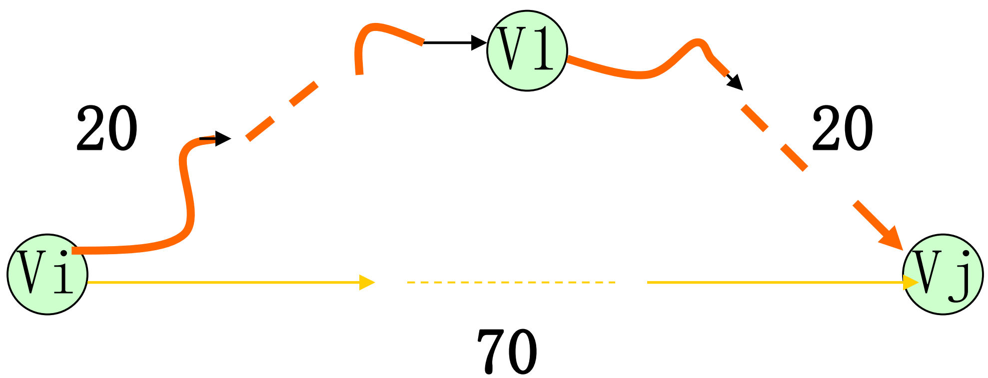
假设求Vi到Vj的最短路径，如果从Vi到Vj有弧，则存在一条长度为arcs[i][j]的路径，该路径不一定是最短路径，尚需进行n次试探。

首先考虑(Vi,V0,Vj)是否存在（即判断(Vi,V0)和(V0,Vj)是否存在)，如果存在，比较(Vi, Vj)和(Vi,V0)+(V0,Vj)，取长度较短的为从Vi到Vj的中间顶点序号不大于0的最短路径。



再考虑路径上再增加一个顶点V1，如果考虑(Vi,...V1)和(V1,.... Vj)，(Vi,...V1)和(V1,.... Vj)都是**中间顶点序号不大于0的最短路径。**(Vi,...V1，.... Vj)可能是**从Vi到Vj的中间顶点序号不大于1的最短路径**。

比较Vi到Vj的中间顶点序号不大于0的最短路径和(Vi,...V1)+(V1,.... Vj)，取长度较短的为**从Vi到Vj的中间顶点序号不大于1的最短路径。**



以此类推，经过n次比较后，求得Vi到Vj的最短路径。

假定邻接矩阵为cost[N][N]

Floyd算法的基本思想是递推产生一个矩阵序列:

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

$D^{(-1)}[i][j] = cost[i][j]$

$D^{(k)}[i][j] = Min\{\mathbf{D^{(k-1)}[i][j]},\ \mathbf{D^{(k-1)}[i][k] + D^{(k-1)}[k][j]}\}$

$0 \leqslant k \leqslant n-1$

算法C程序:

```
for(k=0;k<N;k++)          //依次选定中间顶点V0,V1,...Vn-1
 for(i=0; i<N;i++)        //i,j配合处理所有顶点Vi,Vj
   for(j=0; j<N;j++)
     if (cost[i][j]>cost[i][k]+cost[k][j])
       cost[i][j]=cost[i][k]+cost[k][j]; //取较短路径
```