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 最短路径

7.4 图的连通性问题

Prim 算法的基本描述

设G＝(V，E)是个带权无向连通图，U是最小生成树的顶点集合，T是最小生成树的边集合：

{ 从G中任选u∈V;

U＝{u}; T = φ;

while (U != V)

{ 在u∈U, v∈V-U中找一条权值最小的边(u,v);

U = U + {v};

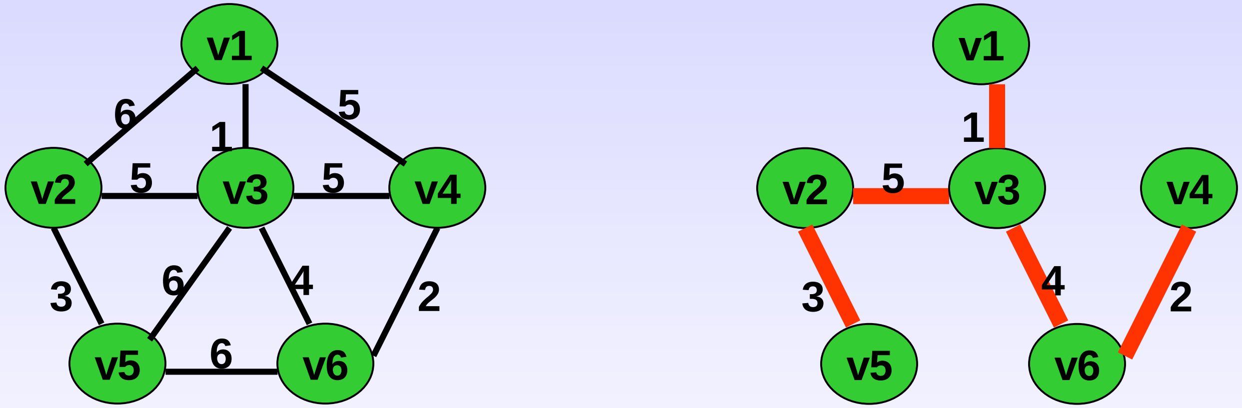
T = T + (u,v);

}

}

7.4 图的连通性问题

- Prim 算法的实例



要点：每当新的顶点并入 U 之后，则调整仍在V-U 集合中的顶点至 U 中顶点的最小距离。

7.4 图的连通性问题

```
typedef struct{  VertexType      vexs[MAX];
                 int             Edge[MAX][MAX];
                 int             vexnum, arcnum;
} MGraph;
struct{  VertexType  adjvex;
         int          lowcost;
} closedge[ MAX ];
void MST_Prim( MGraph G, VertexType u )
{  k = LocateVex( G, u );                  // 求顶点u的序号
   for( j=1; j<=G.vexnum; j++ )            // 初始化辅助数组closedge
     if( j!=k )  closedge[ j ] = { u, G.Edge[ k ][ j ] };
   closedge[ k ].lowcost = 0;              // 初始时U={u}
```

7.4 图的连通性问题

```
for( i=1; i<G.vexnum; i++)
{   min = ∞;  k = 0;                    // 查找U到V-U中的权值最小的边
    for( j=1; j<=G.vexnum; j++ )
      if( closedge[ j ].lowcost>0 && closedge[ j ].lowcost < min )
      {   k = j; min = closedge[ j ].lowcost; }
    printf(closedge[ k ]. adjvex, G.vexs[ k ]);  // 输出该边
    closedge[ k ].lowcost = 0;      // 顶点k并入U
    for( j= 1; j<=G.vexnum; j++ )  // 修改V-U中每个顶点到U的最小边
      if( G.Edge[ k ][ j ] < closedge[ j ].lowcost)
         closedge[ j ] = { G.vexs[ k ], G.Edge[ k ][ j ] };
  }
}
```

7.4 图的连通性问题

2 Kruscal 算法

- Kruscal算法的基本思想：

设有一个有 n 个顶点的连通网络 $N = \{ V, E \}$,最初先构造一个只有 n 个顶点，没有边的非连通图 $T = \{ V, \varnothing \}$，图中每个顶点自成一个连通分量。当在 E 中选到一条具有最小权值的边时,若该边的两个顶点落在不同的连通分量上*（没有构成回路）*，则将此边加入到 T 中；否则将此边舍去，重新选择一条权值最小的边。如此重复下去，直到所有顶点在同一个连通分量上为止。

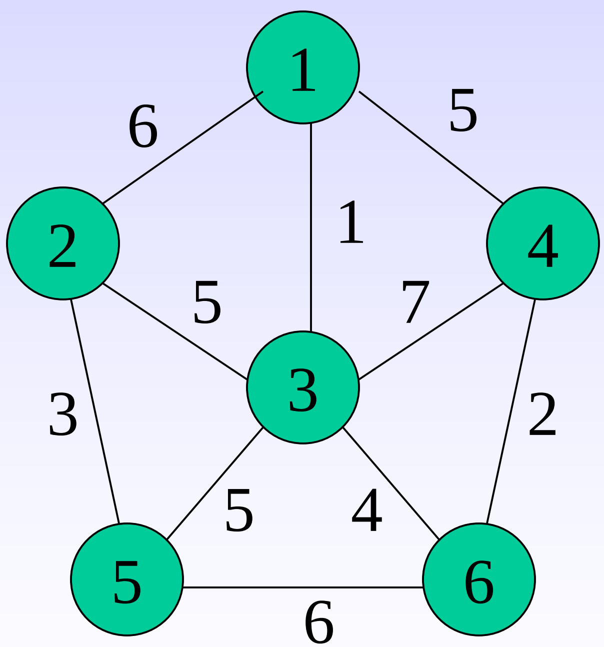
7.4 图的连通性问题

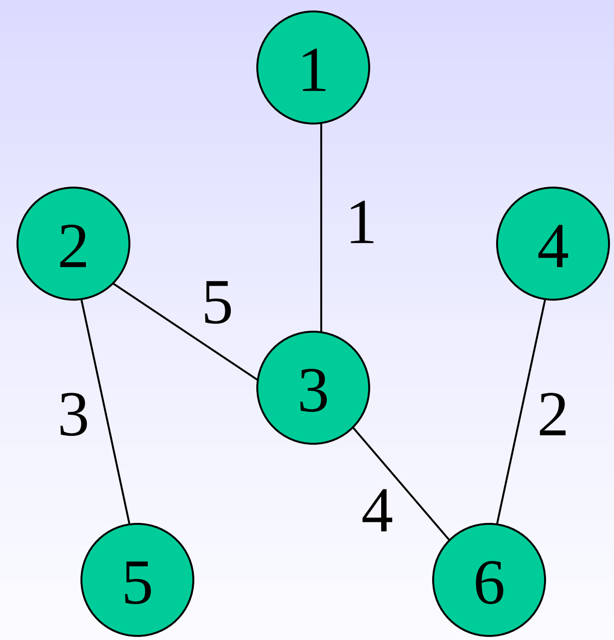
Kruscal 算法的基本描述

设G＝(V，E)是个有权无向连通图：

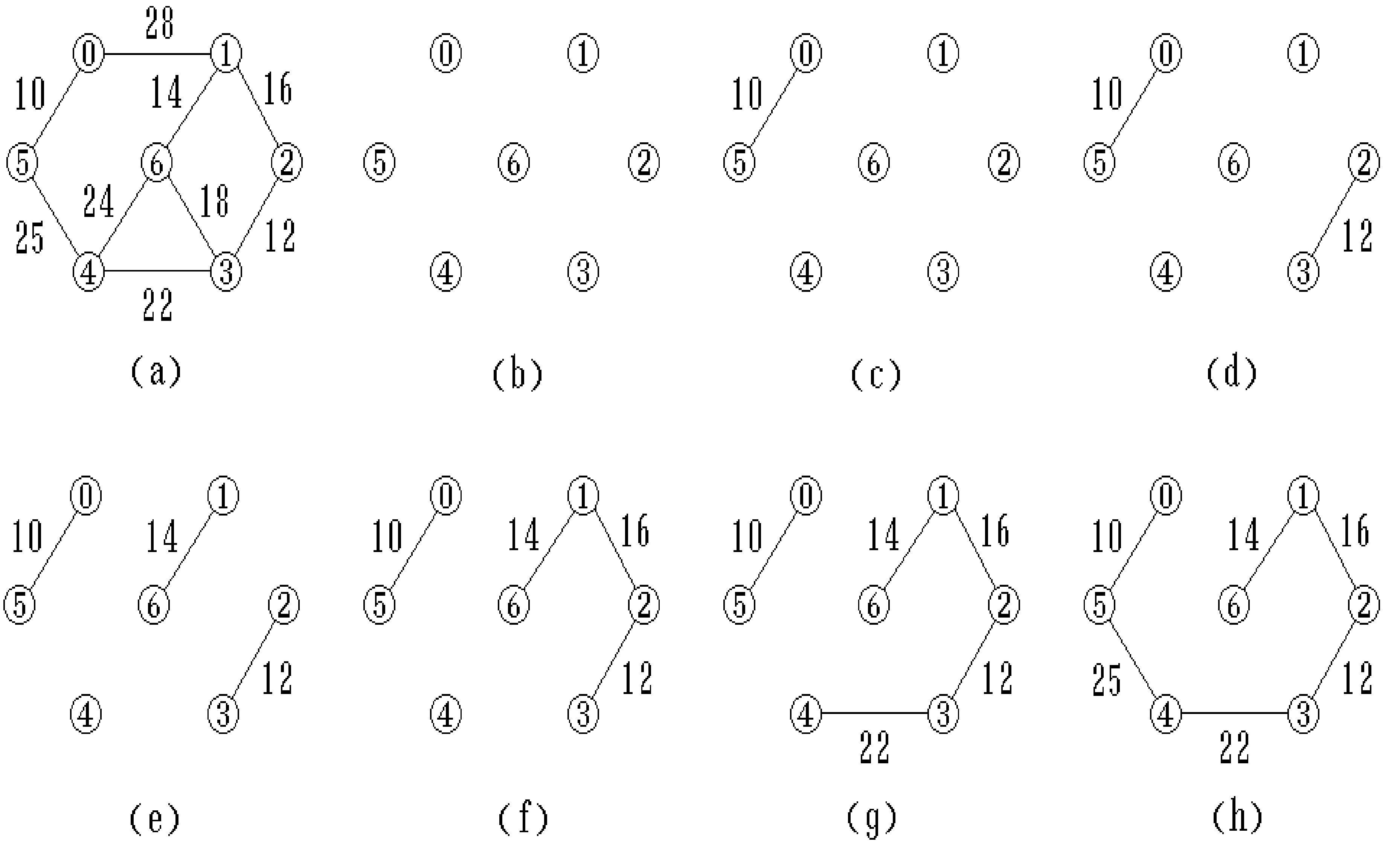
```
{  T＝{ V, ∅ };
   while( T 的边数 < n - 1)
   {  在 E 中选择代价最小的边 ( u,v );
      E＝E－ ( u,v );
      if( ( u,v )不和 T 中已有的边构成回路 )
      // 或  if( u,v)分属T的不同的连通分量 )
            T＝T + ( u,v );
   }
}
```

7.4 图的连通性问题





应用克鲁斯卡尔算法构造最小生成树的过程



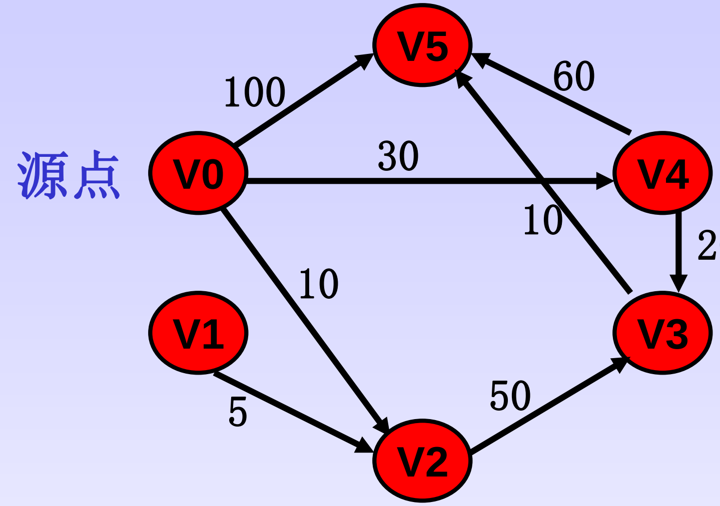
7.6 最短路径

- 路径：在图 G=(V, E) 中, 若从顶点 v_i 出发, 沿一些边经过一些顶点 v_{p1}, v_{p2}, …, v_{pm}, 到达顶点v_j。则称顶点序列 (v_i v_{p1} v_{p2} … v_{pm} v_j) 为从顶点v_i到顶点 v_j 的路径。它经过的边(v_i, v_{p1})、(v_{p1}, v_{p2})、…、(v_{pm}, v_j)应是属于E的边。
- 带权图的路径长度：是指路径上各边的权之和。
- 最短路径：从图中某一顶点u(称为源点)到达另一顶点v(称为终点)的路径中路径长度最短的一条路径称为顶点u到顶点v的最短路径。
- 最短路径问题的应用：
  - 计算机交通咨询
  - 互连网中的路由算法

7.6 最短路径

7.6.1 单源最短路径问题：

给定带权有向图G和源点v，求从源点v到G中其余各顶点的最短路径。



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

源点	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

7.6 最短路径

最短路径的性质：

设vi在u到v的最短路径上，则u到v的最短路径是u到vi和vi到v之间最短路径之和。

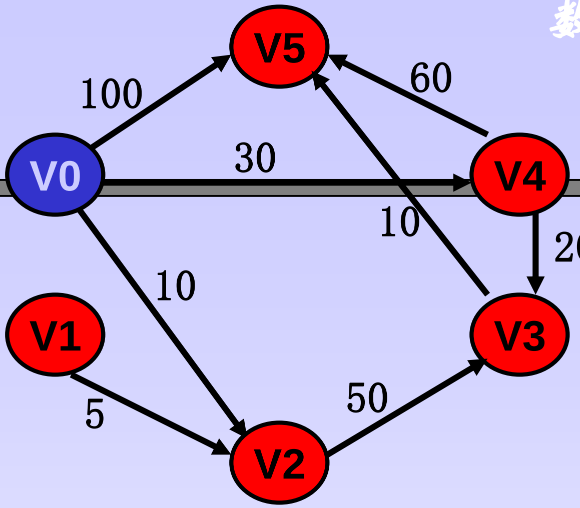
7.6 最短路径

Dijkstra 算法求单源最短路径：

设 V 是该有向图的顶点的集合， S 是已求得最短路径的顶点的集合。引进一个辅助数组D，D[i]表示当前所找到的从v0 至其余各顶点vi 的最短路径长度。

```
{   S = { v0 } ;        //初始时S中只有顶点v0
    for ( i=1; i<n; i++ )
        D[i] = Edge[0][i]; // v0至vi的边长
    for ( i=1; i<n; i++ )
    {   选择顶点vj，满足条件 D[j] = Min{ D[i] | vi∈V-S };
        S = S + vj;           //将vj 加入集合 S
        for ( 每一个顶点vk ∈V-S )
            D[k] = Min( D[k], D[j] + Edge[j][k] ) ;
    } // 注：顶点v0到顶点vk的最短路径或者是弧(v0,vk)，或者是
}   // (v0,vp1...vpm,vk)，其中vp1...vpm∈S。
```

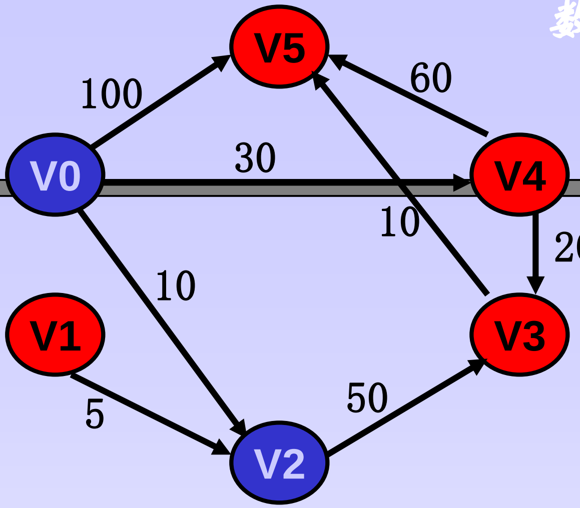
7.6 最短路径



Step0： 初始化，D[k]为源点V0到V_k的边长。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞				
v2	10 (v0,v2)				
v3	∞				
v4	30 (v0,v4)				
v5	100 (v0,v5)				
vj					
S					

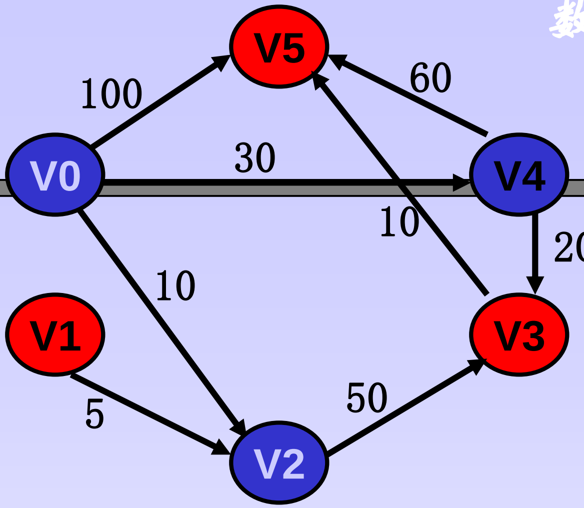
7.6 最短路径



Step1： 选择当前的最短路径**(v0,v2)**，将**v2**加入**S**。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞				
v2	10 (v0,v2)				
v3	∞				
v4	30 (v0,v4)				
v5	100 (v0,v5)				
vj	V2				
S	{v0,v2}				

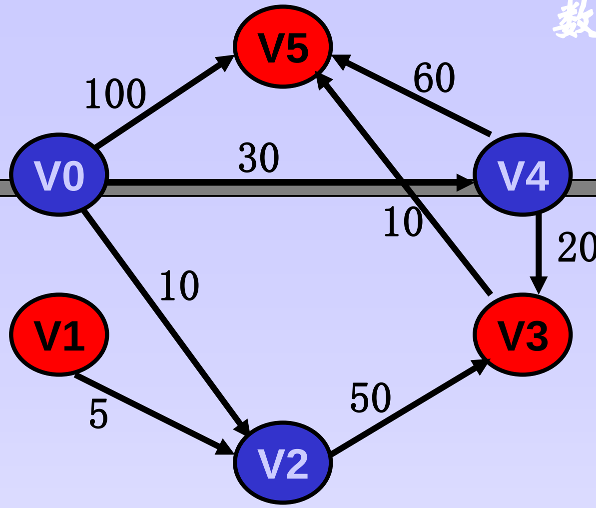
7.6 最短路径



Step2： 更新从v0到v1、v3、v4、v5的最短路径。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞			
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)			
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)			
vj	V2				
S	{v0,v2}				

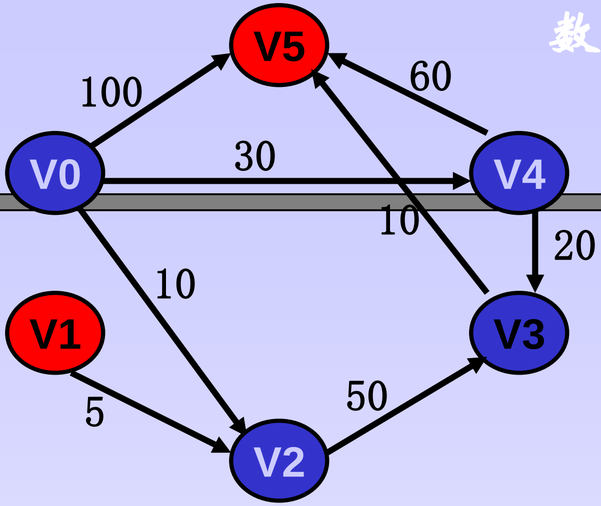
7.6 最短路径



Step3： 选择当前的最短路径**(v0,v4)**，将**v4**加入**S**。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞			
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)			
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)			
vj	V2	V4			
S	{v0,v2}	{v0,v2,v4}			

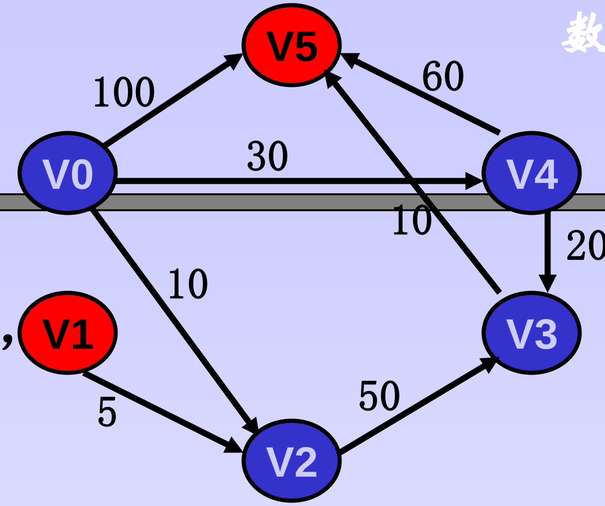
7.6 最短路径



Step4: 更新从v0到v1、v3、v5的最短路径

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞	∞		
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)	50 (v0,v4,v3)		
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)	90 (v0,v4,v5)		
vj	V2	V4			
S	{v0,v2}	{v0,v2,v4}			

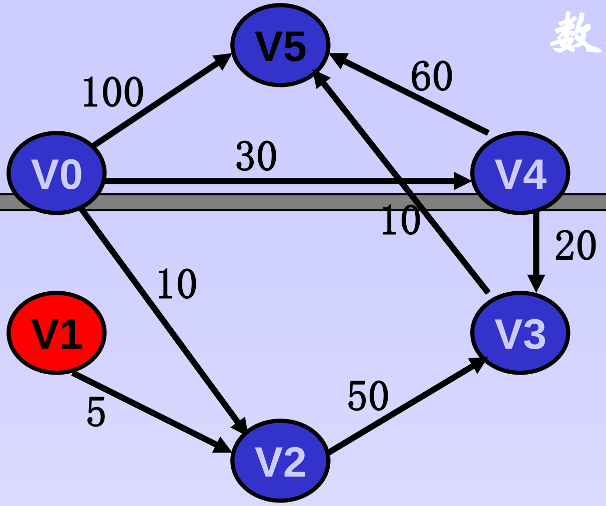
7.6 最短路径



Step5: 选择当前的最短路径(v0,v4,v3)，将v3加入S。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞	∞		
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)	50 (v0,v4,v3)		
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)	90 (v0,v4,v5)		
vj	V2	V4	V3		
S	{v0,v2}	{v0,v2,v4}	{v0,v2,v3, v4}		

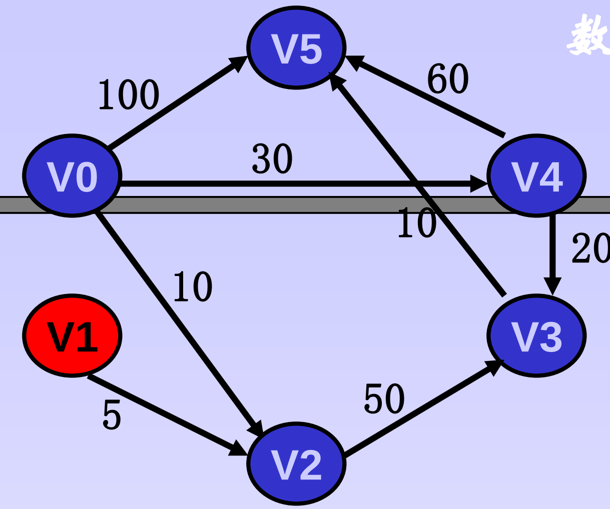
7.6 最短路径



Step6：更新从v0到v1、v5的最短路径

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞	∞	∞	
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)	50 (v0,v4,v3)		
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)	90 (v0,v4,v5)	60 (v0,v4,v3,v5)	
vj	V2	V4	V3		
S	{v0,v2}	{v0,v2,v4}	{v0,v2,v3, v4}		

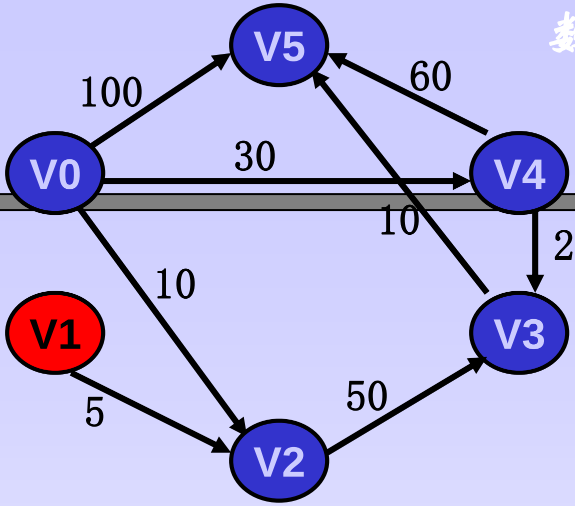
7.6 最短路径



Step7： 选择当前的最短路径 **(v0,v4,v3, v5)**，将v5加入S。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞	∞	∞	
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)	50 (v0,v4,v3)		
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)	90 (v0,v4,v5)	60 (v0,v4,v3,v5)	
vj	V2	V4	V3	V5	
S	{v0,v2}	{v0,v2,v4}	{v0,v2,v3, v4}	{v0,v2,v3,v4, v5}	

7.6 最短路径



Step8：v0到v1不存在路径，其路径长度为∞。

终点	i=1	i=2	i=3	i=4	i=5
v1	∞	∞	∞	∞	∞，无
v2	10 (v0,v2)				
v3	∞	60 (v0,v2,v3)	50 (v0,v4,v3)		
v4	30 (v0,v4)	30 (v0,v4)			
v5	100 (v0,v5)	100 (v0,v5)	90 (v0,v4,v5)	60 (v0,v4,v3,v5)	
vj	V2	V4	V3	V5	
S	{v0,v2}	{v0, v4}	{v0, v4,v3}	{v0, v4,v3,v5}	

7.6 最短路径

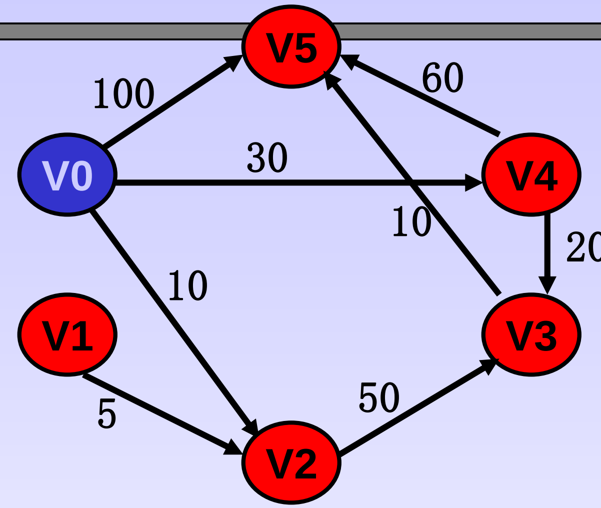
```
typedef int P[MAX][MAX]; //PathMatrix
typedef int D[MAX]; //ShortPathTable
void ShortestPath(MGraph G, int v0, PathMatrix &P,
                         ShortPathTable &D)
// 求图G从v0到其余顶点v的最短路径P[v]和路径长度D[v],
// 如果P[v][w]为TRUE，则w为v0到v的最短路径上的顶点。
{  for( i=0; i<G.vexnum; i++)    //初始化
   {   final[i] = FALSE;    // 表示顶点i∉S
       D[i] = G.Edge[v0][i];
       for( j=0; j<G.vexnum; j++)     P[i][j] = FALSE;
       if( D[i] < ∞) {  P[i][v0] = TRUE;  P[i][i] = TURE; }
   }
   D[v0] = 0;  final[v0] = TRUE;
```

7.6 最短路径

```
for( i=1; i<G.vexnum; i++)   //主循环，每次求一个v到v0的最短路径
{  min = ∞;
   for( j=0; j<G.vexnum; j++)  // 求当前的最短路径的终点vj
     if( !final[j])
       if( D[j] < min )  { vj = j;  min = D[j]; }
   final[vj] = TRUE;    // 将顶点vj加入S
   for( w=0; w<G.vexnum; w++)  // 更新当前的最短路径及长度
     if( !final[w] && (min+G.Edge[vj][w]<D[w] ))
    {  D[w] = min+G.Edge[vj][w];
        for(j=0; j<G.vexnum; j++)  P[w][j] = P[vj][j]; // P[w]=P[vj]，若到w，
                                                          // 则首先到vj
        P[w][w] = TURE;
    }
}
}
```

7.6 最短路径

(1) 初试状态



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

G.Edge

∞	∞	10	∞	30	100

矩阵D

T	F	F	F	F	F

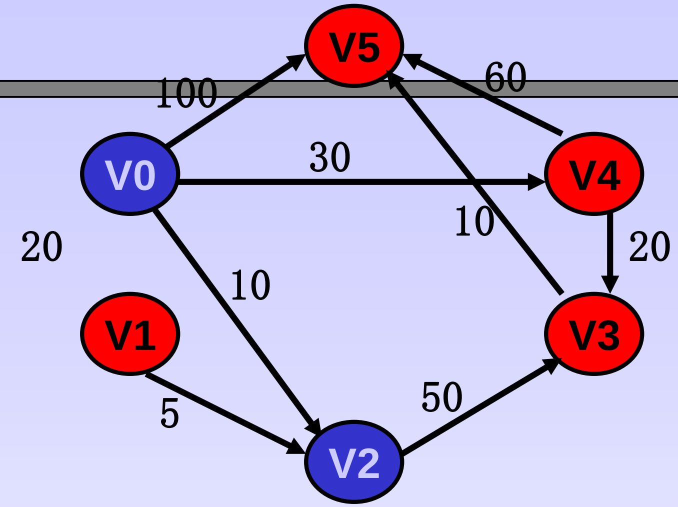
矩阵final

T	F	F	F	F	F
F	F	F	F	F	F
T	F	T	F	F	F
F	F	F	F	F	F
T	F	F	F	T	F
T	F	F	F	F	T

矩阵P

7.6 最短路径

(2) 找到V0--V2的最短路径，并调整V0到V3的最短路径



∞	∞	10	**60**	30	100

矩阵D

T	F	T	F	F	F

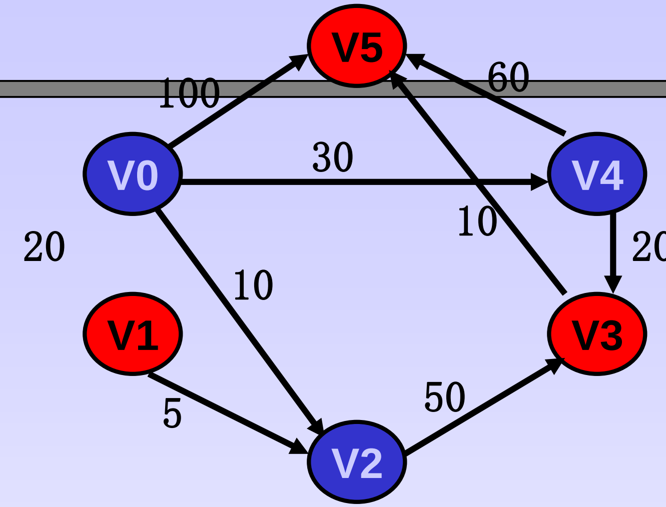
矩阵final

T	F	F	F	F	F
F	F	F	F	F	F
T	F	T	F	F	F
T	F	T	T	F	F
T	F	F	F	T	F
T	F	F	F	F	T

矩阵P

7.6 最短路径

(3) 找到V0—V4的最短路径,
并调整V0到V3,V0到V5的最
短路径



∞	∞	10	**50**	30	90

矩阵D

T	F	T	F	T	F

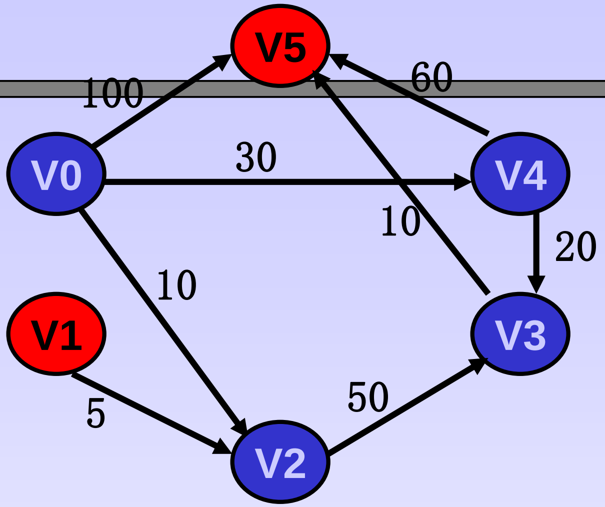
矩阵final

T	F	F	F	F	F
F	F	F	F	F	F
T	F	T	F	F	F
T	F	F	T	T	F
T	F	F	F	T	F
T	F	F	F	T	T

矩阵P

7.6 最短路径

(4) 找到V0—V3的最短路径，
并调整V0到V5的最短路径



∞	∞	10	**50**	30	60

矩阵D

T	F	T	T	T	F

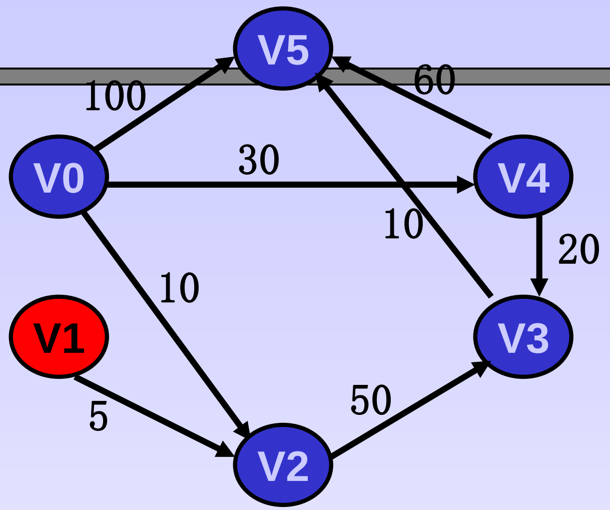
矩阵final

T	F	F	F	F	F
F	F	F	F	F	F
T	F	T	F	F	F
T	F	F	T	T	F
T	F	F	F	T	F
T	F	F	T	T	T

矩阵P

7.6 最短路径

(5) 找到V0—V5的最短路径



∞	∞	10	**50**	30	60

矩阵D

T	F	T	T	T	T

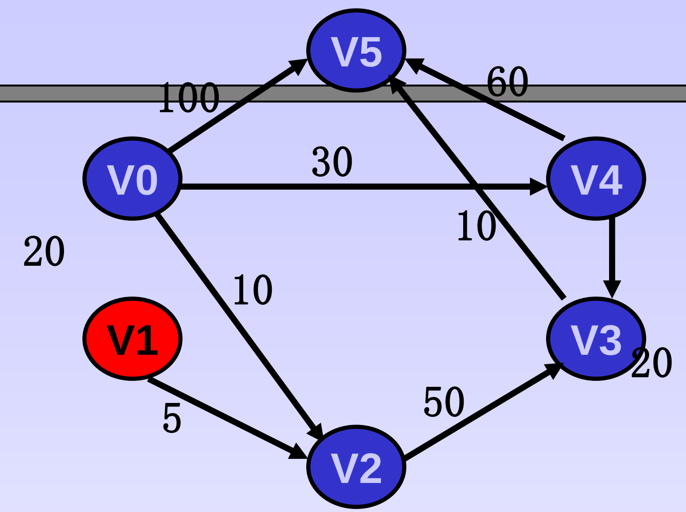
矩阵final

T	F	F	F	F	F
F	F	F	F	F	F
T	F	T	F	F	F
T	F	F	T	T	F
T	F	F	F	T	F
T	F	F	T	T	T

矩阵P

7.6 最短路径

(6) V0—V1无路径,结束



∞	∞	10	**50**	30	60

矩阵D

T	F	T	T	T	T

矩阵final

T	F	F	F	F	F
F	F	F	F	F	F
T	F	T	F	F	F
T	F	F	T	T	F
T	F	F	F	T	F
T	F	F	T	T	T

矩阵P

7.6 最短路径

7.6.2 每一对顶点之间的最短路径：Floyd 算法

- 问题的提法：已知一个各边权值均大于0的带权有向图，对每一对顶点 $v_i \neq v_j$，要求求出v_i与v_j之间的最短路径和最短路径长度。
- Floyd算法的基本思想：

定义一个n阶方阵序列： $D^{(-1)}, D^{(0)}, \ldots, D^{(n-1)}$.

其中: $D^{(-1)}[i][j] = Edge[i][j]$;

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

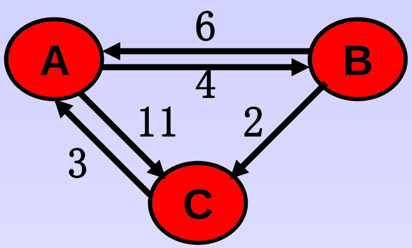
$$k = 0,1,\ldots, n-1$$

$D^{(0)}[i][j]$是从顶点v_i到v_j, 中间顶点是v_0的最短路径的长度, $D^{(k)}[i][j]$是从顶点v_i到v_j，中间顶点的个数不大于k的最短路径的长度, $D^{(n-1)}[i][j]$是从顶点v_i到v_j的最短路径长度。

- 最短路径的表示：采用n个PathMatrix类型的矩阵P，表示每个顶点到其余顶点的最短路径

7.6 最短路径

- Floyd算法的运行过程例1：



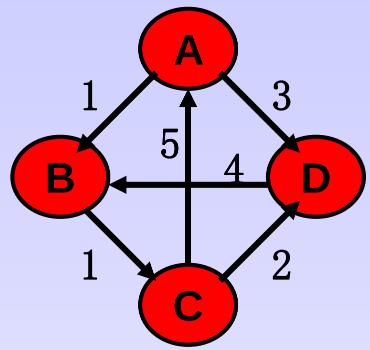
$$
\begin{array}{c|ccc}
 & A & B & C \\
\hline
A & 0 & 4 & 11 \\
B & 6 & 0 & 2 \\
C & 3 & \infty & 0
\end{array}
$$

$D^{(-1)}$			$D^{(0)}$			$D^{(1)}$			$D^{(2)}$		
0	4	11	0	4	11	0	4	6	0	4	6
6	0	2	6	0	2	6	0	2	5	0	2
3	∞	0	3	7	0	3	7	0	3	7	0
$P^{(-1)}$			$P^{(0)}$			$P^{(1)}$			$P^{(2)}$		
	AB	AC		AB	AC		AB	ABC		AB	ABC
BA		BC	BA		BC	BA		BC	BCA		BC
CA			CA	CAB		CA	CAB		CA	CAB	

7.6 最短路径

- **Floyd算法的运行过程例2：**

注：下图中没有画出$D^{(3)}$与$P^{(3)}$，因为$D^{(3)}$与$D^{(2)}$相同，$P^{(3)}$与$P^{(2)}$相同。



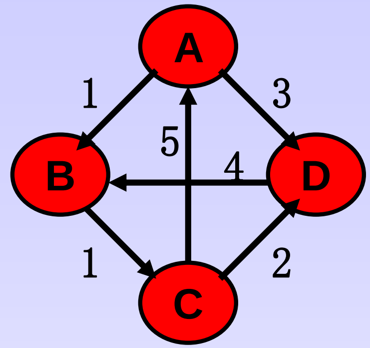
$$
\begin{array}{c}
 & \begin{array}{cccc} A & B & C & D \end{array} \\
\begin{array}{c} A \\ B \\ C \\ D \end{array} &
\begin{pmatrix} 0 & 1 & \infty & 3 \\ \infty & 0 & 1 & \infty \\ 5 & \infty & 0 & 2 \\ \infty & 4 & \infty & 0 \end{pmatrix}
\end{array}
$$

$D^{(-1)}$				$D^{(0)}$				$D^{(1)}$				$D^{(2)}$			
0	1	∞	3	0	1	∞	3	0	1	2	3	0	1	2	3
∞	0	1	∞	∞	0	1	∞	∞	0	1	∞	6	0	1	3
5	∞	0	2	5	6	0	2	5	6	0	2	5	6	0	2
∞	4	∞	0	∞	4	∞	0	∞	4	5	0	10	4	5	0
$P^{(-1)}$				$P^{(0)}$				$P^{(1)}$				$P^{(2)}$			
	AB		AD		AB		AD		AB	ABC	AD		AB	ABC	AD
		BC				BC				BC		BCA		BC	BCD
CA			CD	CA	CAB		CD	CA	CAB		CD	CA	CAB		CD
	DB				DB				DB	DBC		DBCA	DB	DBC	

7.6 最短路径

- **Floyd算法的运行过程例2：**

注：下图中没有画出$D^{(3)}$与$P^{(3)}$，因为$D^{(3)}$与$D^{(2)}$相同，$P^{(3)}$与$P^{(2)}$相同。



$$\begin{array}{cc} & \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{array}{c} A \\ B \\ C \\ D \end{array} & \begin{pmatrix} 0 & 1 & \infty & 3 \\ \infty & 0 & 1 & \infty \\ 5 & \infty & 0 & 2 \\ \infty & 4 & \infty & 0 \end{pmatrix} \end{array}$$

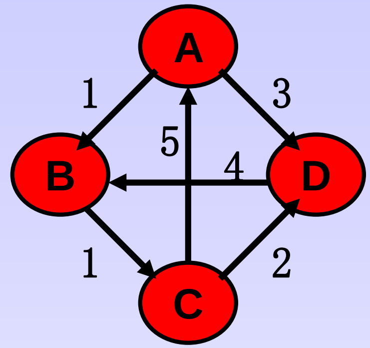
$D^{(-1)}$				$D^{(0)}$				$D^{(1)}$				$D^{(2)}$			
0	1	∞	3	0	1	∞	3	0	1	2	3	0	1	2	3
∞	0	1	∞	∞	0	1	∞	∞	0	1	∞	6	0	1	3
5	∞	0	2	5	6	0	2	5	6	0	2	5	6	0	2
∞	4	∞	0	∞	4	∞	0	∞	4	5	0	10	4	5	0

$P^{(-1)}$				$P^{(0)}$				$P^{(1)}$				$P^{(2)}$			
	AB		AD		AB		AD		AB	ABC	AD		AB	ABC	AD
		BC				BC				BC		BCA		BC	BCD
CA			CD	CA	CAB		CD	CA	CAB		CD	CA	CAB		CD
	DB				DB				DB	DBC		DBCA	DB	DBC	

7.6 最短路径

- **Floyd算法的运行过程例2：**

注：下图中没有画出$D^{(3)}$与$P^{(3)}$，因为$D^{(3)}$与$D^{(2)}$相同，$P^{(3)}$与$P^{(2)}$相同。



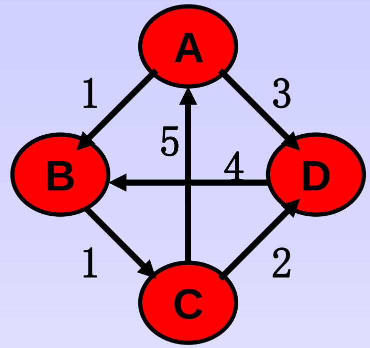
$$
\begin{array}{c|cccc}
 & A & B & C & D \\
\hline
A & 0 & 1 & \infty & 3 \\
B & \infty & 0 & 1 & \infty \\
C & 5 & \infty & 0 & 2 \\
D & \infty & 4 & \infty & 0
\end{array}
$$

$D^{(-1)}$				$D^{(0)}$				$D^{(1)}$				$D^{(2)}$			
0	1	∞	3	0	1	∞	3	0	1	2	3	0	1	2	3
∞	0	1	∞	∞	0	1	∞	∞	0	1	∞	6	0	1	3
5	∞	0	2	5	6	0	2	5	6	0	2	5	6	0	2
∞	4	∞	0	∞	4	∞	0	∞	4	5	0	10	4	5	0
$P^{(-1)}$				$P^{(0)}$				$P^{(1)}$				$P^{(2)}$			
	AB		AD		AB		AD		AB	ABC	AD		AB	ABC	AD
		BC				BC				BC		BCA		BC	BCD
CA			CD	CA	CAB		CD	CA	CAB		CD	CA	CAB		CD
	DB				DB				DB	DBC		DBCA	DB	DBC	

7.6 最短路径

- Floyd算法的运行过程例2：

注：下图中没有画出$D^{(3)}$与$P^{(3)}$，因为$D^{(3)}$与$D^{(2)}$相同，$P^{(3)}$与$P^{(2)}$相同。



$$\begin{array}{c|cccc} & A & B & C & D \\ \hline A & 0 & 1 & \infty & 3 \\ B & \infty & 0 & 1 & \infty \\ C & 5 & \infty & 0 & 2 \\ D & \infty & 4 & \infty & 0 \end{array}$$

$D^{(-1)}$				$D^{(0)}$				$D^{(1)}$				$D^{(2)}$			
0	1	∞	3	0	1	∞	3	0	1	2	3	0	1	2	3
∞	0	1	∞	∞	0	1	∞	∞	0	1	∞	6	0	1	3
5	∞	0	2	5	6	0	2	5	6	0	2	5	6	0	2
∞	4	∞	0	∞	4	∞	0	∞	4	5	0	10	4	5	0
$P^{(-1)}$				$P^{(0)}$				$P^{(1)}$				$P^{(2)}$			
	AB		**AD**		**AB**		**AD**		**AB**	**ABC**	**AD**		**AB**	**ABC**	**AD**
		BC				**BC**				**BC**		**BCA**		**BC**	**BCD**
CA			**CD**	**CA**	**CAB**		**CD**	**CA**	**CAB**		**CD**	**CA**	**CAB**		**CD**
	DB				**DB**				**DB**	**DBC**		**DBCA**	**DB**	**DBC**	