结果是所有结点具有了该网络的等同的、完整的视图。于是每个结点都能够像其他结点一样，运行 LS 算法并计算出相同的最低费用路径集合。

我们下面给出的链路状态路由选择算法叫做 Dijkstra 算法，该算法以其发明者命名。一个密切相关的算法是 Prim 算法，有关图算法的一般性讨论参见［Cormen 2001］。Dijkstra 算法计算从某结点（源结点，我们称之为 u）到网络中所有其他结点的最低费用路径。Dijkstra 算法是迭代算法，其性质是经算法的第 k 次迭代后，可知道到 k 个目的结点的最低费用路径，在到所有目的结点的最低费用路径之中，这 k 条路径具有 k 个最低费用。我们定义下列记号。

- $D(v)$：到算法的本次迭代，从源结点到目的结点 v 的最低费用路径的费用。
- $p(v)$：从源到 v 沿着当前最低费用路径的前一结点（v 的邻居）。
- N'：结点子集；如果从源到 v 的最低费用路径已确知，v 在 N' 中。

该全局路由选择算法由一个初始化步骤和其后的循环组成。循环执行的次数与网络中结点个数相同。一旦终止，该算法就计算出了从源结点 u 到网络中每个其他结点的最短路径。

源结点 u 的链路状态（LS）算法

```
Initialization:
   N' = {u}
   for all nodes v
     if v is a neighbor of u
       then D(v) = c(u,v)
     else D(v) = ∞

Loop
   find w not in N' such that D(w) is a minimum
   add w to N'
   update D(v) for each neighbor v of w and not in N':
        D(v) = min( D(v), D(w) + c(w,v) )
   /* new cost to v is either old cost to v or known
    least path cost to w plus cost from w to v */
until N'= N
```

举一个例子，考虑图 4-27 中的网络，计算从 u 到所有可能目的地的最低费用路径。该算法的计算过程以表格方式总结于表 4-3 中，表中的每一行给出了迭代结束时该算法的变量的值。我们详细地考虑前几个步骤。

表 4-3　在图 4-27 中的网络上运行的链路状态算法

步骤	N'	$D(v)$，$p(v)$	$D(w)$，$p(w)$	$D(x)$，$p(x)$	$D(y)$，$p(y)$	$D(z)$，$p(z)$
0	u	2，u	5，u	1，u	∞	∞
1	ux	2，u	4，x		2，x	∞
2	uxy	2，u	3，y			4，y
3	$uxyv$		3，y			4，y
4	$uxyvw$					4，y
5	$uxyvwz$					

- 在初始化步骤，从 u 到与其直接相连的邻居 v、x、w 的当前已知最低费用路径分别初始化为 2、1 和 5。特别值得注意的是，到 w 的费用被设为 5（尽管我们很快就会看见一条费用更小的路径确实存在），因为这是从 u 到 w 的直接（一跳）链路费用。到 y 与 z 的费用被设为无穷大，因为它们不直接与 u 连接。