```
                改变原来路径及其长度，否则什么都不做*/
                if(set[j]==0&&dist[u]+g.edges[u][j]<dist[j])
                {
                    dist[j]=dist[u]+g.edges[u][j];
                    path[j]=u;
                }
            }
        }
        /*关键操作结束*/
    }
    /*函数结束时，dist[]数组中存放了 v 点到其余顶点的最短路径长度，path[]中存放 v 点到其余各顶点的最短路径*/
```

3．迪杰斯特拉算法时间复杂度分析

由算法代码可知，本算法主要部分为一个双重循环，外层循环内部有两个并列的单层循环，可以任取一个循环内的操作作为基本操作，基本操作执行的总次数即为双重循环执行的次数，为 n^2 次，因此本算法的时间复杂度为 $O(n^2)$。

7.5.2 弗洛伊德算法

迪杰斯特拉算法是求图中**某一顶点到其余各顶点的最短路径**，如果求图中**任意一对顶点间**的最短路径，则通常用弗洛伊德算法。

考试中涉及最多的是求用四阶方阵表示的图中每两点之间的最短路径的过程，方阵的阶数高了，计算量就比较大，考试中不会涉及太多。本算法的代码写起来要比迪杰斯特拉算法简单得多，因此在考试中，如果不对时间复杂度要求过于苛刻，尽量写本算法的代码，以减少错误。

对于本算法，需要掌握它的求解过程和程序代码，这两点比较简单，记住即可。对于考研要求，不需要钻研太多。下面就通过一个例子来总结用本算法求解最短路径的一般方法。

图 7-24 所示为一个有向图，各边的权值如图所示，用弗洛伊德算法求解其最短路径的过程如下。

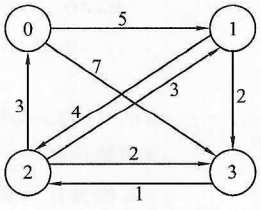

图 7-24 有向图

对于图 7-24，对应的邻接矩阵如下：

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

初始时要设置两个矩阵 **A** 和 **Path**，**A** 用来记录当前已经求得的任意两个顶点最短路径的长度，**Path** 用来记录当前两顶点间最短路径上要经过的中间顶点。

1）初始时有：$\mathbf{A}_{-1}=\begin{pmatrix} 0 & 5 & \infty & 7 \\ \infty & 0 & 4 & 2 \\ 3 & 3 & 0 & 2 \\ \infty & \infty & 1 & 0 \end{pmatrix}$ $\mathbf{Path}_{-1}=\begin{pmatrix} -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{pmatrix}$（矩阵名的下标代表每一步中所选的中间顶点，图的顶点编号从 0 开始，初始的时候没有中间点，因此下标设为-1）。

2）以 0 为中间点，参照上一步矩阵中的结果，检测所有顶点对：{0，1}，{0，2}，{0，3}，{1，0}，{1，2}，{1，3}，{2，0}，{2，1}，{2，3}，{3，0}，{3，1}，{3，2}，假设当前所检测的顶点对为{i，j}，如果 A[i][j]>A[i][0]+A[0][j]，则将 A[i][j]改为 A[i][0]+A[0][j]的值，并且将 Path[i][j]改为 0。

经本次检测与修改，所得矩阵如下：

$$\mathbf{A}_0=\begin{pmatrix}0&5&\infty&7\\\infty&0&4&2\\3&3&0&2\\\infty&\infty&1&0\end{pmatrix}\qquad \mathbf{Path}_0=\begin{pmatrix}-1&-1&-1&-1\\-1&-1&-1&-1\\-1&-1&-1&-1\\-1&-1&-1&-1\end{pmatrix}$$

3）以 1 为中间点，参照上一步矩阵中的结果，检测所有顶点对，其中有 A[0][2]>A[0][1]+A[1][2]=5+4=9，因此将 A[0][2]改为 9，Path[0][2]改为 1。

经本次检测与修改，所得矩阵如下：

$$\mathbf{A}_1=\begin{pmatrix}0&5&9&7\\\infty&0&4&2\\3&3&0&2\\\infty&\infty&1&0\end{pmatrix}\qquad \mathbf{Path}_1=\begin{pmatrix}-1&-1&1&-1\\-1&-1&-1&-1\\-1&-1&-1&-1\\-1&-1&-1&-1\end{pmatrix}$$

4）以 2 为中间点，参照上一步矩阵中的结果，检测所有顶点对，其中有 A[1][0]>A[1][2]+A[2][0]=4+3=7，A[3][1]>A[3][2]+A[2][1]=1+3=4，A[3][0]>A[3][2]+A[2][0]= 1+3=4，因此将 A[1][0]改为 7，将 A[3][1]改为 4，将 A[3][0]改为 4，将 Path[1][0]、Path[3][0]和 Path[3][1]都改为 2。

经本次检测与修改，所得矩阵如下：

$$\mathbf{A}_2=\begin{pmatrix}0&5&9&7\\7&0&4&2\\3&3&0&2\\4&4&1&0\end{pmatrix}\qquad \mathbf{Path}_2=\begin{pmatrix}-1&-1&1&-1\\2&-1&-1&-1\\-1&-1&-1&-1\\2&2&-1&-1\end{pmatrix}$$

5）以 3 为中间点，参照上一步矩阵中的结果，检测所有顶点对，其中有 A[0][2]>A[0][3]+A[3][2]=7+1=8，A[1][0]>A[1][3]+A[3][0]=2+4=6，A[1][2]>A[1][3]+A[3][2]= 2+1=3，因此将 A[0][2]改为 8，将 A[1][0]改为 6，将 A[1][2]改为 3，将 Path[0][2]、Path[1][0]和 Path[1][2]都改为 3。

经本次检测与修改，所得矩阵如下：

$$\mathbf{A}_3=\begin{pmatrix}0&5&8&7\\6&0&3&2\\3&3&0&2\\4&4&1&0\end{pmatrix}\qquad \mathbf{Path}_3=\begin{pmatrix}-1&-1&3&-1\\3&-1&3&-1\\-1&-1&-1&-1\\2&2&-1&-1\end{pmatrix}$$

至此，得最终的矩阵 **A** 与 **Path** 如下：

$$\mathbf{A}=\begin{pmatrix}0&5&8&7\\6&0&3&2\\3&3&0&2\\4&4&1&0\end{pmatrix}\qquad \mathbf{Path}=\begin{pmatrix}-1&-1&3&-1\\3&-1&3&-1\\-1&-1&-1&-1\\2&2&-1&-1\end{pmatrix}$$

由矩阵 **A** 可以查出图中任意两点间的最短路径长度。例如，要求顶点 1 到顶点 3 的最短路径长度，可查得 A[1][3]为 2。

由矩阵 **A** 和矩阵 **Path** 可以算出任意两点间最短路径上的顶点序列或边序列。例如，要求顶点 1 到顶点 0 最短路径上的顶点序列，可按照如下步骤进行。

① 由 A[1][0]=6 可知，顶点 1 到顶点 0 存在路径且最短，则可执行下边的步骤（若 A[1][0]=∞，则说明 1 到 0 不存在路径，路径顶点序列计算结束）；

② 由 Path[1][0]=3 可知，从顶点 1 到顶点 0 要经过顶点 3，将 3 作为下一步的起点；

③ 由 Path[3][0]=2 可知，从顶点 3 到顶点 0 要经过顶点 2，将 2 作为下一步的起点；

④ 由 Path[2][0]=−1 可知，从顶点 2 有直接指向顶点 0 的边，求解结束。

由此得到从顶点 1 到顶点 0 最短路径为 1→3→2→0。

注意：细心的同学现在应该看出，上边输出两点间路径的步骤是一个递归的过程。反映其执行过程

的伪代码如下：

```
void printPath(int u,int v,int path[][max], int A[][max])
//输出从 u 到 v 的最短路径上顶点序列
{
    if(A[u][v] == INF)                    //INF 是已定义的常量，数值很大不可能小于图中
                                            边的权值，代表无穷大输出无路径提示；
    else
    {
       if(path[u][v]==-1)
           直接输出边<u,v>;
       else
       {
           int mid=path[u][v];
           printPath(u,mid,path);    //处理 mid 前半段路径
           printPath(mid,v,path);    //处理 mid 后半段路径
       }
    }
}
```


从上述示例中可以总结出用弗洛伊德算法求解最短路径的一般过程，具体如下：

1）设置两个矩阵 **A** 和 **Path**，初始时将图的**邻接矩阵**赋值给 **A**，将矩阵 **Path** 中元素全部设置为-1。

2）以顶点 k 为中间顶点，k 取 0～n-1（n 为图中顶点个数），对图中所有顶点对{i，j}进行如下检测与修改：

如果 A[i][j]>A[i][k]+A[k][j]，则将 A[i][j]改为 A[i][k]+A[k][j]的值，将 Path[i][j]改为 k，否则什么都不做。

由上述弗洛伊德算法求解最短路径的一般过程可写出以下弗洛伊德算法代码，其中定义两个二维数组 A[][]和 Path[][]，用来保存上述矩阵 **A** 和 **Path**。

```
void Floyd(MGraph* g,int Path[][maxSize],int A[][maxSize])
{   //图 g 的边矩阵中，用 INF 来表示两点之间不存在边
    int i,j,k;
    /*这个双循环对数组 A[][]和 Path[][]进行了初始化*/
    for(i=0;i<g->n;++i)
       for(j=0;j<g->n;++j)
       {
          A[i][j]=g->edges[i][j];
          Path[i][j]=-1;
       }
    /*下面这个三层循环是本算法的主要操作，完成了以 k 为中间点对所有的顶点对(i,j)进行检测和修改*/
    for(k=0;k<g->n;++k)
       for(i=0;i<g->n;++i)
          for(j=0;j<g->n;++j)
             if(A[i][j]>A[i][k]+A[k][j])
             {
                A[i][j]=A[i][k]+A[k][j];
```

```
                    Path[i][j]=k;
                }
    }
```

弗洛伊德算法的时间复杂度分析：

由算法代码可知，本算法的主要部分是一个三层循环，取最内层循环的操作作为基本操作，则基本操作执行次数为 n^3，因此时间复杂度为 $O(n^3)$。

7.6 拓扑排序

7.6.1 AOV 网

活动在顶点上的网（Activity On Vertex network，AOV）是一种可以形象地反映出整个工程中各个活动之间的先后关系的有向图。图 7-25 所示为制造一个产品的 AOV 网。制造该产品需要 3 个环节，第一个环节获得原材料，第二个环节生产出 3 个部件，第三个环节由 3 个部件组装成成品。在原材料没有准备好之前不能生产部件，在 3 个部件全部被生产出来之前不能组装成成品，这样一个工程各个活动之间的先后次序关系就可以用一个有向图来表示，称为 AOV 网。

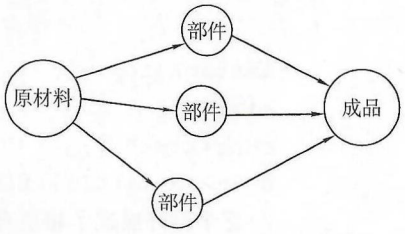

图 7-25 反映一个产品生产过程先后次序的 AOV 网

考试中，只要知道 AOV 网是一种以顶点表示活动、以边表示活动的先后次序且没有回路的有向图即可。因为 AOV 网有实际意义，所以出现回路就代表一项活动以自己为前提，这显然违背实际。

7.6.2 拓扑排序核心算法

对一个有向无环图 G 进行拓扑排序，是将 G 中所有顶点排成一个线性序列，使得图中任意一对顶点 u 和 v，若存在由 u 到 v 的路径，则在拓扑排序序列中一定是 u 出现在 v 的前边。

在一个有向图中找到一个拓扑排序序列的过程如下：

1）从有向图中选择一个没有前驱（入度为 0）的顶点输出；

2）删除 1）中的顶点，并且删除从该顶点发出的全部边；

3）重复上述两步，直到剩余的图中不存在没有前驱的顶点为止。

以邻接表为存储结构，怎样实现拓扑排序的算法呢？因为上述步骤中提到要选取入度为 0 的结点并将其输出，要对邻接表表头结构定义进行修改，可加上一个统计结点入度的计数器，修改如下：

```
typedef struct
{
    char  data;
    int count;                    //此句为新增部分，count 来统计顶点当前的入度
    ArcNode *firstarc;
}VNode;
```

假设图的邻接表已经生成，并且各个顶点的入度都已经记录在 count 中，在本算法中要设置一个栈，用来记录当前图中入度为 0 的顶点，还要设置一个计数器 n，用来记录已经输出的顶点个数。

算法开始时置 n 为 0，扫描所有顶点，将入度为 0 的顶点入栈。然后在栈不空的时候循环执行：出栈，将出栈顶点输出，执行++n，并且将由此顶点引出的边所指向的顶点的入度都减 1，并且将入度变为 0 的顶点入栈；出栈，…，栈空时循环退出，排序结束。循环退出后判断 n 是否等于图中的顶点个数。如果相等则返回 1，拓扑排序成功；否则返回 0，拓扑排序失败。