数据结构

第7章 图(Graph)

7.1 图的定义和术语

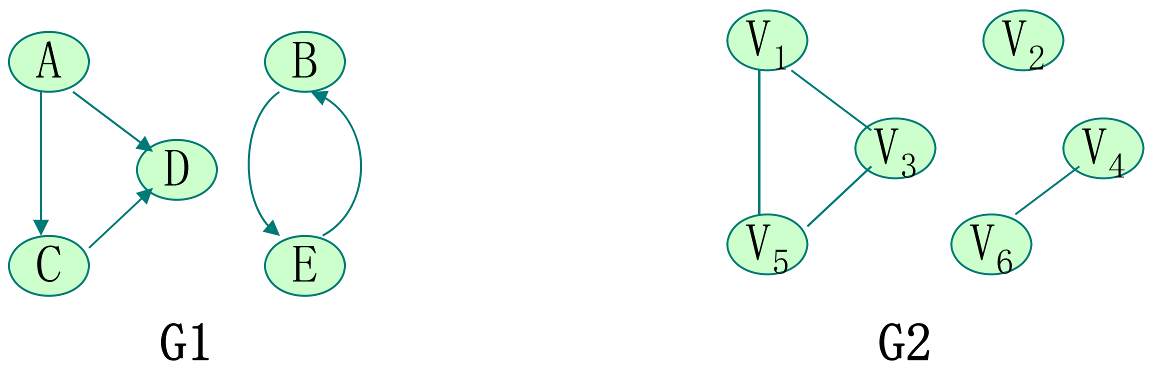
1.图的定义

图G由顶点集V和关系集VR组成,记为:

G=(V,VR)

V是顶点(元素)的有穷非空集,

VR是两个顶点之间的关系的集合。



2. 有向图、弧（有向边）：

若图G任意两顶点a, b之间的关系为有序对<a, b>, 即<a, b>∈VR, 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，

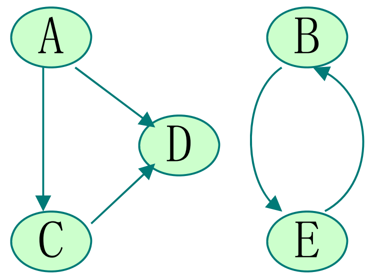
b是<a, b>的弧头；

例 G1=(V1, E1), V1={A, B, C, D, E}

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

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

称该图G1为有向图。



G1

3. 无向图、边（无向边）：

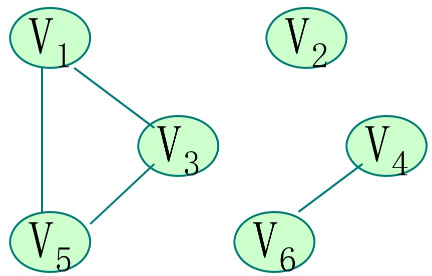
若图G的任意两顶点a,b之间的关系为无序对(a,b)， 则称(a,b)为无向边(边）,称该图G是无向图。 无向图可简称为图。

(a,b)表示a、b互为邻接点， (a,b)依附于a和b， (a,b)与a和b相关联

例 G2=(V2,E2),

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

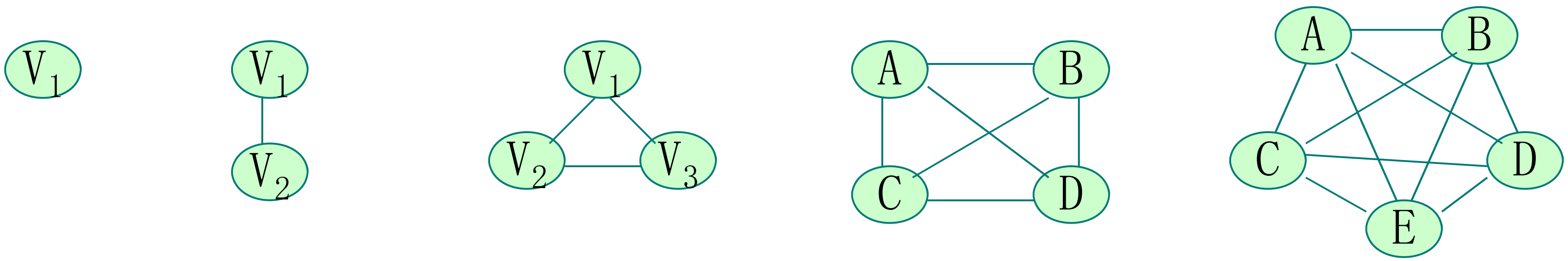
E2={(1,3),(1,5),(3,5),(4,6)}



G2

4. 完全图:

有n个顶点和n(n-1)/2条边的无向图



G1　G2　G3　G4　G5

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

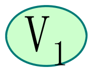
e=2(2-1)/2
=1

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

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

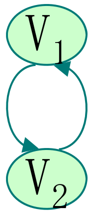
e=5(5-1)/2
=10

5. 有向完全图：有n个顶点和n(n-1)条弧的有向图。



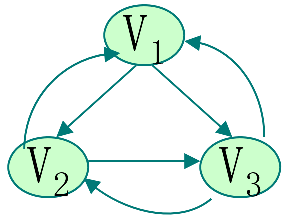
G1

e=1(1-1) =0



G2

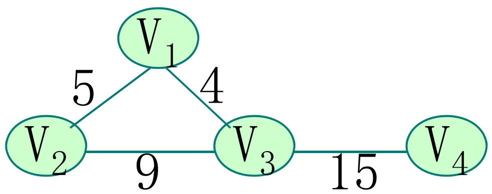
e=2(2-1)=2



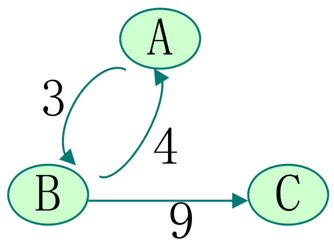
G3

e=3(3-1)=6

6. 网(Network)：边(弧)上加权(weight)的图。



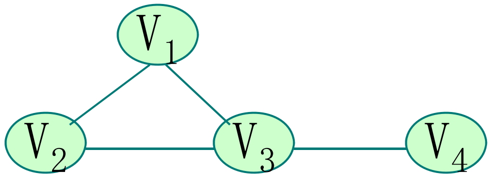
无向网G1



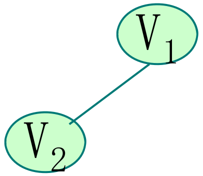
有向网G2

7. 子图：

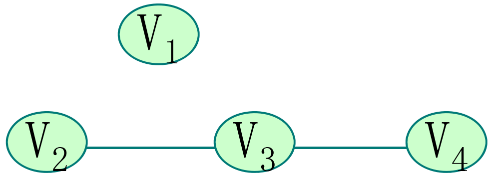
对图 G=(V,E)和G’=(V’,E’)，若V’$\subseteq$ V 且 E’$\subseteq$ E，则称G’是G的一个子图



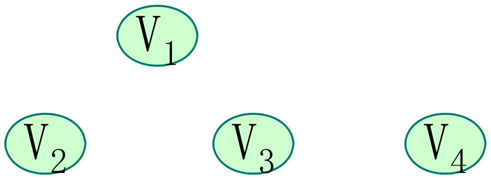
G



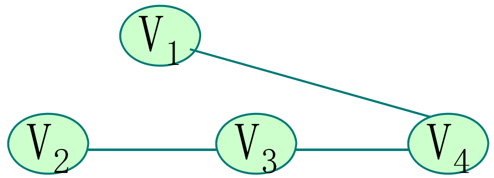
G1



G2



G3

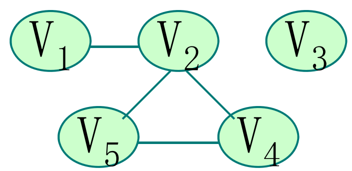


G4

G1,G2,G3是G的子图　G4不是G的子图

8. 度：

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)，



G1

$TD(V_1)=1$
$TD(V_2)=3$
$TD(V_3)=0$

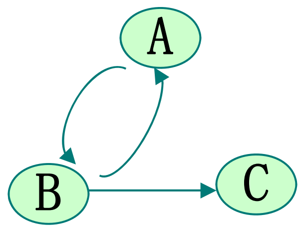
9. 出度:

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



G2

10. 入度:

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

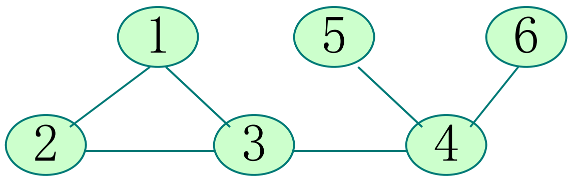
ID(C)=1

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

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

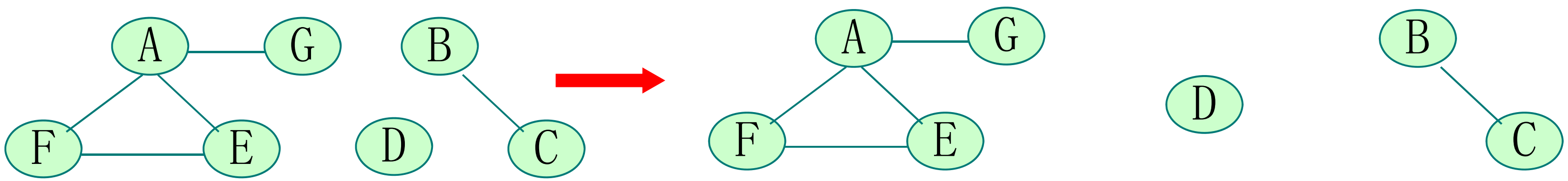
11. 连通图及连通分量：（无向图G）

- 若从顶点v_i到v_j有路径，则称v_i和v_j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。



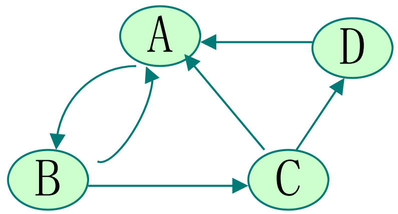
无向图G

G1 G2 G3

G有三个连通分量

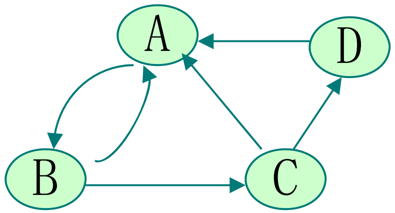
12. 强连通图及强连通分量：（有向图G）

- 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。

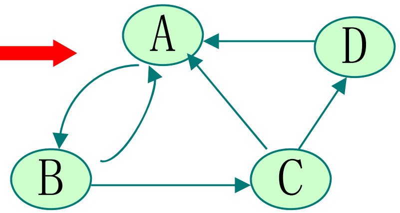


G1

- 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)

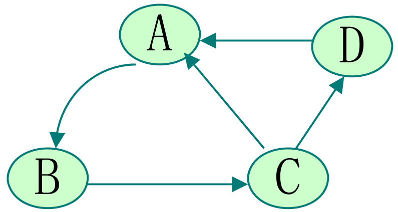


G1



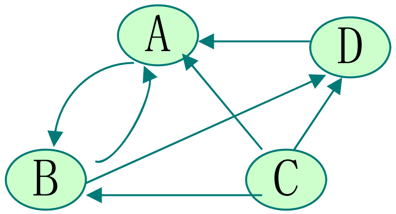
G11

是G1的强连通分量

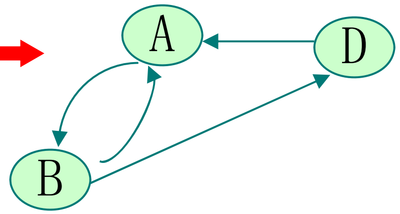


G12

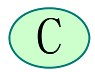
不是G1的强连通分量



G2



G21

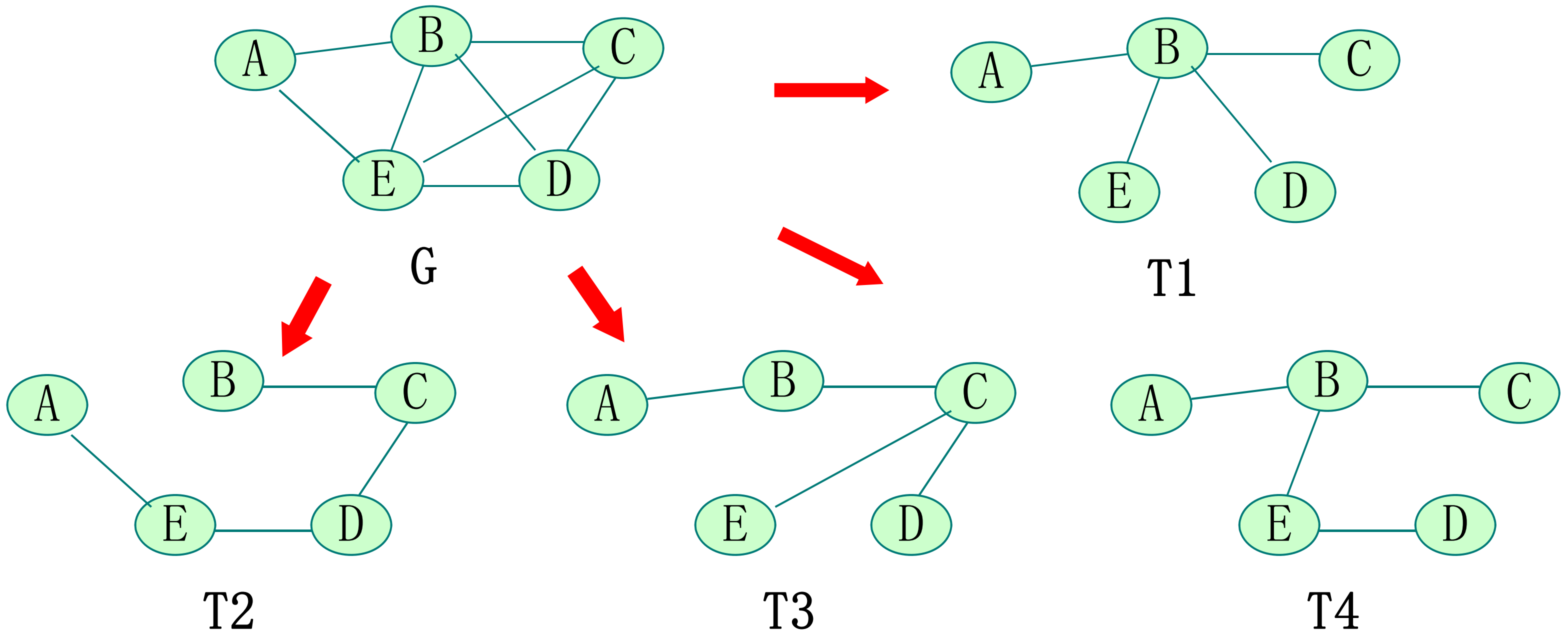


G22

G2有两个强连通分量

13.生成树:

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

14. 图的操作

(1) CreateCraph(&G,V,VR)　根据顶点集V和关系集VR生成图

(2) DestroyCraph(&G)　销毁图

(3) Locate(G,u)　查找顶点u的位置

(4) GetVex(G,v)　读取顶点v的信息

(5) PutVex(&G,v,value)　给顶点v的赋值value

(6) FirstAdjVex(G,v)　读v的第一个邻接顶点

(7) NextAdjVex(G,v,w)　读v(相对于w)的下一个邻接顶点

(8) InsertVex(&G,v)　插入顶点

(9) DeleteVex(&G,v)　删除顶点

(10) InsertArc(&G,v,w)　插入弧<v,w>

(11) DeleteArc(&G,v,w)　删除弧<v,w>

(12) DFSTraverse(G,visit())　深度优先遍历图

(13) BFSTraverse(G,visit())　宽度优先遍历图

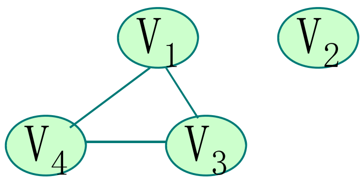
……

7.2 图的存储结构

7.2.1 数组表示法/邻接矩阵(顺序+顺序)

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)



无向图

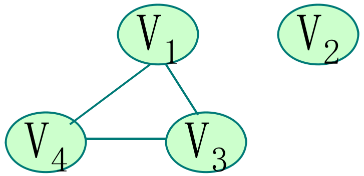
V_1	V_2	V_3	V_4
0	1	2	3

顶点数组vexs

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

邻接矩阵arcs

例1



G

V[0...max_vertext_num-1]

V_1	V_2	V_3	V_4	…
0	1	2	3	…

顶点数组

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

邻接矩阵

顶点vi的 $TD(v_i)$=邻接矩阵arcs中第i行元素之和

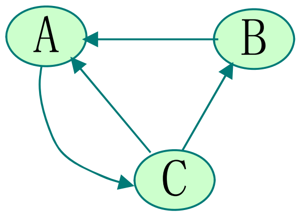
$$=\sum_{j=0}^{n-1} arcs[i][j]$$

或:

顶点vi的 $TD(v_i)$=邻接矩阵arcs中第i列元素之和

$$=\sum_{j=0}^{n-1} arcs\ [j][i]$$

例2



A	B	C
0	1	2 …

顶点数组vexs

$$\begin{array}{c} \\ 0 \\ 1 \\ 2 \end{array}\begin{array}{c} \begin{array}{ccc} 0 & 1 & 2 \end{array} \\ \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix} \end{array}$$

邻接矩阵arcs

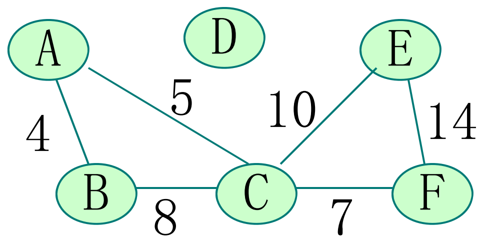
顶点vi的 $OD(v_i)$=邻接矩阵arcs中第i行元素之和

$$= \sum_{j=0}^{n-1} arcs[i][j]$$

顶点v_i的 $ID(v_i)$=邻接矩阵arcs中第i列元素之和

$$= \sum_{j=0}^{n-1} arcs\,[j][i]$$

例3



网G

$$M=\begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array}\begin{array}{c} \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\ \left(\begin{array}{cccccc} \infty & 4 & 5 & \infty & \infty & \infty \\ 4 & \infty & 8 & \infty & \infty & \infty \\ 5 & 8 & \infty & \infty & 10 & 7 \\ \infty & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & 10 & \infty & \infty & 14 \\ \infty & \infty & 7 & \infty & 14 & \infty \end{array}\right) \end{array}$$

邻接矩阵

思考题：

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

数组表示法的数据类型定义

```
#define MAX_VERTEX_NUM   20
typedef  enum{DG,DN,UDG,UDN} GraphKind;
typedef  struct {
    VertexType vexs[MAX_VERTEX_NUM];
    VRType arcs[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
    int vexnum,arcnum;
    GraphKind kind;
}Mgraph;
```

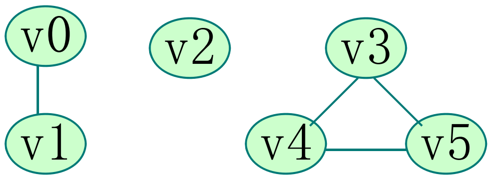
0	1	2	3		19
V_1	V_2	V_3	V_4	…	…

$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & \cdots\cdots & 19 \\
\hline
0 & 0 & 0 & 1 & 1 & \cdots\cdots & \\
1 & 0 & 0 & 0 & 0 & \cdots\cdots & \\
2 & 1 & 0 & 0 & 1 & & \\
3 & 1 & 0 & 1 & 0 & & \\
4 & \cdots\cdots & & & & & \\
. & & & & & & \\
19 & \cdots\cdots & & & & \cdots\cdots &
\end{array}
$$

7.2.2 邻接表、逆邻接表(顺序+链式)：

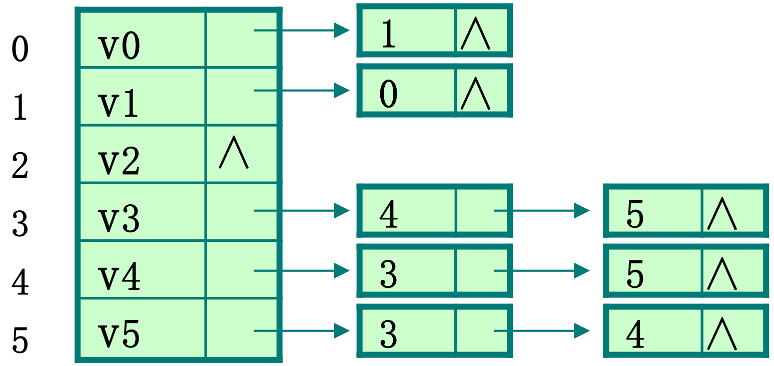
（1） 无向图的邻接表 ：

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v_i的边。



图G

序号 头结点数组 表结点单链表



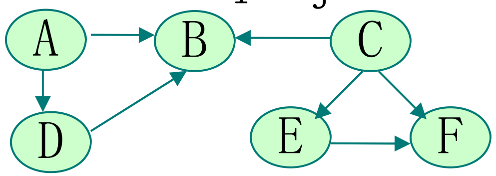
图G邻接表

- 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v_i的度为第i个单链表的长度。

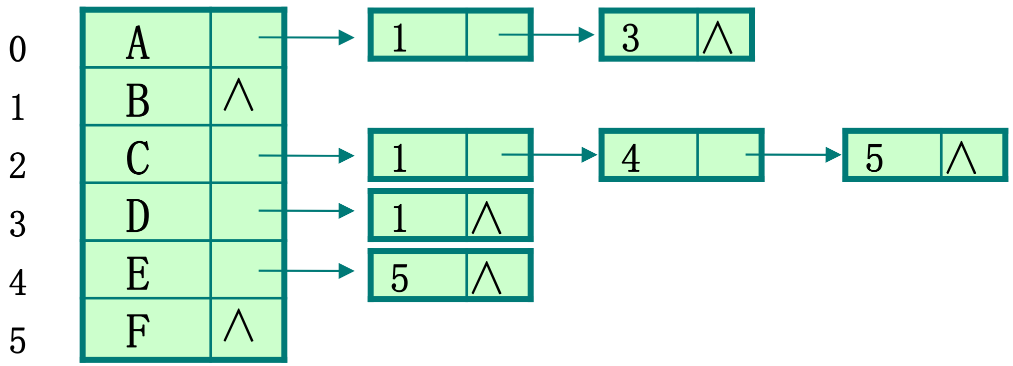
（2）有向图的邻接表 ：

第i个单链表中的表结点的值为j，表示以顶点v_i为弧尾的一条弧(v_i, v_j) 。

序号 头结点数组 表结点单链表



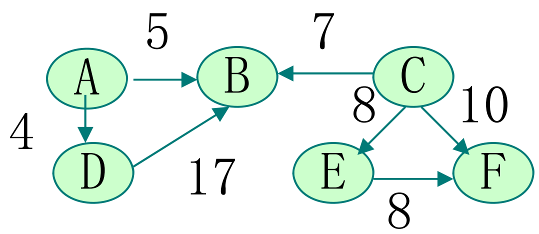
有向图G



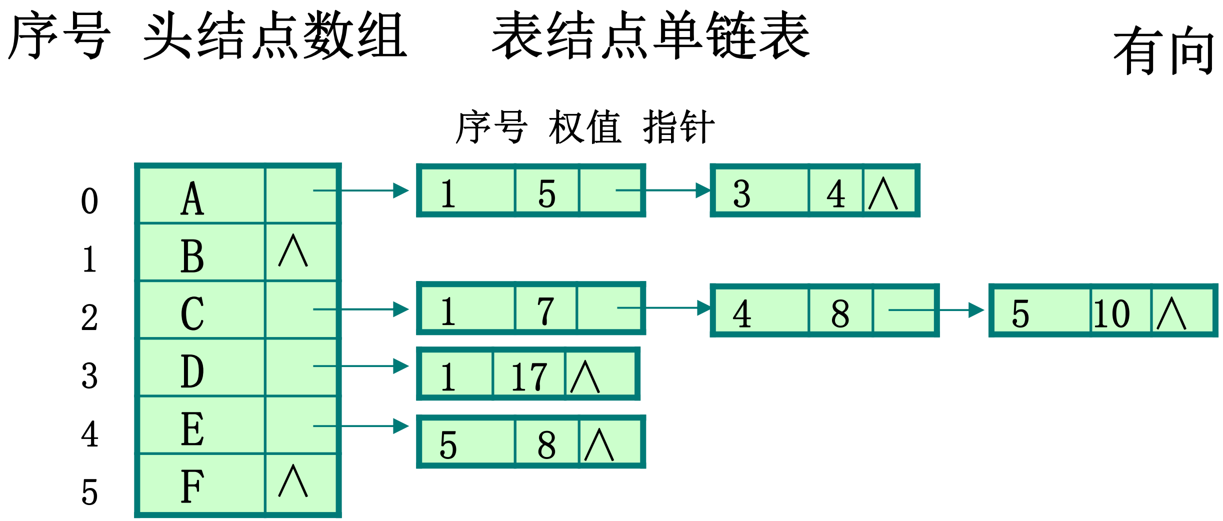
图G的邻接表

- 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）有向网的邻接表



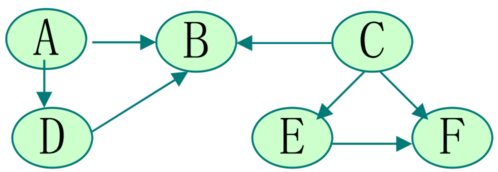
有向网G



有向网G的邻接表

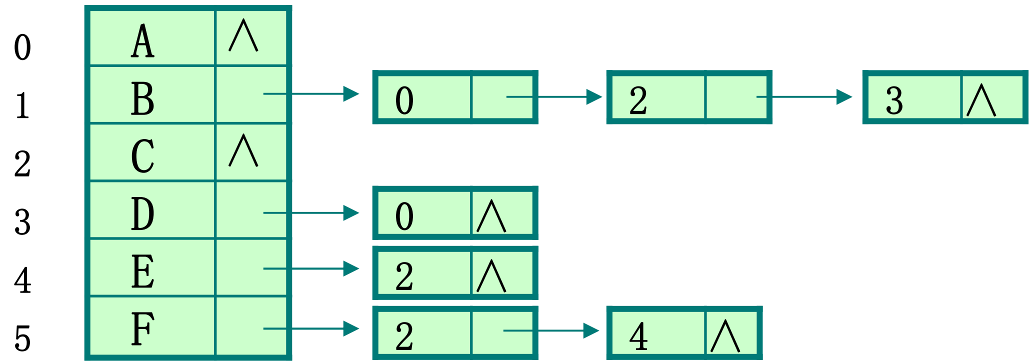
（4）有向图的逆邻接表

第j个单链表中的表结点的值为i，表示一条以顶点v_i为弧尾的一条弧(v_i, v_j)。



有向图G

序号　头结点数组　　表结点单链表



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。

邻接表表示法的数据类型定义

```
#define MAX_VERTEX_NUM 20
typedef struct ArcNode {        //表结点类型定义，对网需要加权值属性
   int adjvex;                  //顶点位置编号
   struct ArcNode *nextarc;     //下一个表结点指针
   InfoType *info;
} ArcNode;
typedef struct VNode{           //头结点及其数组类型定义
   VertexType data;             //顶点信息
   ArcNode *firstarc;           //指向第一条弧
   }VNode,AdjList[MAX_VERTEX_NUM];
typedef  struct {               //邻接表的类型定义
      AdjList vertices;         //头结点数组
      int vexnum,arcnum;        //顶点数、弧数
      GraphKind  kind;          //图的类型
     } ALGraph;
```

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。

（1）每条弧有一个弧结点。

弧结点：

tailvex	headvex	hlink	tlink

其中：tailvex：弧尾的位置； headvex：弧头的位置；

hlink： 指向下一条弧头相同的弧；

tlink： 指向下一条弧尾相同的弧。

（2）每个顶点有一个顶点结点。

顶点结点

data	firstin	firstout

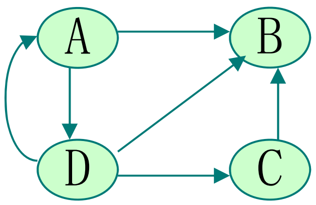
其中:：data：顶点信息；

firstin： 指向以该顶点为弧头的第一条弧；

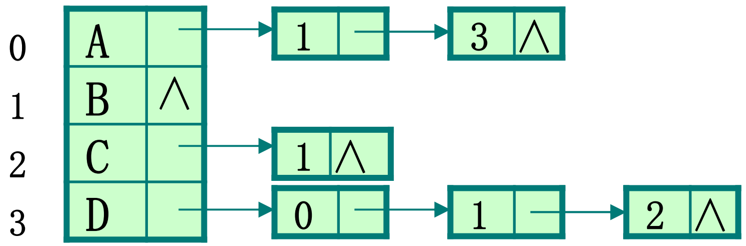
firstout：指向以该顶点为弧尾的第一条弧。

有向图十字链表存储表示的数据类型定义：

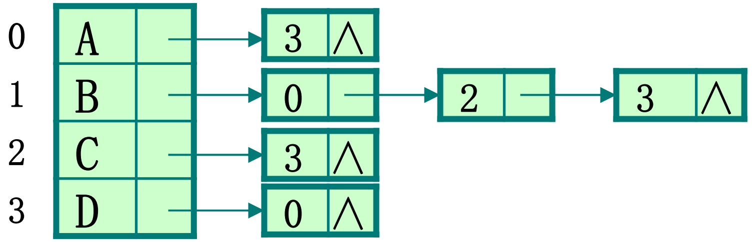
```
#define MAX_VERTEXT_NUM 20
typedef struct ArcBox {
    int          tailvex,headvex;   //该弧的尾和头顶点的位置
    struct ArcBox *hlink, *tlink;   //分别为弧头相同和弧尾相同的弧的链域
}  ArcBox;
typedef struct VexNode {
    VertexType    data;
    ArcBox        *firstin,*firstout;//分别指向该顶点的第一条入弧和出弧
}VexNode;
typedef struct {
    VexNode  xlist[MAX_VERTEXT_NUM];  //表头向量
    int          vexnum,arcnum;                     //有向图的当前顶点数和弧数
} OLGraph;
```



G

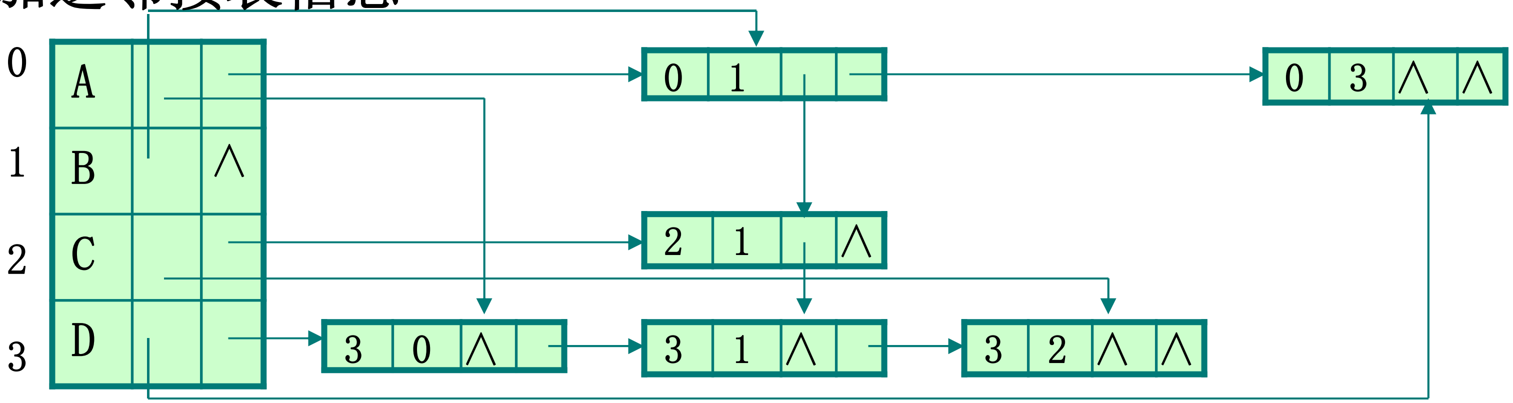


G的邻接表



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 (无向图)邻接多重表

（1）每个顶点有一个头结点

头结点

data	firstedge

其中： data： 顶点信息；

firstedge： 指向第一条依附于该顶点的边。

（2）每一条边有一个表结点

表结点

mark	ivex	jvex	ilink	jlink

其中：

mark： 标志域，可用以标记该条边是否被搜索过；

ivex、jvex：该条边依附的两个顶点在顶点数组的位置；

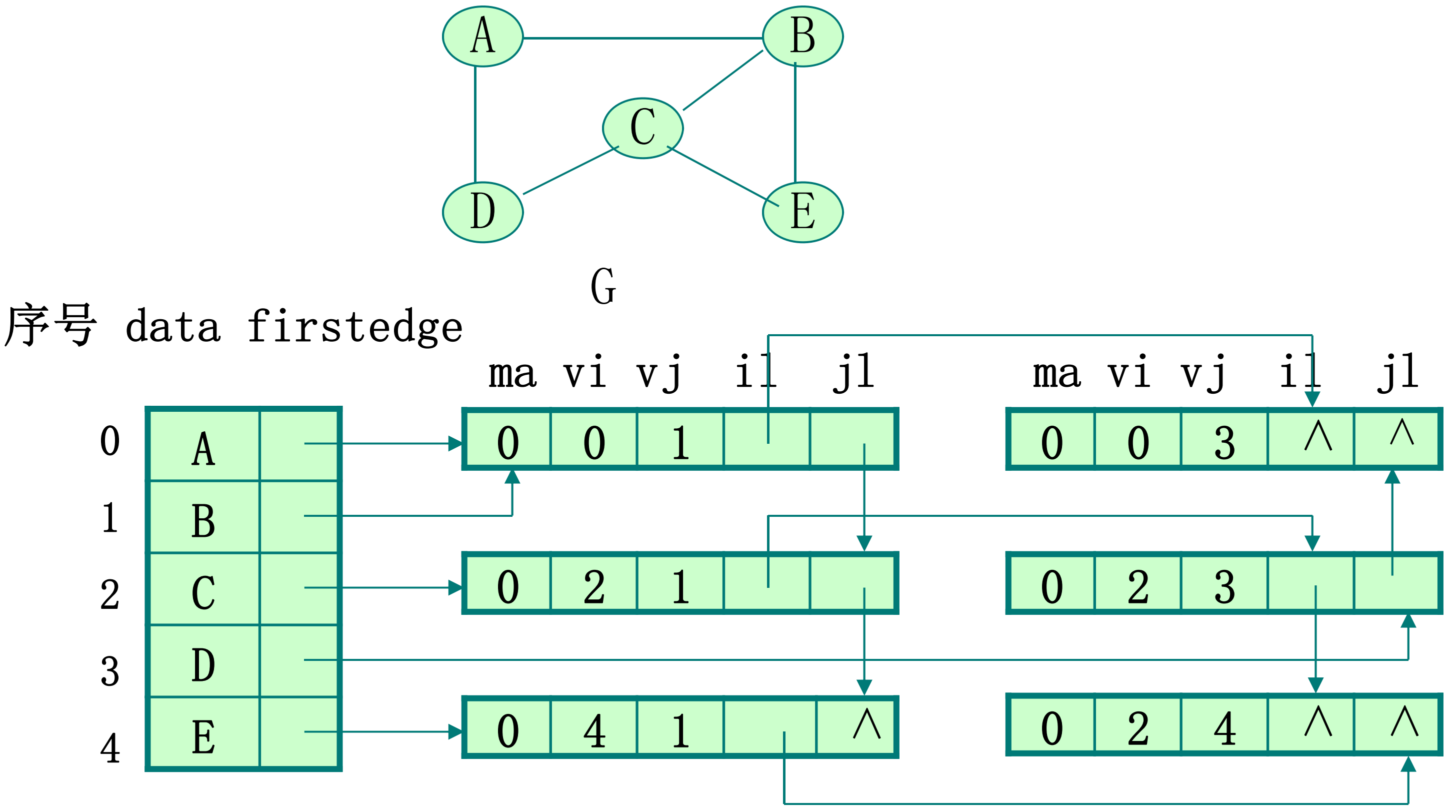
ilink：指向下一条依附于顶点vi的边；

jlink：指向下一条依附于顶点vj的边。

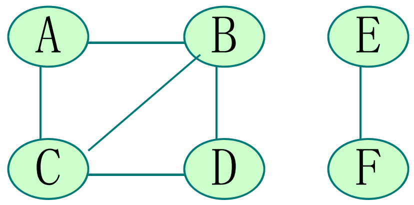
无向图邻接多重表存储表示的数据类型定义：

```
#define MAX_VERTEXT_NUM 20
typedef enum {unvisited,visited} visitedIf;
typedef  struct  EBox{
    visitedIf       mark;              //访问标记
    int             ivex,jvex;         //该边依附的两个顶点的位置
    struct EBox  *ilink,*jlink;     //分别指向依附于这两个顶点的下一条边
} EBox;
typedef  struct  VexBox{
    VertexType data;
    EBox *firstedge;                 //指向第一条依附于该顶点边
} VexBox;
typedef  struct {
    VexBox adjmullist[MAX_VERTEXT_NUM];
    int       vexnum,edgenum;   //无向图的当前顶点数和边数
} AMLGraph;
```

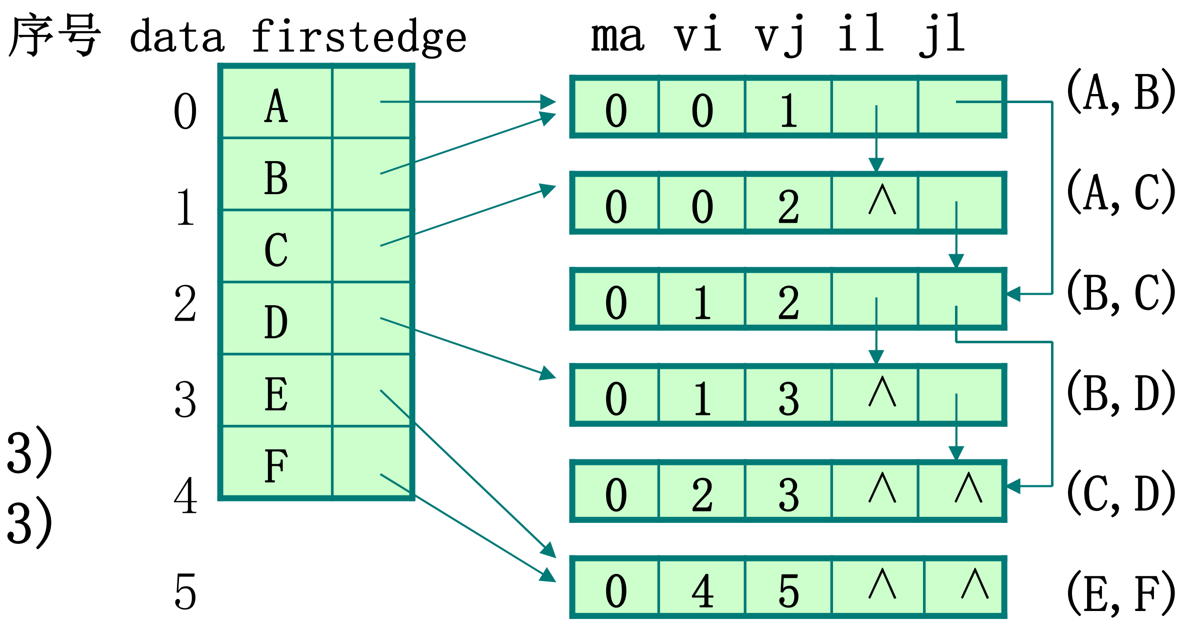
7.2.4 邻接多重表（续）


A
B
C
D
E
G
序号 data firstedge
ma vi vj il jl
ma vi vj il jl
0 A
1 B
2 C
3 D
4 E
0 0 1
0 0 3 ∧ ∧
0 2 1
0 2 3
0 4 1 ∧
0 2 4 ∧ ∧

7.2.4 邻接多重表（续）



G



隐含的链接表：

A →(0, 1) →(0, 2)

B →(0, 1) →(1, 2) →(1, 3)

C →(0, 2) →(1, 2) →(2, 3)

D →(1, 3) →(2, 3)

E →(4, 5)

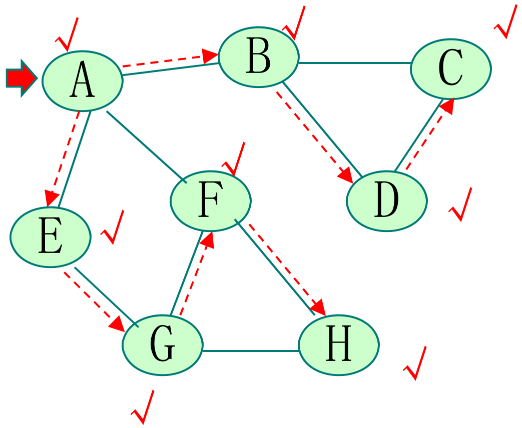
F →(4, 5)

7.3 图的遍历

从图G的某定点v_i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

本次访问的顶点序列：

A E G F H B D C

其它的顶点访问序列：

A F G E H B D C

A B C D E G H F

……

不可能的顶点访问序列：

A E C F H B D G

……

深度优先搜索遍历算法代码（假定结点序号从0开始）：

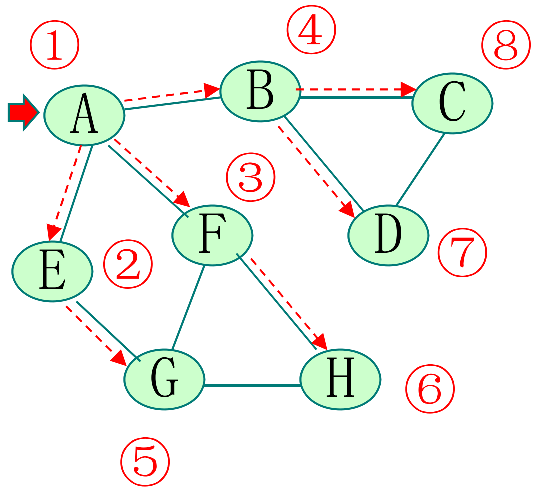
```
boolean visited[MAX];
void DFSTraverse(Graph G,Status (*visit)()) {
  for(v=0;v<G.vexnum; v++)     //初始化各顶点未访问状态
     visited[v]=false;
  for(v=0;v<G.vexnum; v++)
     if (!visited[v])          //从一个未访问的顶点开始
          DFS(G, v, visit);
}
void DFS(Graph G,int v,Status (*visit)()) {
  visited[v]=true; visit(v);
  for(w=FirstAdjVex(G,v),w>=0; w=NextAdjVex(G,v,w))
       if (!visited[w])      //处理所有未访问的邻接顶点
            DFS(G, w,visit);
}
```

邻接矩阵： T（n）=O（n^2）

邻 接 表： T（n）=O（n+e）

7.3.2 图的广(宽)度优先搜索

BFS(Breadth First Search)



图G

本次访问的顶点序列:

A E F B G H D C

其它的顶点访问序列:

A B F E C D H G

A F B E H G C D

……

不可能的顶点访问序列:

A E F B C D H G

……

广度优先搜索遍历算法代码:

```
void BFSTraverse(Graph G,Status (*visit)()) {
  for(v=0; v<G.vexnum; v++)
    visited[v]=false;
  InitQueue(Q);
  for(v=0;v<G.vexnum; v++)   //按顶点位置序号依次选择顶点
    if (!visited[v])  {      //遇到未访问过的顶点开始遍历
      visited[v]=true; visit(v); EnQueue(Q,v);
      while(!QueueEmpty(Q)){
         DeQueue(Q,u);
         for(w=FirstAdjVex(G,u),w>=0; w=NextAdjVex(G,u,w))
           if (!visited[w])
             { visited[w]=true; visit(w);EnQueue(Q,w);}
      }
    }
}
```

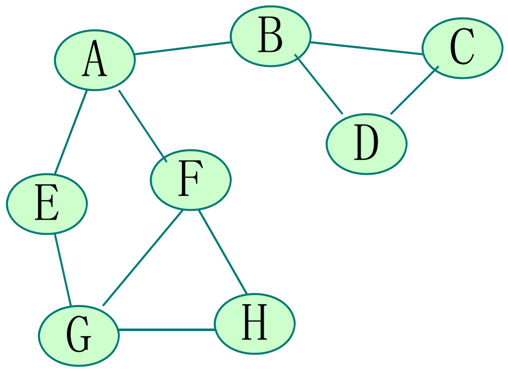
邻接矩阵: T(n)=O(n^2)

邻 接 表: T(n)=O(n+e)

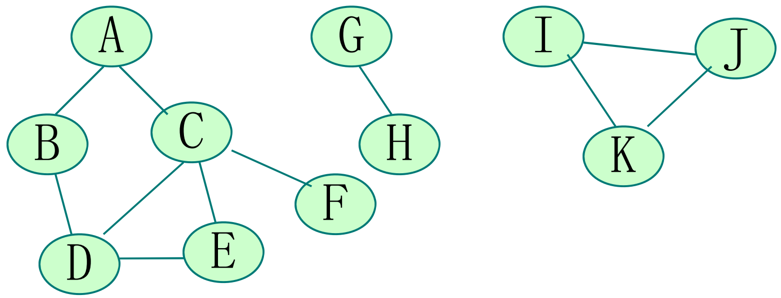
7.4 图的连通性问题

7.4.1 无向图的连通分量和生成树

（1）连通分量

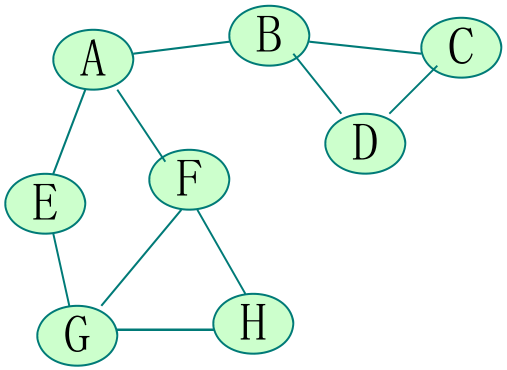


图G1

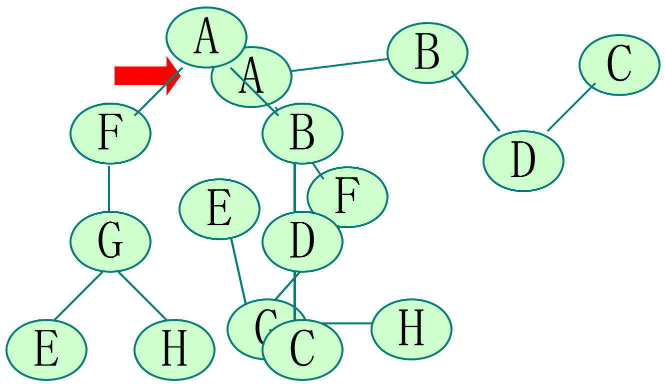


图G2

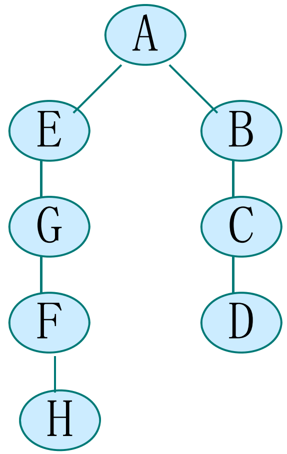
（2）DFS生成树



图G

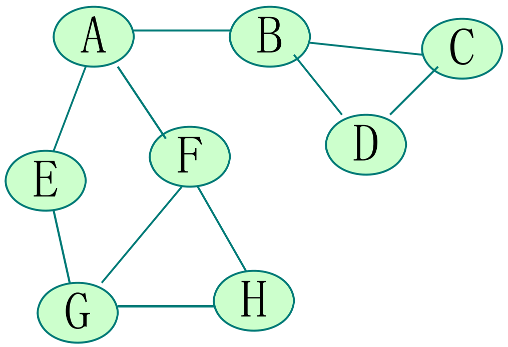


DFS生成树T1

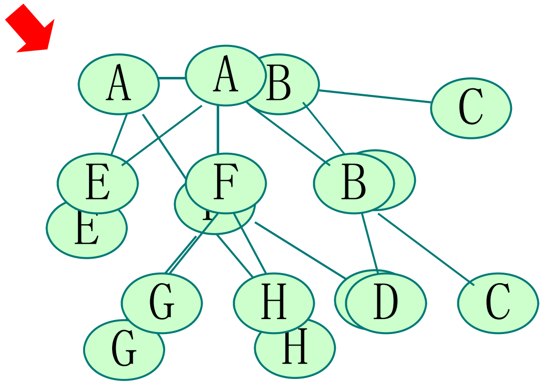


其它： DFS生成树T2 ……

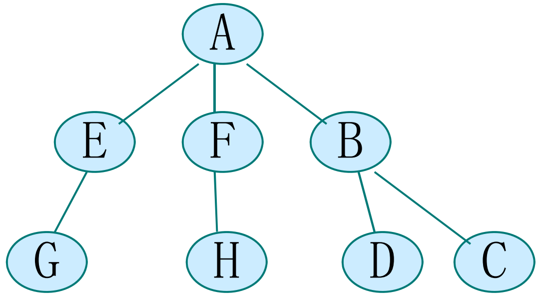
（3）BFS生成树



图G

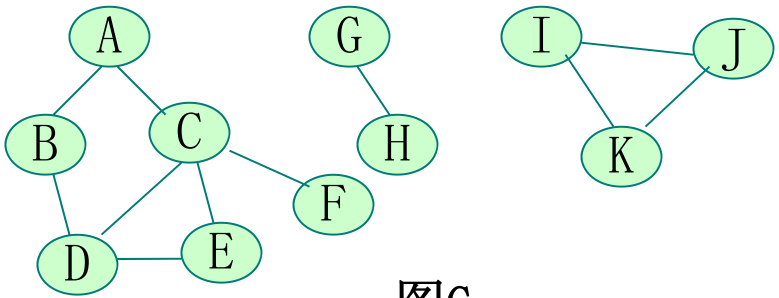


BFS生成树T1



其它：BFS生成树T2 ……

（4）DFS生成森林

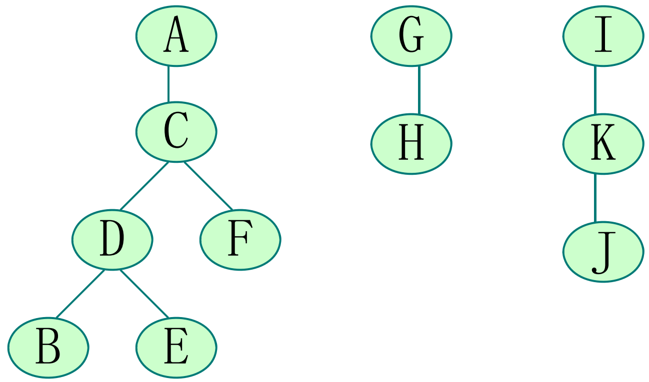


图G

从A出发，得树T1：

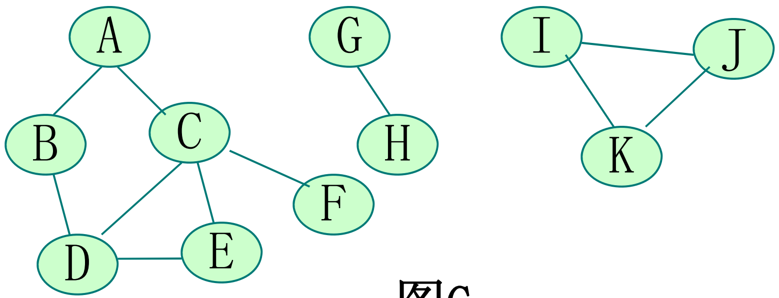
从G出发,得树T2：

从I出发,得树T3：



T1 T2 T3

（5） BFS生成森林

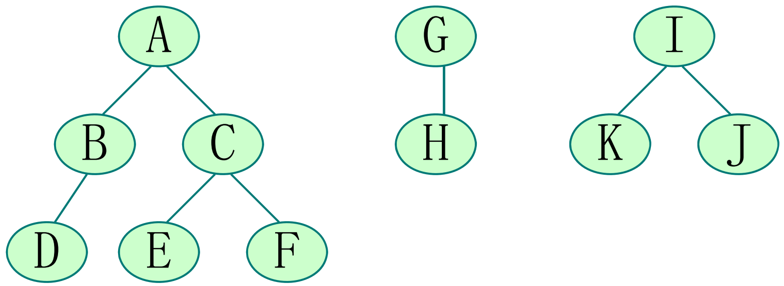


图G

从A出发，得树T1：

从G出发,得树T2：

从I出发,得树T3：

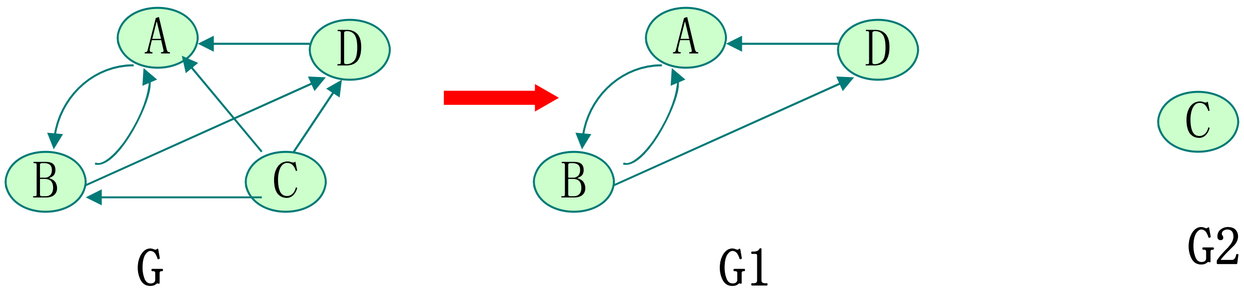


T1　　T2　　T3

7.4.2 有向图的强连通分量:

在有向图G中，从某个顶点v出发，顺着弧的方向进行深度优先搜索遍历，得到顶点集合V_1；再顶点v出发，逆着弧的方向进行深度优先搜索遍历，得到顶点集合V_2；这样得到一个强连通分量:

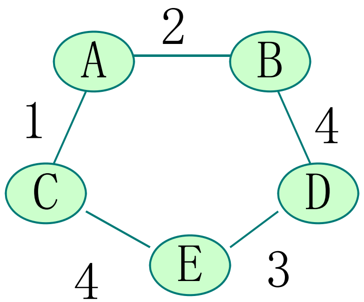
G_s=（V_s,VR_s），其中：$V_s = V_1 \cap V_2$；VR_s为V_s中所有顶点在G中的弧.



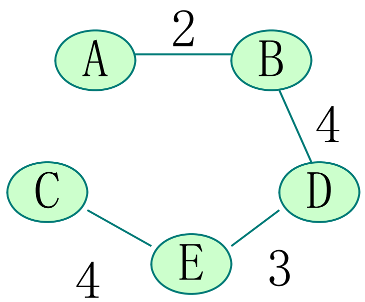
从A出发，顺着弧的方向得到顶点集合：{A、B、D}；逆的弧的方向得到：{A、B、C、D}；交集为： {A、B、D}；加上它们之间的所有弧得到强连通分量G1

7.4.3 网的最小生成树：

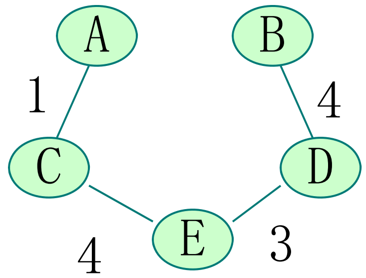
在网G的各生成树中，其中各边的权之和最小的生成树称为G的最小生成树



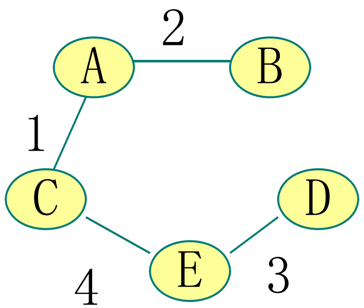
网G



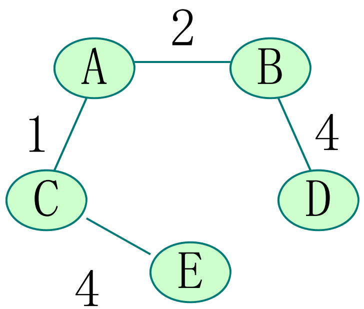
生成树T1（13）



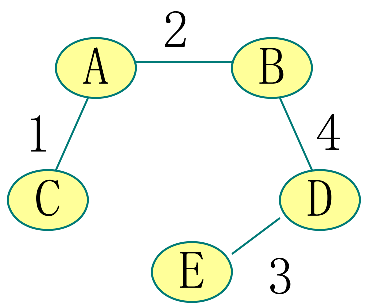
生成树T2（12）



生成树T3（10）



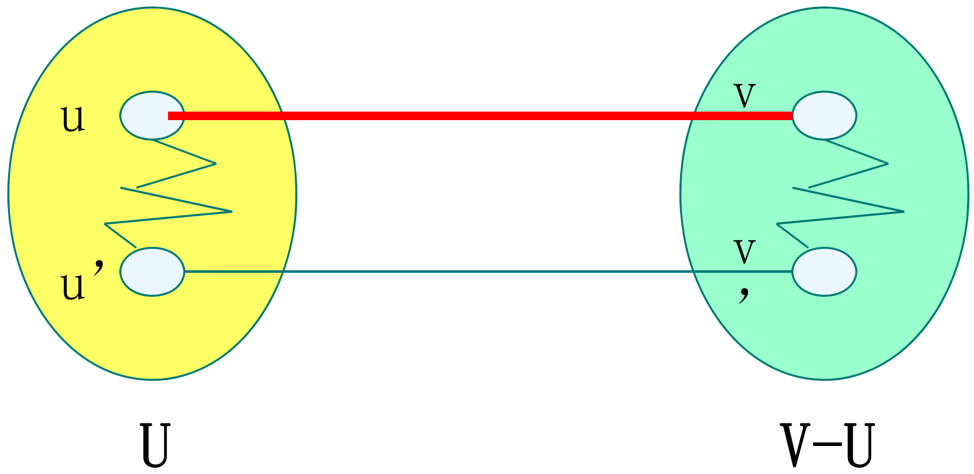
生成树T4（11）



生成树T5（10）

MST性质：

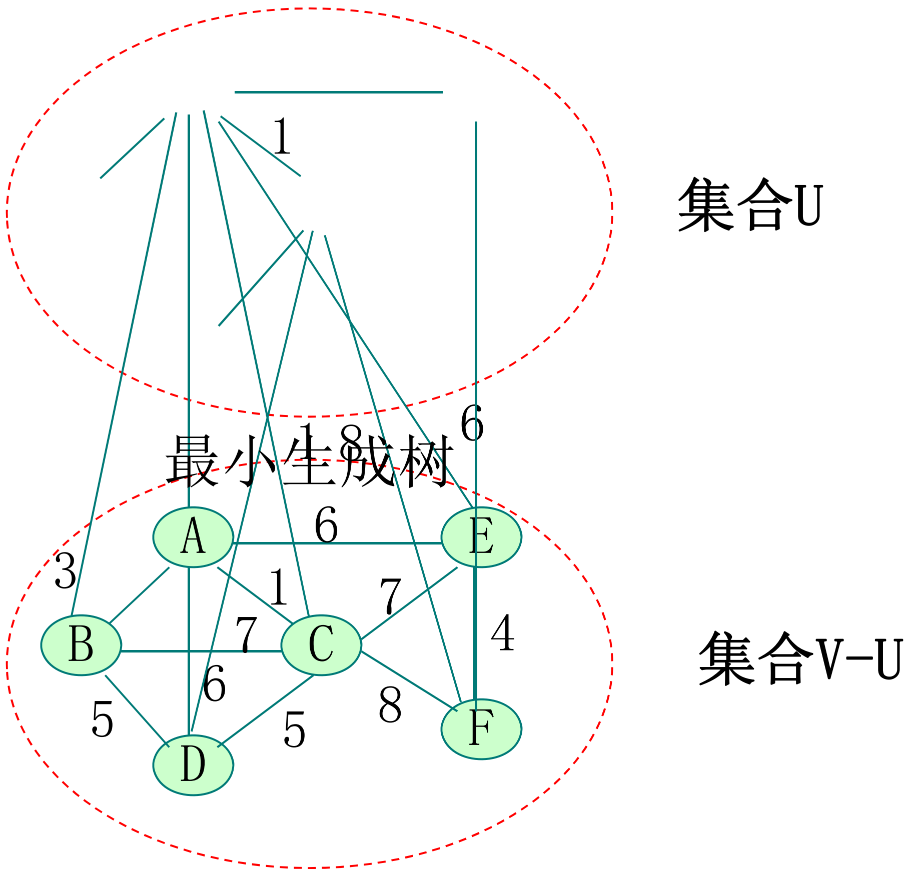
设G=（V，E）是一个连通网，U是V的一个非空子集。如果边（u,v)是G中所有一端在U中（即u∈U）而另一端在V-U中（即v∈V-U）具有最小值的一条边，则存在一棵包含边（u,v)的最小生成树。



1.普里姆(prim)算法　　以选顶点为主

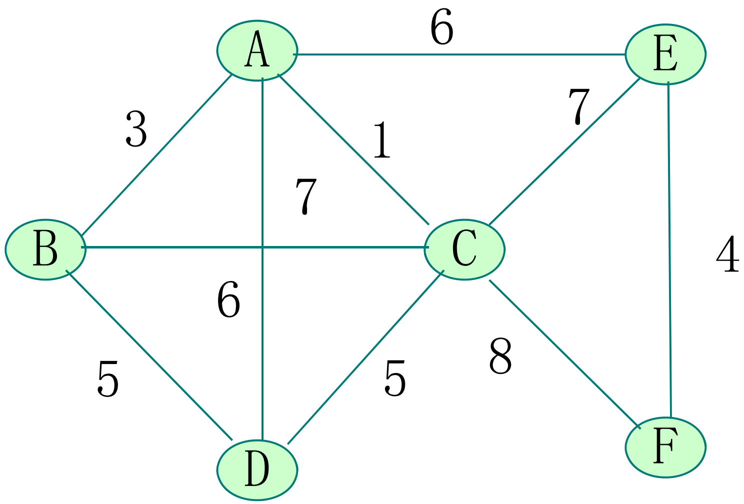
　　对n个顶点的连通网，初始时，T=（U，TE），U为一个开始顶点，TE=Φ，以后根据MST性质，每次增加一个顶点和一条边，重复n-1次。U不断增大，V－U不断减小直到为空。

例：从A出发

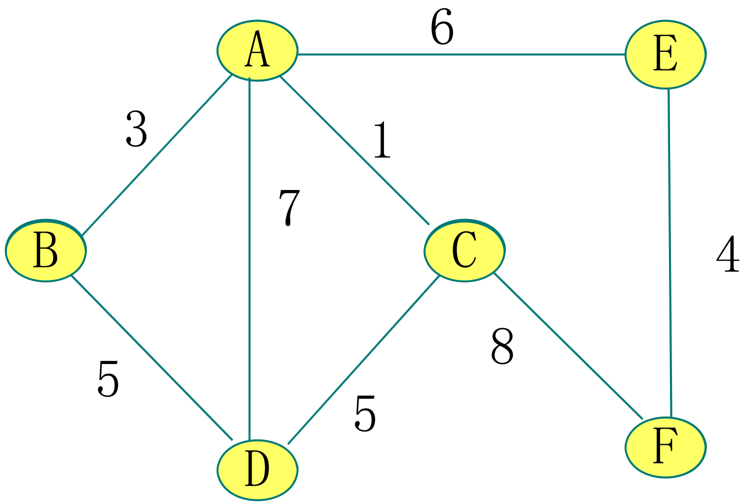


1.普里姆(prim)算法(续)

另一棵最小生成树：从D出发

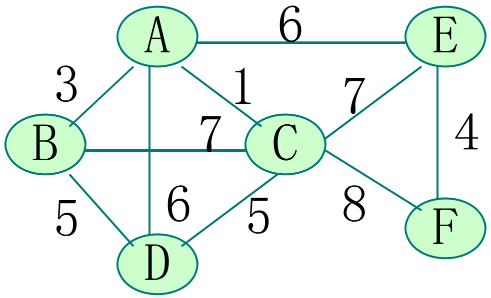


网G



最小生成树T

Prime算法思想：



图G

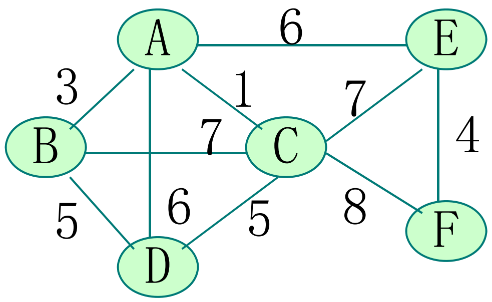
0表示U中的顶点，
其它为V-U的顶点

$$
\begin{array}{c}
 \\ A \\ B \\ C \\ D \\ E \\ F
\end{array}
\begin{array}{c}
\begin{array}{cccccc} A & B & C & D & E & F \end{array} \\
\left(\begin{array}{cccccc}
\infty & 3 & 1 & 6 & 6 & \infty \\
3 & \infty & 7 & 5 & \infty & \infty \\
1 & 7 & \infty & 5 & 7 & 8 \\
6 & 5 & 5 & \infty & \infty & \infty \\
6 & \infty & 7 & \infty & \infty & 4 \\
\infty & \infty & 8 & \infty & 4 & \infty
\end{array}\right)
\end{array}
$$

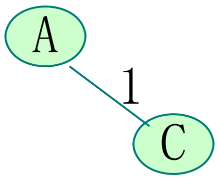
初始化

V-U中各顶点到U的最短边的长度：	0	3	1	6	6	∞
相邻顶点adjvex：	A	A	A	A	A	A
	0	1	2	3	4	5
	A	B	C	D	E	F

Prime算法:



图G

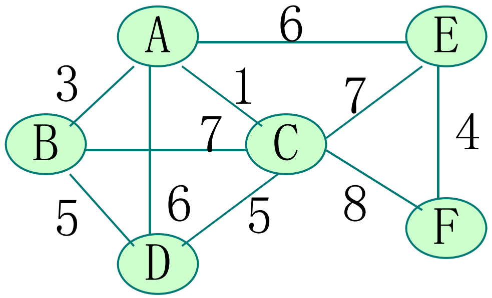


$$
\begin{array}{c|cccccc}
 & A & B & C & D & E & F \\
\hline
A & \infty & 3 & 1 & 6 & 6 & \infty \\
B & 3 & \infty & 7 & 5 & \infty & \infty \\
C & 1 & 7 & \infty & 5 & 7 & 8 \\
D & 6 & 5 & 5 & \infty & \infty & \infty \\
E & 6 & \infty & 7 & \infty & \infty & 4 \\
F & \infty & \infty & 8 & \infty & 4 & \infty
\end{array}
$$

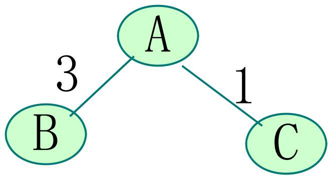
比较大小

0	3	0	5	6	8
A	A	A	C	A	C
0	1	2	3	4	5
A	B	C	D	E	F

Prime算法：



图G

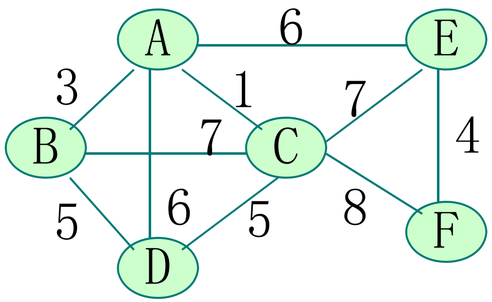


	A	B	C	D	E	F
A	∞	3	1	6	6	∞
B	3	∞	7	5	∞	∞
C	1	7	∞	5	7	8
D	6	5	5	∞	∞	∞
E	6	∞	7	∞	∞	4
F	∞	∞	8	∞	4	∞

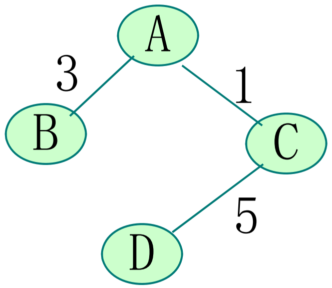
比较大小

0	**0**	**0**	**5**	**6**	**8**
A	**A**	**A**	**C**	**A**	**C**
0	**1**	**2**	**3**	**4**	**5**
A	**B**	**C**	**D**	**E**	**F**

Prime算法:



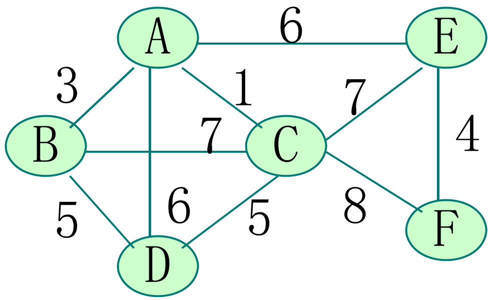
图G



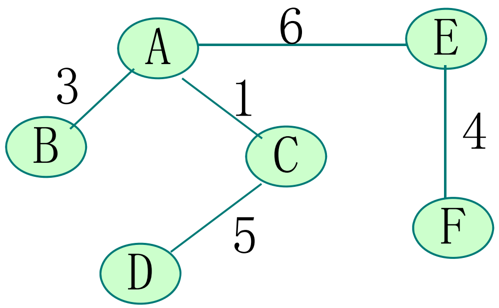
	A	B	C	D	E	F
A	∞	3	1	6	6	∞
B	3	∞	7	5	∞	∞
C	1	7	∞	5	7	8
D	6	5	5	∞	∞	∞
E	6	∞	7	∞	∞	4
F	∞	∞	8	∞	4	∞

0	0	0	0	6	8
A	A	A	C	A	C
0	1	2	3	4	5
A	B	C	D	E	F

Prime算法:



图G



最小生成树

$$
\begin{array}{c|cccccc}
 & \mathbf{A} & \mathbf{B} & \mathbf{C} & \mathbf{D} & \mathbf{E} & \mathbf{F} \\
\hline
A & \infty & 3 & 1 & 6 & 6 & \infty \\
B & 3 & \infty & 7 & 5 & \infty & \infty \\
C & 1 & 7 & \infty & 5 & 7 & 8 \\
D & 6 & 5 & 5 & \infty & \infty & \infty \\
E & 6 & \infty & 7 & \infty & \infty & 4 \\
F & \infty & \infty & 8 & \infty & 4 & \infty
\end{array}
$$

比较大小

0	0	0	0	0	4
A	A	A	C	A	E
0	**1**	**2**	**3**	**4**	**5**
A	**B**	**C**	**D**	**E**	**F**

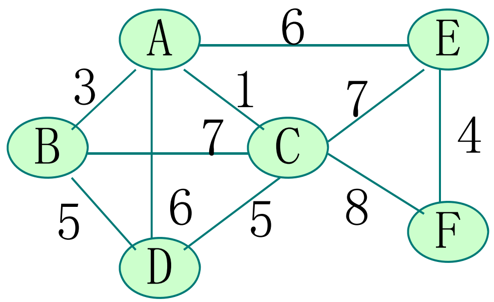
Prim算法代码(closedge包含最小权值与依附顶点2个属性)：

```
void Prim(MGraph G,VertexType u){
  k=LocateVex(G,u);                //确定起始顶点u的位置序号
  for(j=0;j<G.vexnum;j++)          //初始化最短边权值和依附顶点值
    if (j!=k) closedge[j]={u ,G.arcs[k][j].adj};
  closedge[k].lowcost=0;                   //选定起点u到U中
  for(i=1;i<G.vexnum;i++) {         //依次加入n-1个顶点、n-1条边
    k=minimum(closedge);            //选择下一个最短边对应的顶点序号
    printf(closedge[k].adjvex,G.vexs[k]) //输出生成树边
    closedge[k].lowcost=0;                //顶点k加到U中
    for(j=0;j<G.vexnum;j++)
         if (G.arcs[k][j].adj<closedge[j].lowcost)
            closedge[j]={G.vexs[k], G.arcs[k][j].adj};
                                   //替换最小权值和依附顶点
}
```

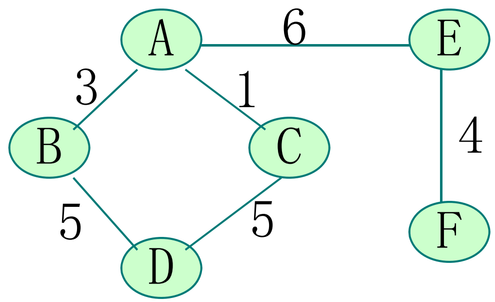
算法适合边稠密的无向连通网，$T(n,e)=O(n^2)$

2.克鲁斯卡尔（Kruskai)算法，以选边为主

需要将边按递增次序排列以供选择。

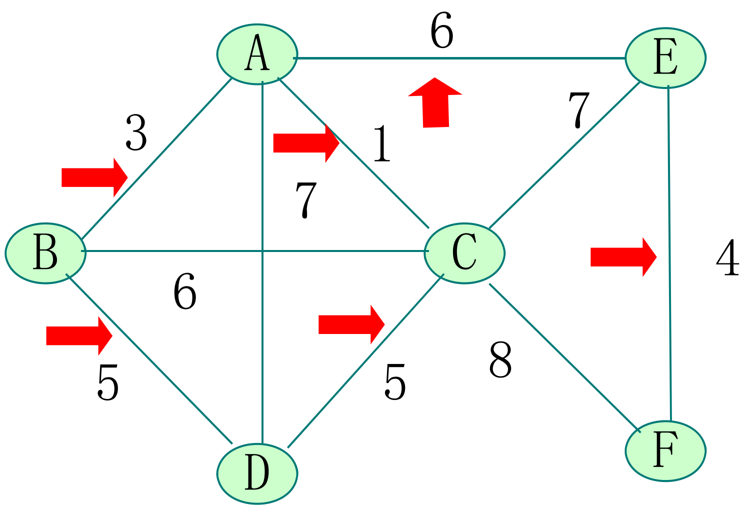


网G

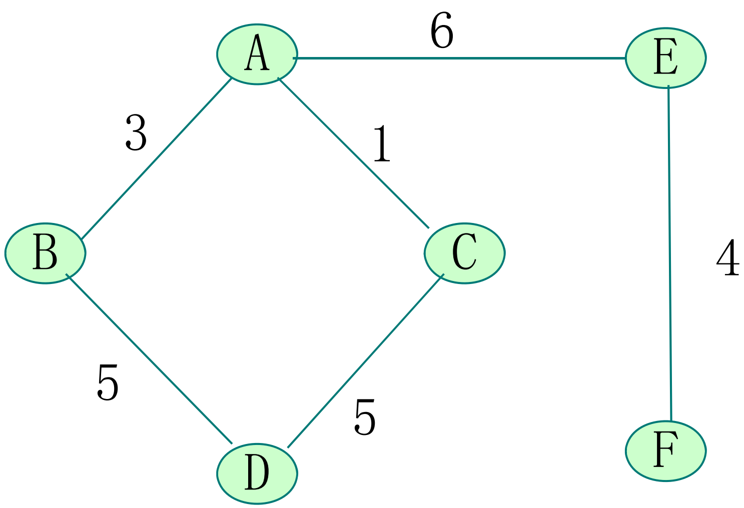


最小生成树T

克鲁斯卡尔（Kruskai)算法的另一最小生成树



网G



最小生成树T

算法适合边稠密的无向连通网，T(n,e)=O(e log e)

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。

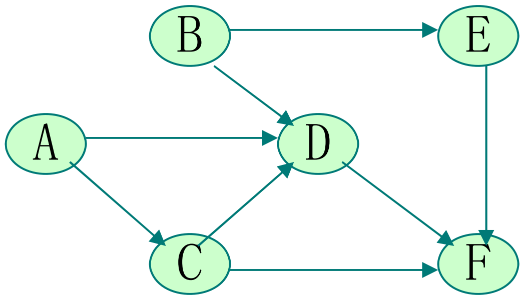


图1

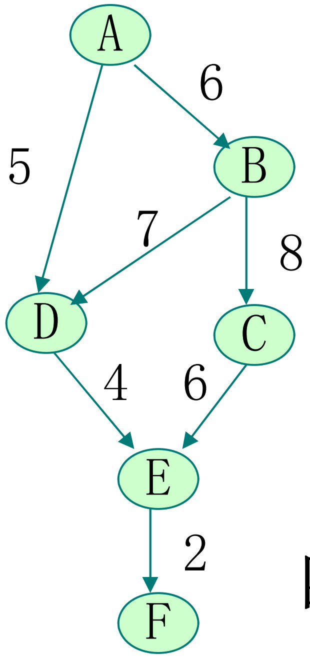


图2

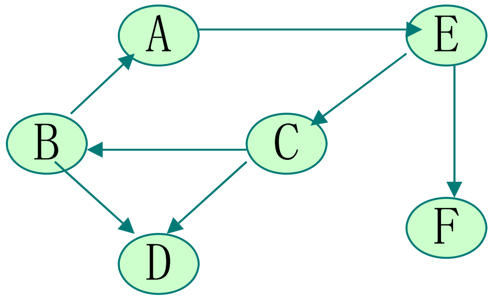
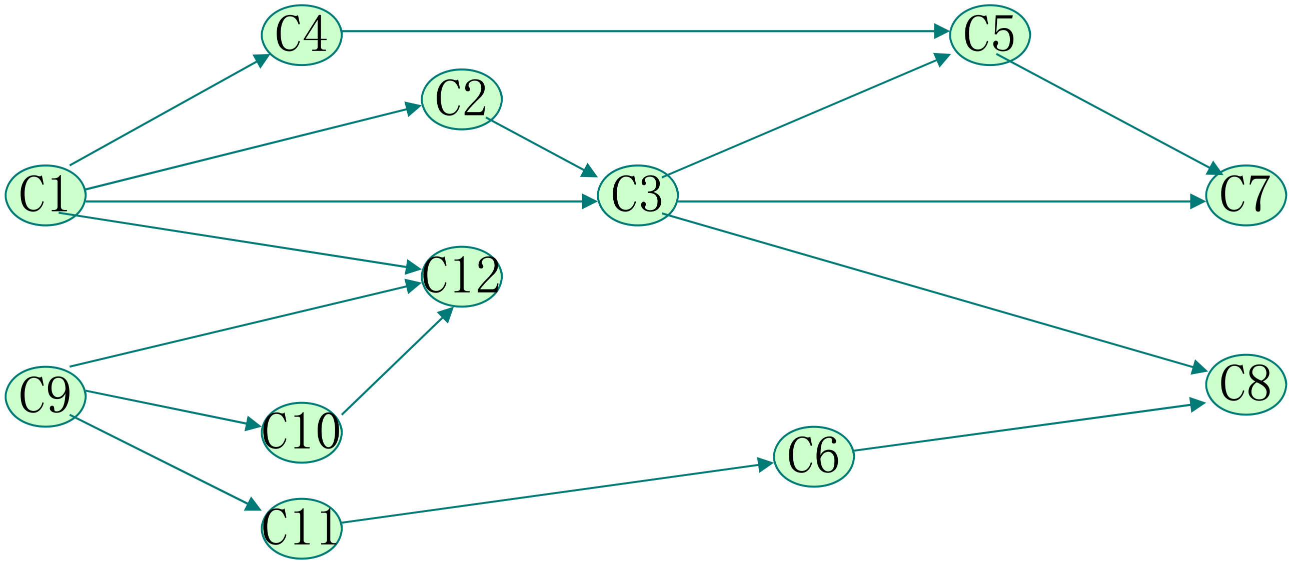


图3（非DAG）

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。

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：

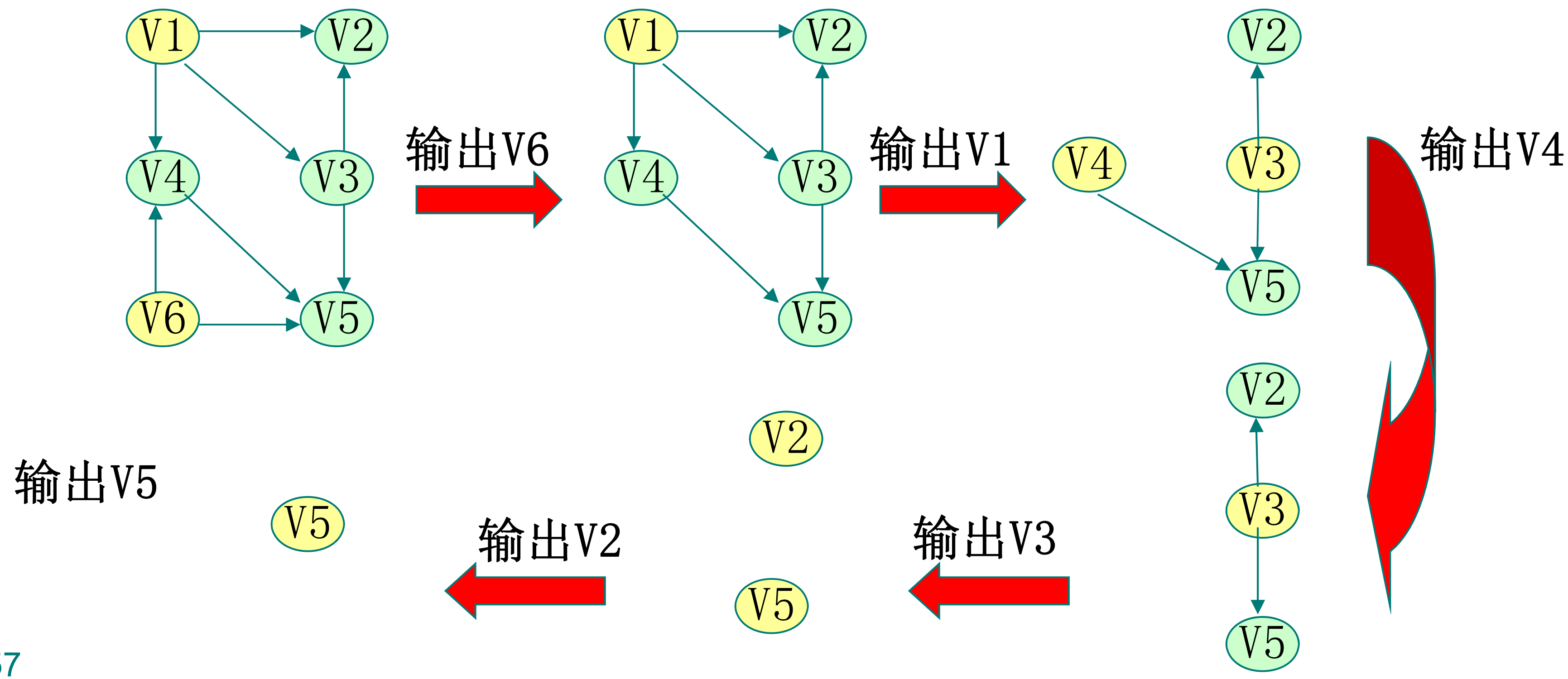
(C1, C2, C3, C4, C5, C7, C9, C10, C11, C6, C12, C8)

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

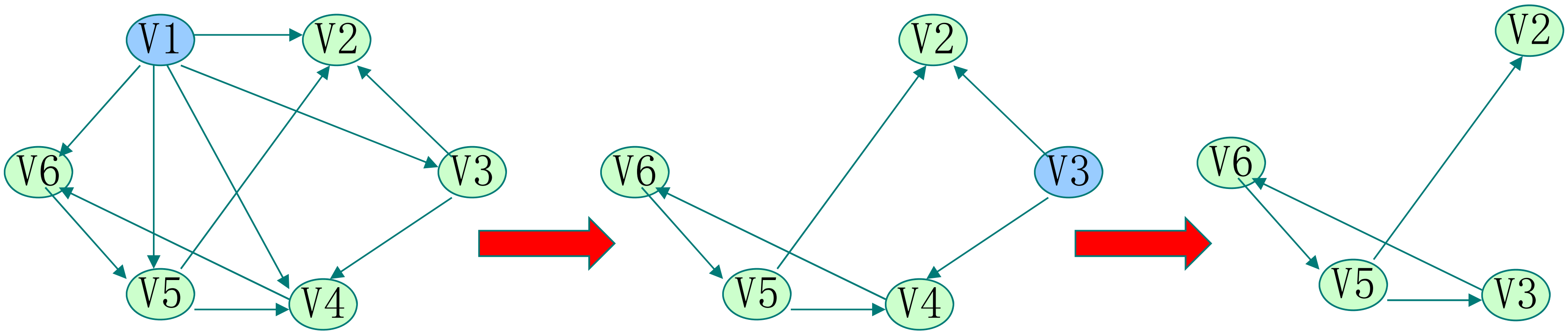
拓扑排序算法思想：重复下列操作，直到所有顶点输出完。

（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

拓扑排序算法代码（物理结构是邻接表）：

```
Status ToplogicalSort(ALGraph G){
 CountInDegree(G,indegree);  //统计顶点入度到indegree[0..G.vexnum-1]
 InitStack(S); count=0;      //初始化栈和访问顶点计数
 for(i=0;i<G.vexnum;i++)     //入度为0的顶点序号进栈
    if (!indegree[i]) Push(S,i);
 while(!StackEmpty(S)){
    Pop(S,i); printf(G.vertices[i].data);count++;
    for(p=G.vertices[i].firstarc; p; p=p->nextarc ) {
        j=p->adjvex;                     //取弧头顶点序号赋值给j
        if (--indegree[j]) Push(S,j)   //入度减一后为0，进栈
        }
    }
 if (count<G.vexnum) return ERROR;  //有回路;
 else                 return OK;
}
```

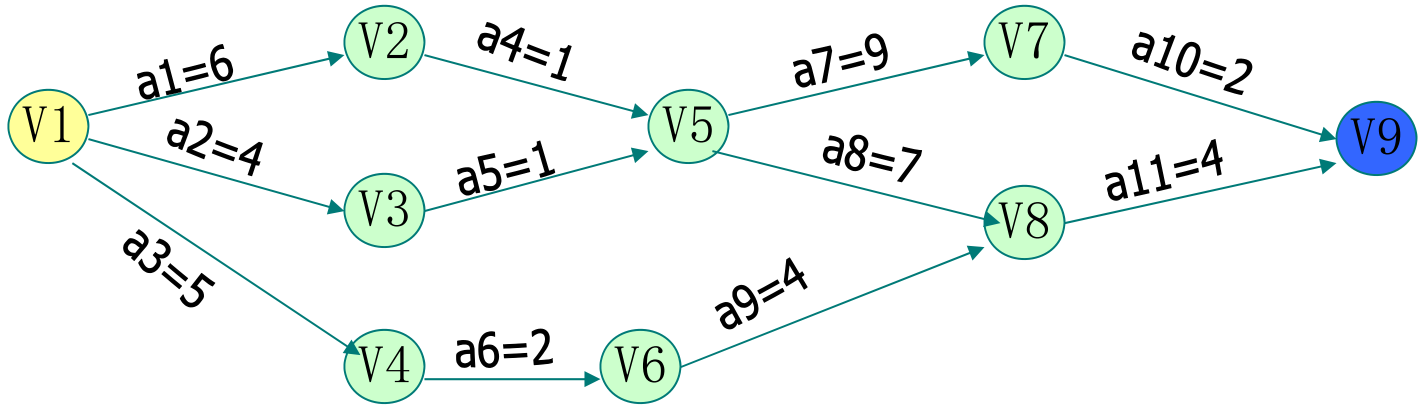
T(n)=O(n+e)

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。

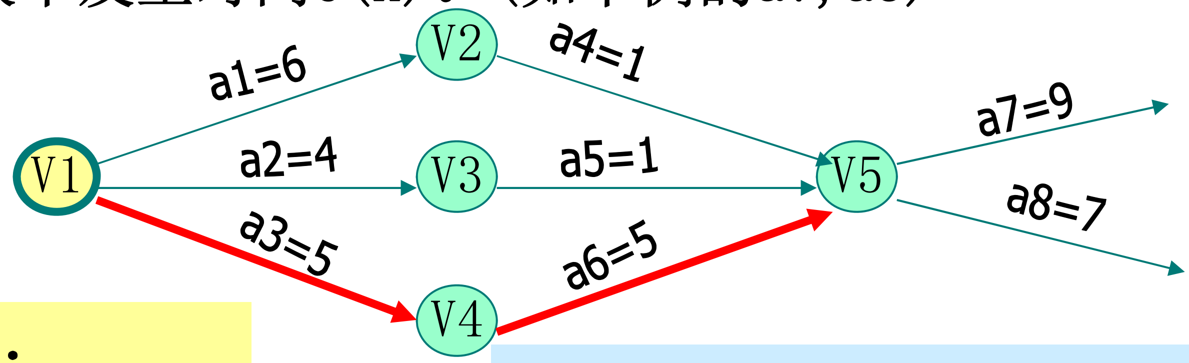
在AOE网中，部分活动可并行进行，所以完成工程的最短时间是从开始点到完成点的最长路径长度（这里是指路径上的权值之和具有最大值）。

路径长度最长的路径称为关键路径（Critical Path）。

（顶点）事件v_i的最早发生时间ve(i)：

从开始点到v_i的最长路径长度。（ve(v1)=0）

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



仅有一个前驱顶点：

$ve(v_2)=ve(v_1)+6=0+6=6$

$ve(v_3)=ve(v_1)+4=0+6=4$

$ve(v_4)=ve(v_1)+6=0+5=5$

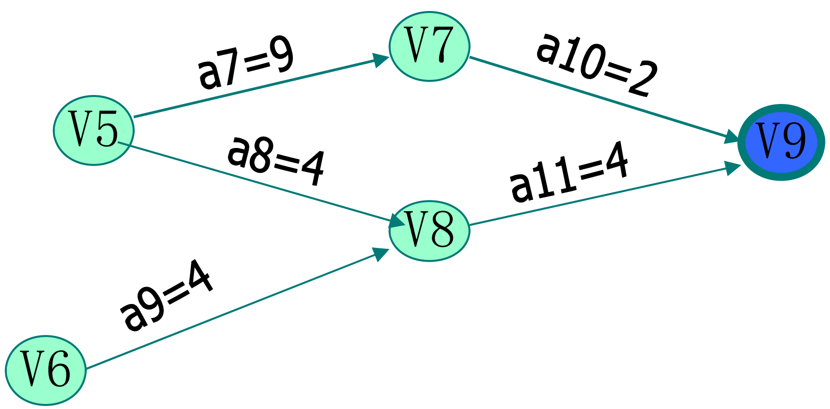
有多个前驱顶点：

$ve(v_5)$=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。

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(i)：　完成点（汇点）v_n的的最早发生时间ve(n)减去vk到vn的最长路径长度。（v_n的的最早发生时间ve(n)等于最迟开始时间vl(n)）。



仅有一个后继顶点：

假定工程18天完成(ve(v9)=18)，则：

vl(v9)=18

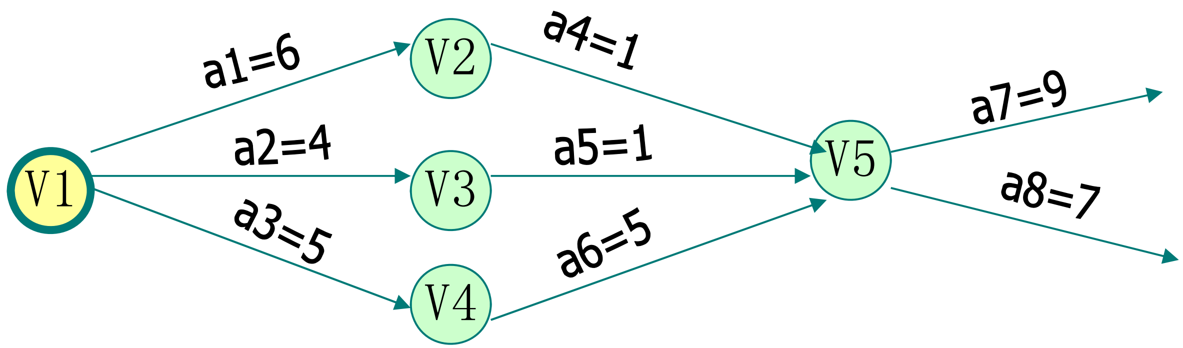
vl(v7)= vl(v9)-2=16

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

vl(v6)= vl(v8)-4=10

有多个后继顶点：

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



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

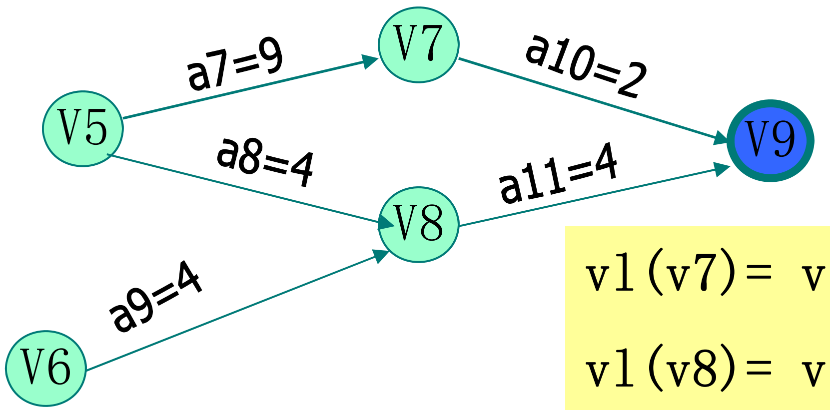
e(a5)=ve(v3)=4

e(a6)=ve(v4)=5

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

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。

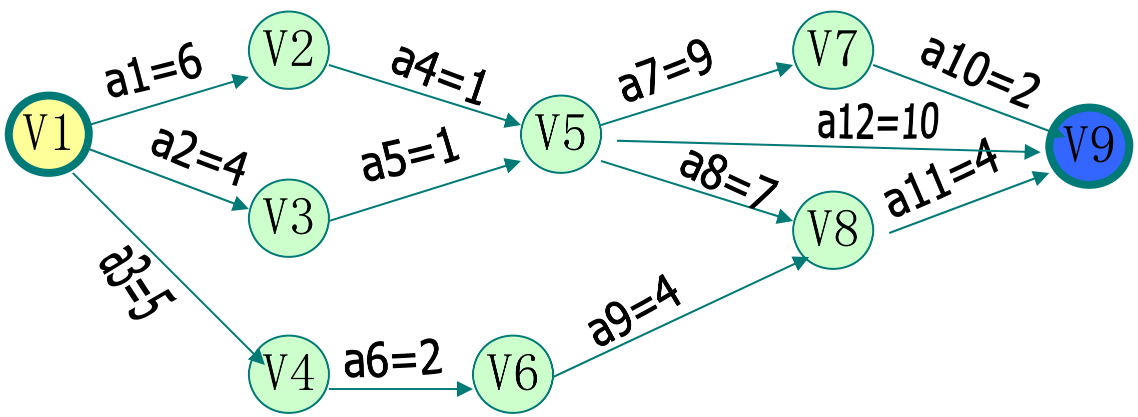
关键路径算法步骤:

（1）初始化各顶点最早发生时间为0。从开始点v1出发，按拓朴排序序列求其它各顶点的最早发生时间

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

(vi为以顶点vj为弧头的所有弧的弧尾对应的顶点)

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

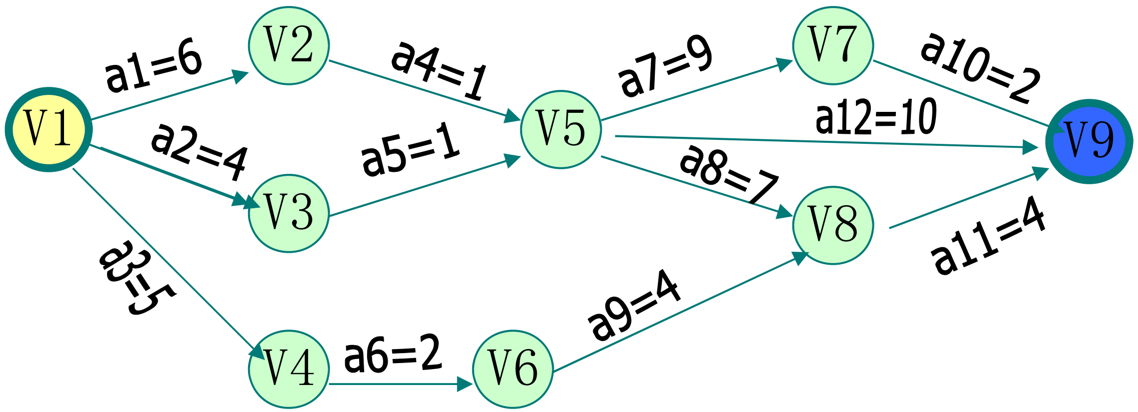


关键路径算法步骤：

（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)

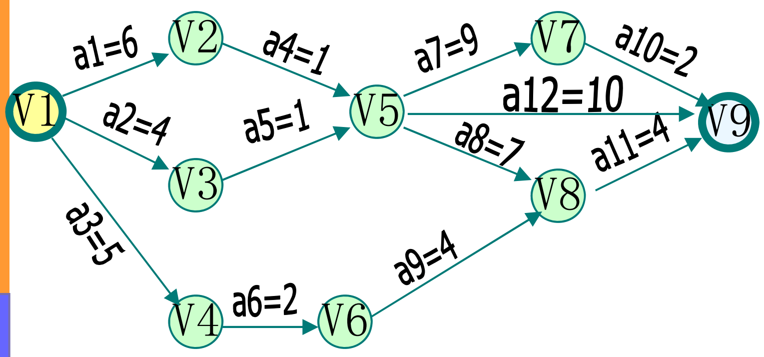


顶点	ve(i)	vl(i)
v_1	0	18/10
v_2	6	18/16
v_3	4	18/16
v_4	5	18/18
v_5	7	18/17
v_6	7	18/10
v_7	16	18/16
v_8	14	18/14
v_9	18	18

关键路径算法步骤：

（3）求每一项活动ai(vj, vk)：

e(i)=ve(j)　　　l(i)=vl(k)-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

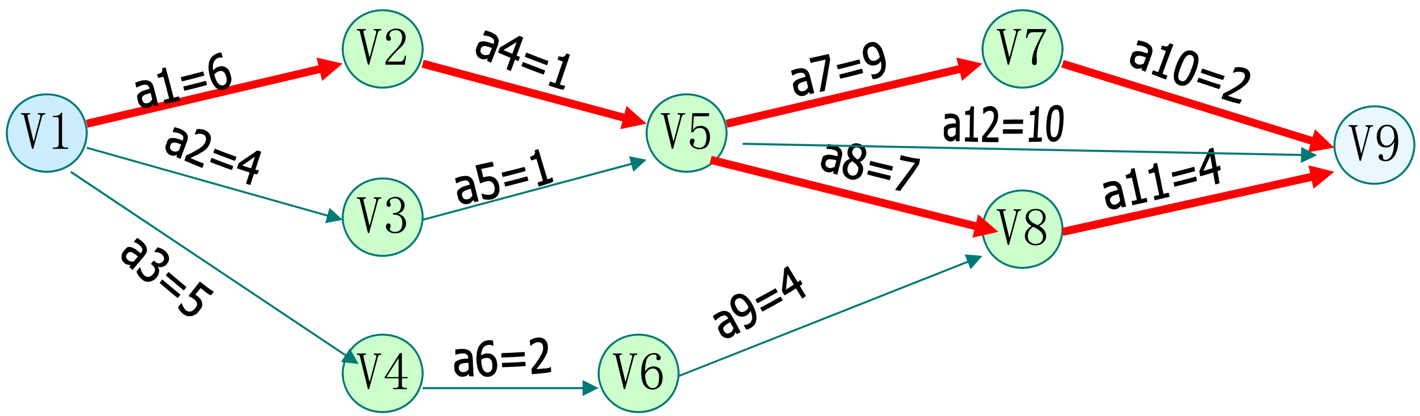
活动	e(i)	l(i)	l(i)-e(i)
a_1	0	0	0
a_2	0	2	2
a_3	0	3	3
a_4	6	6	0
a_5	4	6	2
a_6	5	8	3
a_7	7	7	0
a_8	7	7	0
a_9	7	10	3
a_{10}	16	16	0
a_{11}	14	14	0
a_{12}	7	8	1

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



利用拓扑排序算法计算各顶点最早发生时间代码（拓扑排序结果放在栈T中）：

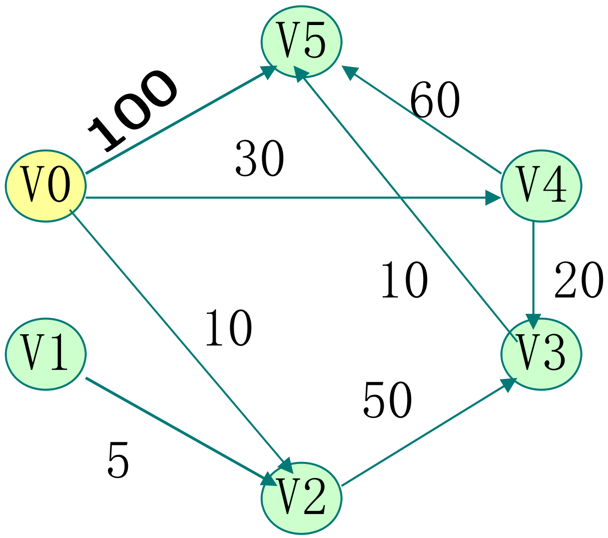
```
Status ToplogicalOrder(ALGraph G,Stack &T){
 CountInDegree(G,indegree);  //统计顶点入度到indegree[0..G.vexnum-1]
 InitStack(S); count=0;      //初始化栈和访问顶点计数
 ve[0...G.vexnum]=0;         //初始化各顶点的最早开始时间
 for(i=0;i<G.vexnum;i++)     //入度为0的顶点序号进栈
    if (!indegree[i]) Push(S,i);
 while(!StackEmpty(S)){
    Pop(S,i); Push(T,i);count++;      //i入栈T，count计数访问过的顶点
    for(p=G.vertices[i].firstarc; p; p=p->nextarc ) {
        j=p->adjvex;                        //取弧头顶点序号赋值给j
        if (--indegree[j]) Push(S,j)        //入度减一后为0，进栈
        if (ve[i]+dut(<i,j>))>ve[j])
            ve[i]=ve[i]+dut(<i,j>);         //用较大值替换替换
        }
    }
                  if (count<G.vexnum) return ERROR;          //有回路;
                  else                return OK;
}
```

计算关键活动代码（拓扑排序结果在栈T中，退栈完成逆拓扑排序）：

```
Status CriticalPath(ALGraph G){
 if (!TopologicalOrde(G,T)) return ERROR;
 vl[0...G.vexnum-1]=ve[G.vexnum-1];          //初始化顶点最迟发生时间
 while(!StackEmpty(T)){                      //逆拓扑排序求顶点最迟发生时间
    for(Pop(S,i), p=G.vertices[i].firstarc; p; p=p->nextarc ) {
      j=p->adjvex;                           //取弧头顶点序号赋值给j
      if (vl[j]-duty(<i,j>)<vl[i])      //dut(<j,k>)表示弧活动持续时间
         ve[i]=vl[j]-duty(<i,j>);                    //用较小值替换
      }
for(i=0;i<G.vexnum;i++)  //按顶点次序，取出该顶点作为弧尾的各条弧分析
    for(p=G.vertices[i].firstarc; p; p=p->nextarc ) {
      j=p->adjvex;                                  //准备分析弧<i,j>
      ee=ve[i]; el=vl[j]-duty(<i,j>); //计算弧<i,j>最早、最迟开始时间
      if (ee=el) printf(<i,j>,duty(<i,j>),ee,el)       //输出关键活动
      }
 return OK; }
```

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路径长度递增法：

$$
\begin{array}{c|c}
 & \begin{array}{ccccc} \mathbf{0} & \mathbf{1} & \mathbf{\cdots\cdots} & \mathbf{n-2} & \mathbf{n-1} \end{array} \\
\begin{array}{c} 0 \\ 1 \\ \cdots\cdots \\ n-2 \\ n-1 \end{array} &
\left(\begin{array}{ccccc}
a_{00} & a_{01} & \cdots\ \cdots & a_{0n-2} & a_{0n-1} \\
a_{10} & a_{11} & \cdots\ \cdots & a_{1n-2} & a_{1n-1} \\
\cdots & \cdots & \cdots\ \cdots\ \cdots\ \cdots & \cdots & \cdots \\
a_{n-20} & a_{n-21} & \cdots\ \cdots & a_{n-2n-2} & a_{n-2n-1} \\
a_{n-10} & a_{n-11} & \cdots\ \cdots & a_{n-1n-2} & a_{n-1n-1}
\end{array}\right)
\end{array}
$$

初始化

	0	1	……	n−2	n−1
最短路径数组D：	a_{00}	a_{01}	…	a_{0n-2}	a_{0n-1}
最短路径的前驱顶点数组：	V0	V0	…	V0	V0
Final：	F	F	…	F	F

Dijkstra路径长度递增法：

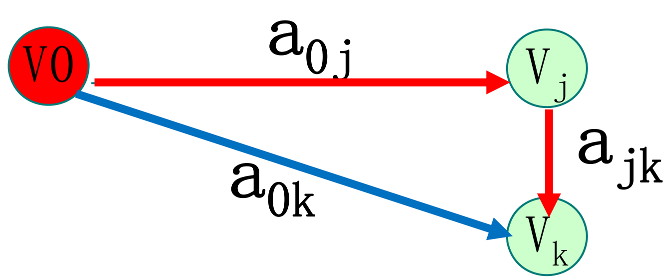
如果**j**满足：

$$D[j]=\min\{D[i] \mid v_i属于V\}$$

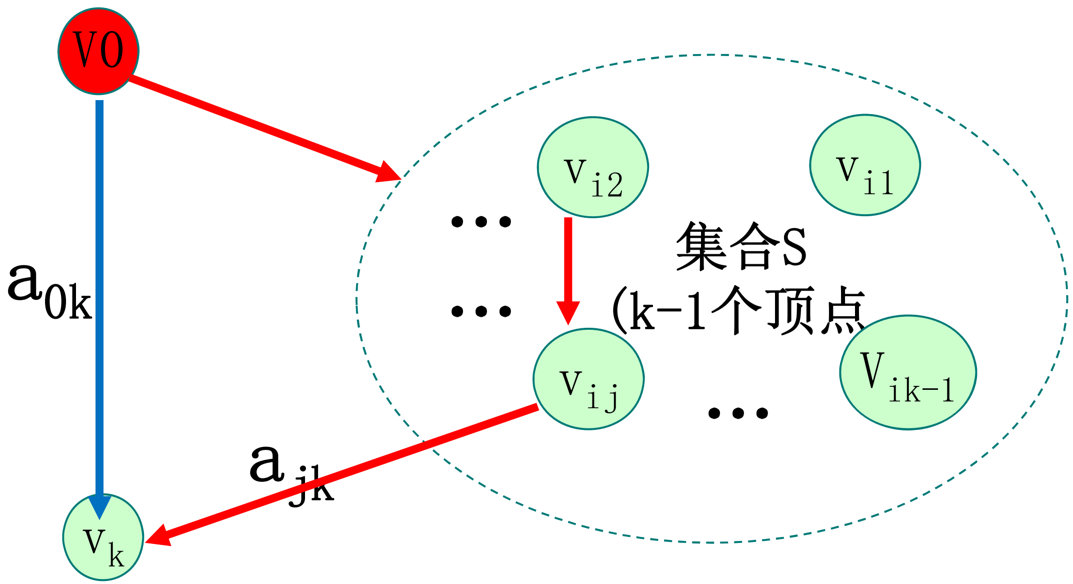
则路径（ v_0 , v_j）是由v_0出发的长度最短的一条最短路径。

如何求由v_0出发的长度次短的一条最短路径。假定终点为v_k。

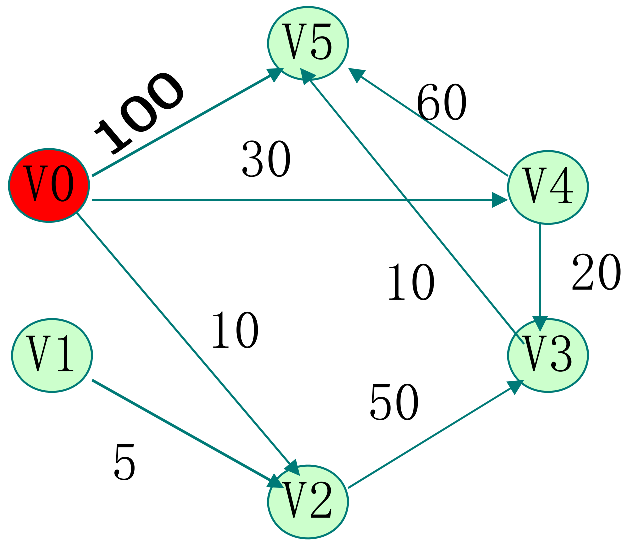
取这两条路径中的路径长度较小者。



假设S为已经求得最短路径的终点的顶点集合，假定下一条最短路径的终点是$\mathbf{v_k}$，



Dijkstra路径长度递增法：

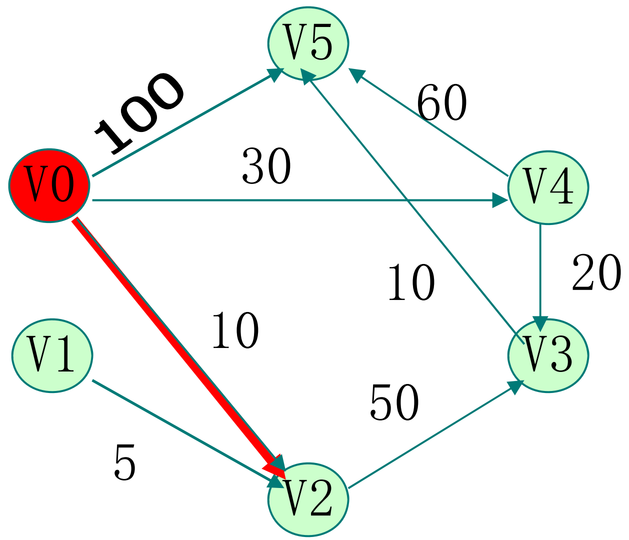


	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞	∞	∞	∞	∞	∞

初始化

	0	1	2	3	4	5
最短路径数组D：	∞	∞	10	∞	30	100
最短路径的前驱顶点数组：	V0	V0	V0	V0	V0	V0
Final：	T	F	F	F	F	F

Dijkstra路径长度递增法：



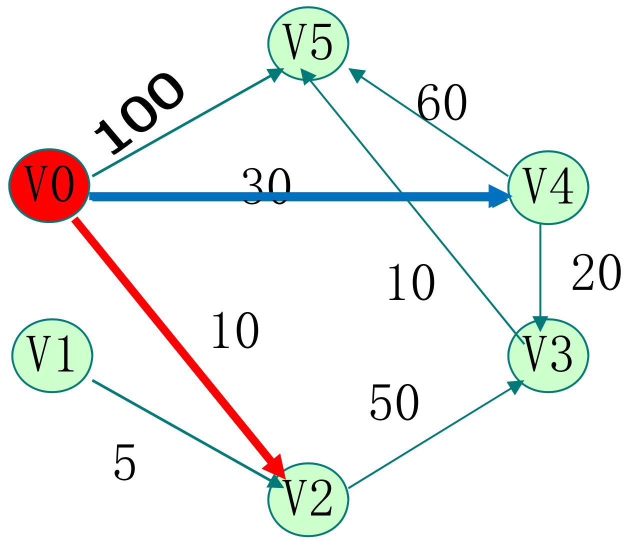
$$
\begin{array}{c|c}
 & \begin{array}{cccccc} 0 & 1 & 2 & 3 & 4 & 5 \end{array} \\
\begin{array}{c} 0 \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} &
\begin{pmatrix}
\infty & \infty & 10 & \infty & 30 & 100 \\
\infty & \infty & 5 & \infty & \infty & \infty \\
\infty & \boxed{\infty} & \infty & 50 & \infty & \infty \\
\infty & \infty & \infty & \infty & \infty & 10 \\
\infty & & \infty & 20 & \infty & 60 \\
\infty & & \infty & \infty & \infty & \infty
\end{pmatrix}
\end{array}
$$

+10

比较大小

	0	1	2	3	4	5
最短路径数组D：	∞	∞	10	60	30	100
最短路径的前驱顶点数组：	V0	V0	V0	V2	V0	V0
Final：	T	F	T	F	F	F

Dijkstra路径长度递增法：



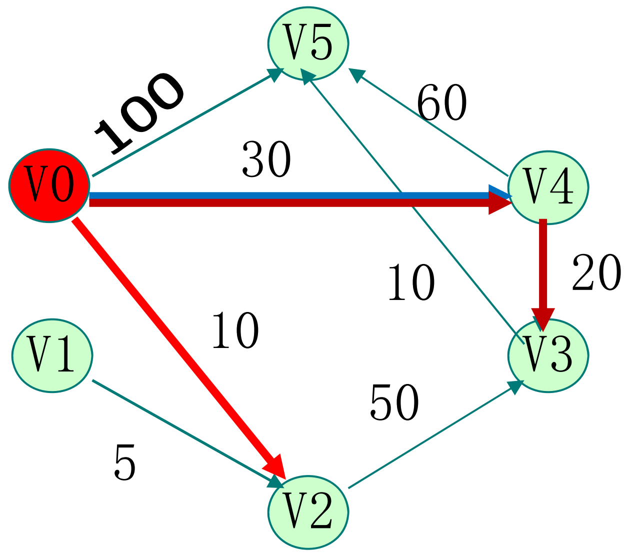
$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & \infty & \infty & 10 & \infty & 30 & 100 \\
1 & \infty & \infty & 5 & \infty & \infty & \infty \\
2 & \infty & \infty & \infty & 50 & \infty & \infty \\
3 & \infty & \infty & \infty & \infty & \infty & 10 \\
4 & \infty & \infty & \infty & 20 & \infty & 60 \\
5 & \infty & \infty & \infty & \infty & \infty & \infty
\end{array}
$$

+30

比较大小

	0	1	2	3	4	5
最短路径数组D：	∞	∞	10	60	30	100 90
最短路径的前驱顶点数组：	V0	V0	V0	V2 V4	V0	V0 V4
Final：	T	F	T	F	T F	F

Dijkstra路径长度递增法：



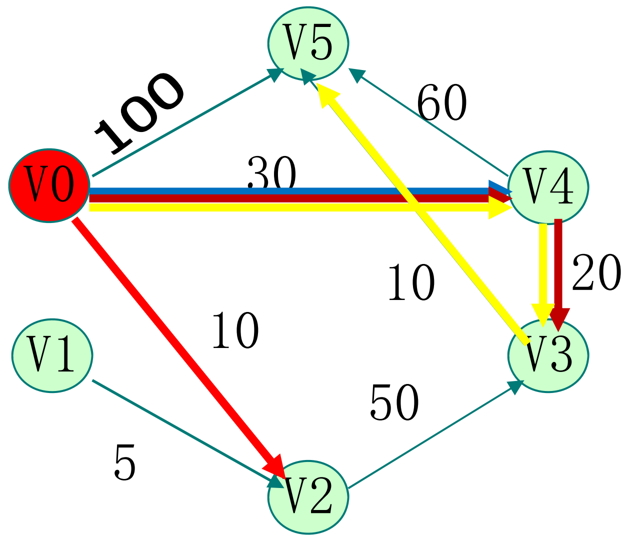
	0	1	2	3	4	5
0	∞	∞	10	∞	30	100
1	∞	∞	5	∞	∞	∞
2	∞	∞	∞	50	∞	∞
3	∞	∞	∞	∞	∞	10
4	∞	∞	∞	20	∞	60
5	∞		∞	∞	∞	∞

+50

比较大小

	0	1	2	3	4	5
最短路径数组D：	∞	∞	10	50	30	60
最短路径的前驱顶点数组：	V0	V0	V0	V4	V0	V3
Final：	T	F	T	T	T	F

Dijkstra路径长度递增法：



$$
\begin{array}{c|cccccc}
 & 0 & 1 & 2 & 3 & 4 & 5 \\
\hline
0 & \infty & \infty & 10 & \infty & 30 & 100 \\
1 & \infty & \infty & 5 & \infty & \infty & \infty \\
2 & \infty & \infty & \infty & 50 & \infty & \infty \\
3 & \infty & \infty & \infty & \infty & \infty & 10 \\
4 & \infty & \infty & \infty & 20 & \infty & 60 \\
5 & \infty & \infty & \infty & \infty & \infty & \infty
\end{array}
$$

+60

	0	1	2	3	4	5
最短路径数组D：	∞	∞	10	50	30	60
最短路径的前驱顶点数组：	V0	V0	V0	V4	V0	V3
Final：	T	F	T	T	T	F

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

始点	终点	长度	前驱顶点序列	最短路径顶点序列
v0	v1	∞	无	无
v0	v2	10	V2←V0	V0→V2
v0	v3	50	V3←V4 ←V0	V0→V4→V3
v0	v4	30	V4←V0	V0→V4
V0	v5	60	V5←V3 ←V4 ←V0	V0→V4→V3→V5

时间复杂度：$O(n^2)$

迪杰斯特拉算法代码：

```
void ShortPath1(MGraph G,int v0,PathMatrix &P,ShorPath &D){
  for(i=0;i<G.vexnum;i++) { //初始化,源点序号为v0
    final[i]=false;
         P[i]=0;      //前驱顶点序号
         D[i]=G.arcs[v0][i];}
    }
  D[v0]=0; final[v0]=true;
  for(i=1;i<G.vexnum;i++)  { //处理剩下的n-1个顶点
    k=minmum(D);    //查找满足P[k]为false且D[k]具有最小的下标k
    final[k]=true;  //确定顶点序号k的最短路径
    for(j=0;j<G.vexnum;j++)
      if (! final[j] && D[j]>G.arcs[v0][k]+G.arcs[k][j])  {
         D[j]=G.arcs[0][k]+G.arcs[k][j];  //修改路径长度
         P[j]=k;                          //修改前驱顶点编号
         }
     }
  }
```

$T(n)=O(n^2)$

7.6.1 每一对顶点之间的最短路径

算法1：迪杰斯特拉（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：弗洛伊德（Floyd）算法：

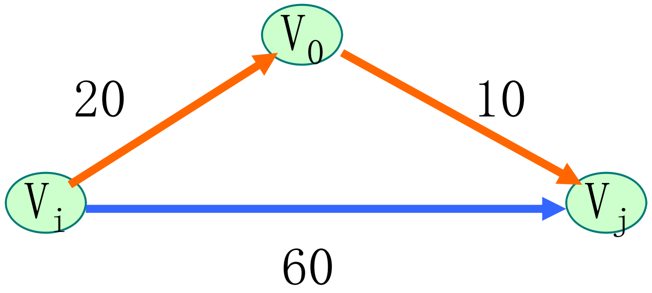
算法思想：

假设求V_i到V_j的最短路径，如果从V_i到V_j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。

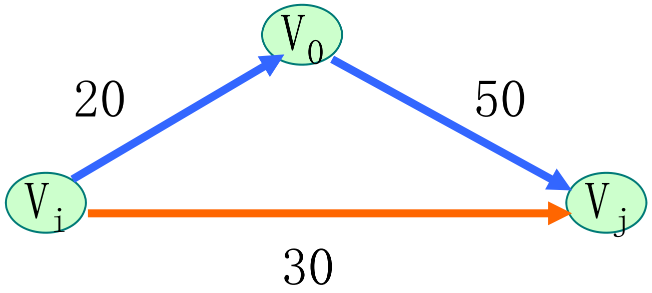
弗洛伊德（Floyd）算法：

首先考虑(V_i, V_0, V_j)是否存在（即判断(V_i, V_0)和(V_0, V_j)是否存在)，如果存在，比较(V_i, V_j)和$(V_i, V_0)+(V_0, V_j)$，取长度较短的值。

得到了从V_i到V_j的中间顶点序号不大于0的路径长度。



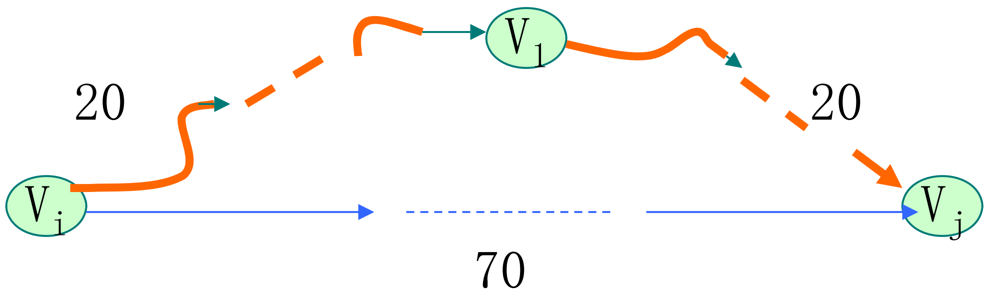
图G1



图G2

再考虑路径上再增加一个顶点V_1，如果考虑$(V_i, \ldots V_1)$和$(V_1, \ldots. V_j)$，$(V_i, \ldots V_1)$和$(V_1, \ldots. V_j)$都是中间顶点序号不大于1的最短路径。$(V_i, \ldots V_1, \ldots. V_j)$可能是从$V_i$到$V_j$的中间顶点序号不大于1的最短路径。

比较V_i到V_j的中间顶点序号不大于0的最短路径和$(V_i, \ldots V_1)+(V_1, \ldots. V_j)$，取长度较短的，得到了从$V_i$到$V_j$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:

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

D^{-1} = G.arcs

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

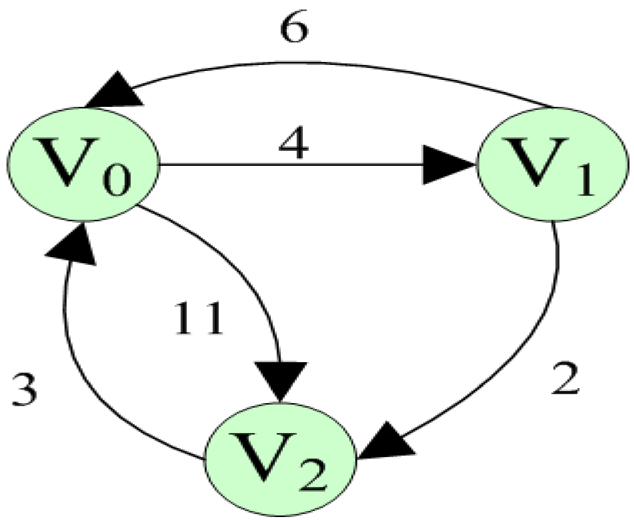
计算最短路径算法:　　0≤k≤n-1　　n=G.vexnum

```
for(k=0;k<G.vexnum;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 (D[i][j]>D[i][k]+D[k][j])
         D[i][j]=D[i][k]+D[k][j]; //取较短路径
```

Floyd算法代码：

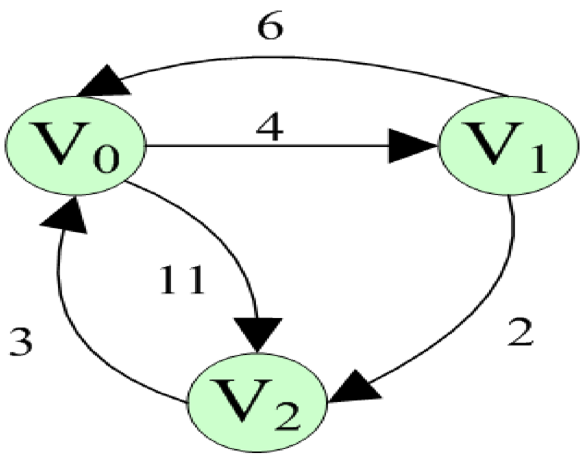
```
void ShortPath2(MGraph G, PathMatrix &P[],ShorPath &D) {
 for(i=0;i<G.vexnum;i++)
    for(j=0;j<G.vexnum;j++){          //初始化
        P[i][j]=-1;                //-1表示无中间顶点
        D[i][j]=G.arcs[i][j];}
 for(k=0;k<G.vexnum;k++)               //依次选定中间顶点V0,V1,…Vn-1
   for(i=0; i<G.vexnum;i++)            //i,j配合处理所有顶点Vi,Vj
     for(j=0; j<G.vexnum;j++)
       if (D[i][j]>D[i][k]+D[k][j]) {
           D[i][j]=D[i][k]+D[k][j]; //取较短路径
           P[i][j]=k;               //Vi到Vj的中间顶点Vk
         }
   }
```

$T(n)=O(n^3)$



$$
\begin{pmatrix}
0 & 4 & 11 \\
6 & 0 & 2 \\
3 & \infty & 0
\end{pmatrix}
$$

邻接矩阵

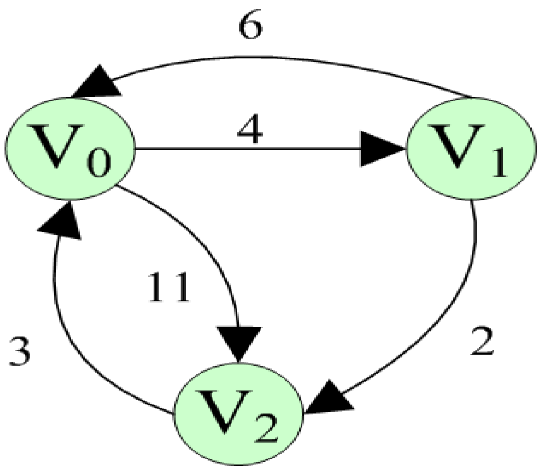


$$D_{-1}=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & \infty & 0\end{pmatrix} \quad P_{-1}=\begin{pmatrix}-1 & -1 & -1\\ -1 & -1 & -1\\ -1 & -1 & -1\end{pmatrix}$$

$$D_{0}=\begin{pmatrix}0 & 4 & 11\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix} \quad P_{0}=\begin{pmatrix}-1 & -1 & -1\\ -1 & -1 & -1\\ -1 & 0 & -1\end{pmatrix}$$

$$D_{1}=\begin{pmatrix}0 & 4 & 6\\ 6 & 0 & 2\\ 3 & 7 & 0\end{pmatrix} \quad P_{1}=\begin{pmatrix}-1 & -1 & 1\\ -1 & -1 & -1\\ -1 & 0 & -1\end{pmatrix}$$

$$D_{2}=\begin{pmatrix}0 & 4 & 6\\ 5 & 0 & 2\\ 3 & 7 & 0\end{pmatrix} \quad P_{2}=\begin{pmatrix}-1 & -1 & 1\\ 2 & -1 & -1\\ -1 & 0 & -1\end{pmatrix}$$



$$D_2=\begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \qquad P_2=\begin{pmatrix} -1 & -1 & 1 \\ 2 & -1 & -1 \\ -1 & 0 & -1 \end{pmatrix}$$

$$\text{P2的含义}=\begin{pmatrix} & (0,1) & (0,1,2) \\ (1,2,0) & & (1,2) \\ (2,0) & (2,0,1) & \end{pmatrix}$$

这里序号为顶点下标，如（0，1，2）表示v0、v1、v2

本章小结

介绍了图的逻辑结构、基本运算、物理结构以及基本运算的实现算法和效率分析。需要重点掌握的内容是：

① 图的概念及术语；
② 图的物理结构；
③ 图的遍历算法；
④ 最小生成树；
⑤ 拓扑排序；

需要了解的内容：关键路径、最短路径等算法。