7.6.2 多源最短路径算法

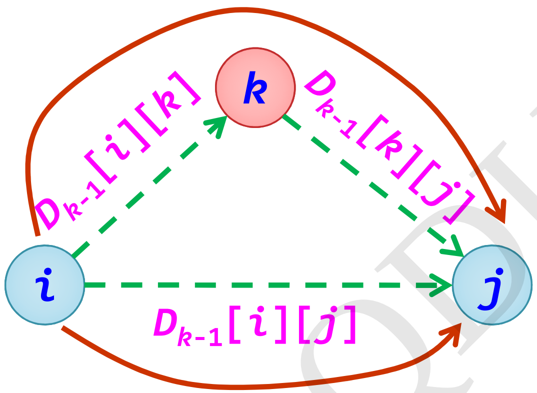
- 用Dijkstra算法可以求得有向图G=(V，E)中每一对顶点间的最短路径。方法是：每次以一个不同的顶点为源点重复Dijkstra算法便可求得每一对顶点间的最短路径，时间复杂度是O(n^3)。
- 弗洛伊德（Floyd）提出了另一个算法，其时间复杂度仍是O(n^3)，但算法形式更为简明，步骤更为简单。

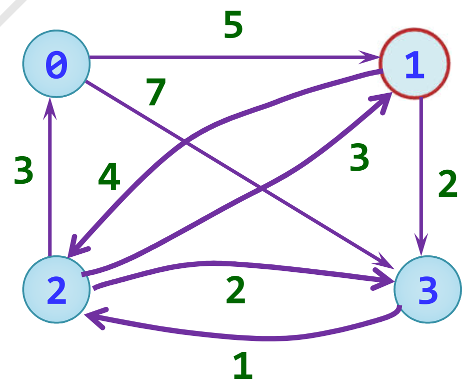
- 假设有向图G=(V,E)采用邻接矩阵g表示，另外设置一个二维数组D用于存放当前顶点之间的最短路径长度，即分量D[i][j]表示当前顶点i到顶点j的最短路径长度。
- 弗洛伊德算法的基本思想：递推产生一个矩阵序列D_0、D_1、…、D_k、…、D_{n-1}，其中D_k[i][j]表示从顶点i到顶点j的路径上所经过的顶点编号不大于k的最短路径长度。

归纳起来，弗洛伊德思想可用如下的表达式来描述：

$D_{-1}[i][j]$=g.edges[i][j]

$D_k[i][j]=MIN\{D_{k-1}[i][j],\ D_{k-1}[i][k]+D_{k-1}[k][j]\}$ $0 \le k \le n-1$

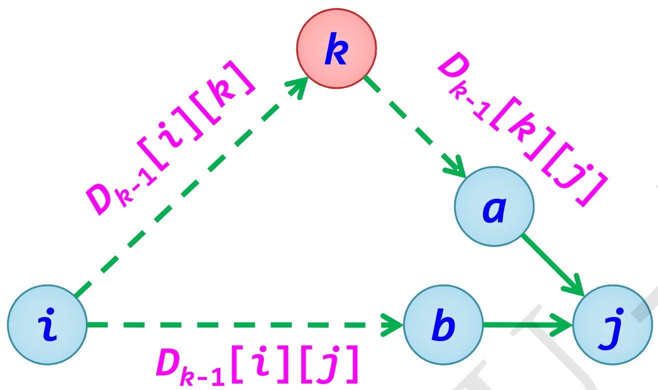




➢ 两条路径中选最小者：

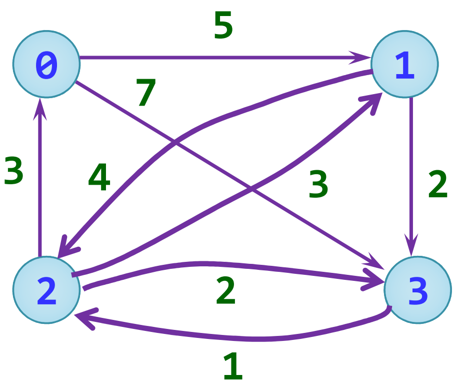
$D_k[i][j]$= **MIN** { $D_{k-1}[i][j]$, $D_{k-1}[i][k]+D_{k-1}[k][j]$ }

- 另外用二维数组path保存最短路径，它与当前迭代的次数有关，即当迭代完毕，path[i][j]存放从顶点i到顶点j的最短路径的前一个顶点的编号。



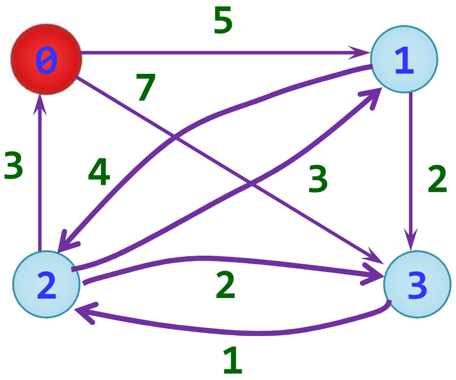
- path$_{k-1}$[i][j]=b，path$_{k-1}$[k][j]=a
- 若D_{k-1}[i][j]>D_{k-1}[i][k]+D_{k-1}[k][j]，选择经过顶点k的路径，即path$_k$[i][j]=a=path$_{k-1}$[k][j]。
- 否则不改变。

弗洛伊德算法求多源最短路径示例



$$\begin{bmatrix} 0 & 5 & \infty & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

D_{-1}				$path_{-1}$			
0	5	∞	7	-1	0	-1	0
∞	0	4	2	-1	-1	1	1
3	3	0	2	2	2	-1	2
∞	∞	1	0	-1	-1	3	-1

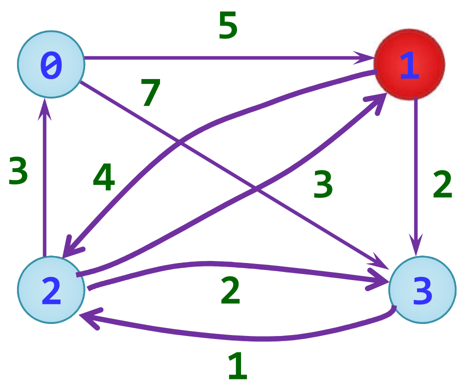


$$\begin{bmatrix} 0 & 5 & \infty & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

✓ 在考虑顶点0时：

没有任何最短路径得到修改！

D_0				$path_0$			
0	5	∞	7	-1	0	-1	0
∞	0	4	2	-1	-1	1	1
3	3	0	2	2	2	-1	2
∞	∞	1	0	-1	-1	3	-1

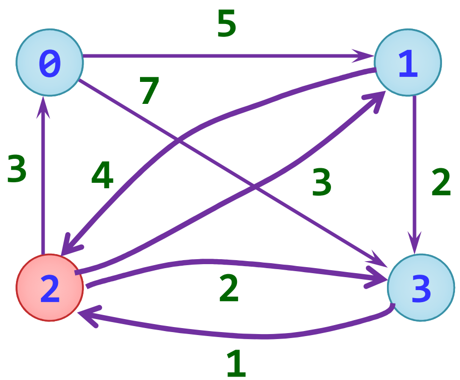


$$\begin{bmatrix} 0 & 5 & \infty & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

在考虑顶点1时：

➢ 0→2：由无路径改为0→1→2，长度为9，path[0][2]改为1

D_1				$path_1$			
0	5	9	7	-1	0	1	0
∞	0	4	2	-1	-1	1	1
3	3	0	2	2	2	-1	2
∞	∞	1	0	-1	-1	3	-1

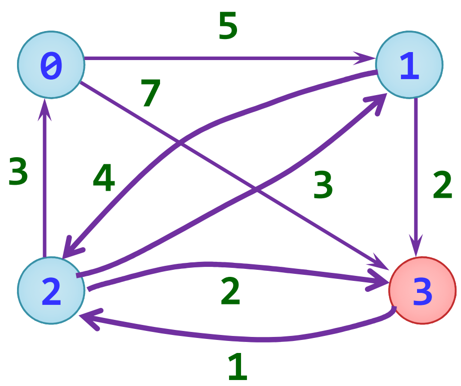


$$\begin{bmatrix} 0 & 5 & 9 & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{bmatrix}$$

在考虑顶点2时：

- 1→0：由无路径改为1→2→0，长度为7， path[1][0]改为2
- 3→0：由无路径改为3→2→0，长度为4， path[3][0]改为2
- 3→1：由无路径改为3→2→1，长度为4， path[3][1]改为2

D_2				$path_2$			
0	5	9	7	-1	0	1	0
7	0	4	2	2	-1	1	1
3	3	0	2	2	2	-1	2
4	4	1	0	2	2	3	-1



$$\begin{bmatrix} 0 & 5 & 9 & 7 \\ 7 & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ 4 & 4 & 1 & 0 \end{bmatrix}$$

在考虑顶点3时:

- 0→2: 由0→1→2改为0→3→2 ，长度为8， path[0][2]改为3
- 1→0: 由1→2→0改为1→3→2→0，长度为6， path[1][0]改为2
- 1→2: 由1→2改为1→3→2，长度为3， path[1][2]改为3

D_3				$path_3$			
0	5	8	7	-1	0	3	0
6	0	3	2	2	-1	3	1
3	3	0	2	2	2	-1	2
4	4	1	0	2	2	3	-1

D_3				$path_3$			
0	5	8	7	-1	0	3	0
6	0	3	2	2	-1	3	1
3	3	0	2	2	2	-1	2
4	4	1	0	2	2	3	-1

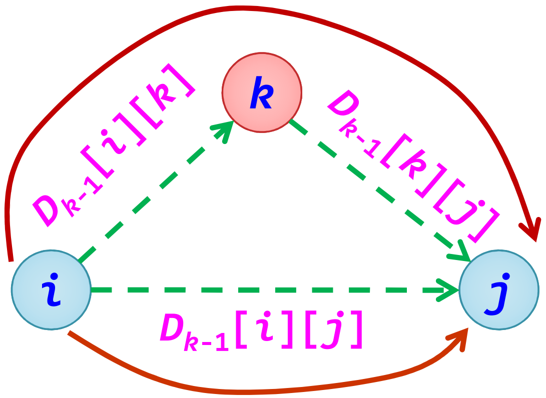
$$\begin{bmatrix} 0 & 5 & 8 & 7 \\ 6 & 0 & 3 & 2 \\ 3 & 3 & 0 & 2 \\ 4 & 4 & 1 & 0 \end{bmatrix}$$

- 在得到D_3和$path_3$后，由D_3数组可以直接得到两个顶点之间的最短路径长度，如D_3[1][0]=6，说明顶点1到0的最短路径长度为6。
- path[1][0]=2，说明顶点0的前一顶点是顶点2，path[1][2]=3，表示顶点2的前一个顶点是顶点3，path[1][3]=1，表示顶点3的前一个顶点是顶点1，找到起点。依次得到的顶点序列为0、2、3、1，则顶点1到0的最短路径为1→3→2→0。

弗洛伊德算法如下：

```
void Floyd(MGraph g)  //求每对顶点之间的最短路径
{  int D[MAXVEX][MAXVEX];    //建立D数组
   int path[MAXVEX][MAXVEX]; //建立path数组
   int i,j,k;

   for (i=0;i<g.n;i++)       //给数组D和path置初值
     for (j=0;j<g.n;j++)
     {  D[i][j]=g.edges[i][j];
        if (i!=j && g.edges[i][j]<INF)
           path[i][j]=i;     //i和j顶点之间有一条边时
        else                 //i和j顶点之间没有一条边时
           path[i][j]=-1;
     }
```

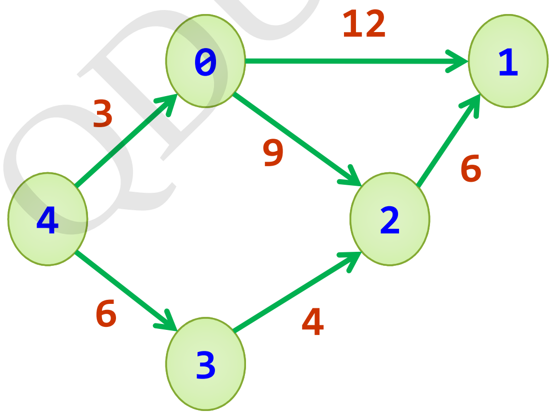



```
for (k=0;k<g.n;k++)                                    //求Dk[i][j]
   {  for (i=0;i<g.n;i++)
         for (j=0;j<g.n;j++)
           if (D[i][j]>D[i][k]+D[k][j])    //找到更短路径
           {  D[i][j]=D[i][k]+D[k][j];     //修改路径长度
              path[i][j]=path[k][j];       //修改最短路径
           }
   }
}
```

弗洛伊德算法Floyd(g)中有三重循环，其时间复杂度为O(n^3)。

【示例】 给定n个村庄之间的交通图，如下图所示，若村庄i与村庄j之间有路可通，则将顶点i与顶点j之间用边连接，边上的权值w_{ij}表示这条道路的长度。

现打算在这n个村庄中选定一个村庄建一所医院。设计一个算法求该医院应建在哪个村庄（称为最佳村庄），能使其他所有村庄到医院的路径总和最短（当有多个这样的村庄时，求出任一个村庄即可）。



解：假设村庄图采用邻接矩阵*g*表示。

- 先采用**Floyd**算法求出图中每对顶点之间的最短路径长度数组*D*。
- 再累加每行的元素之和并放到*B*数组中，其中*B*[*i*]表示顶点*i*到其他所有顶点的最短路径长度之和，
- 最后求出*B*中最小元素*B*[minv]，并返回minv。

求出的*D*:

	0	1	2	3	4
0	0	12	9	9	3
1	12	0	6	10	15
2	9	6	0	4	10
3	9	10	4	0	6
4	3	15	10	6	0

求出的*B*:

0	33
1	43
2	29
3	29
4	34

最佳村庄编号为2或者3

END