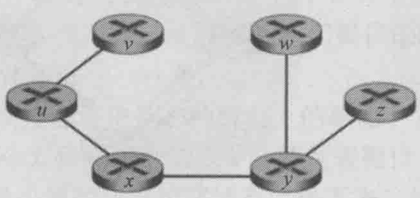
- 在第一次迭代中，我们观察那些还未加到集合 N' 中的结点，并且找出在前一次迭代结束时具有最低费用的结点。那个结点便是 x，其费用是 1，因此 x 被加到集合 N' 中。于是 LS 算法中的第 12 行中的程序被执行，以更新所有结点 v 的 $D(v)$，产生表 4-3 中第 2 行（步骤 1）所示的结果。到 v 的路径费用未变。经过结点 x 到 w（在初始化结束时其费用为 5）的路径费用被发现为 4。因此这条具有更低费用的路径被选中，且沿从 u 开始的最短路径上 w' 的前一结点被设为 x。类似地，到 y（经过 x）的费用被计算为 2，且该表也被相应地更新。
- 在第二次迭代时，结点 v 与 y 被发现具有最低费用路径（2），并且我们任意改变次序将 y 加到集合 N' 中，使得 N' 中含有 u、x 和 y。到仍不在 N' 中的其余结点（即结点 v、w 和 z）的费用通过 LS 算法中的第 12 行进行更新，产生如表 4-3 中第 3 行所示的结果。
- 如此等等。

当 LS 算法终止时，对于每个结点，我们都得到从源结点沿着它的最低费用路径的前一结点。对于每个前一结点，我们又有它的前一结点，以此方式我们可以构建从源结点到所有目的结点的完整路径。通过对每个目的结点存放从 u 到目的地的最低费用路径上的下一跳结点，在一个结点（如结点 u）中的转发表则能够根据此信息而构建。图 4-28 显示了对于图 4-27 中的网络产生的最低费用路径和 u 中的转发表。



目的地	链路
v	(u, v)
w	(u, x)
x	(u, x)
y	(u, x)
z	(u, x)

图 4-28　对于结点 u 的最低费用路径和转发表

该算法的计算复杂性是什么？即给定 n 个结点（不算源结点），在最坏情况下要经过多少次计算，才能找到从源结点到所有目的结点的最低费用路径？在第一次迭代中，我们需要搜索所有的 n 个结点以确定出结点 w——w 不在 N' 中且具有最低费用。在第二次迭代时，我们需要检查 $n-1$ 个结点以确定最低费用。第三次对 $n-2$ 个结点迭代，依次类推。总之，我们在所有迭代中需要搜寻的结点总数为 $n(n+1)/2$，因此我们说前面实现的链路状态算法在最差情况下复杂性为 $O(n^2)$。（该算法的一种更复杂的实现是使用一种称为堆的数据结构，能用对数时间而不是线性时间得到第 9 行的最小值，因此减少其复杂性。）

在完成 LS 算法的讨论之前，我们考虑一下可能出现的问题。图 4-29 显示了一个简单的网络拓扑，图中的链路费用等于链路上承载的负载，例如反映要历经的时延。在该例中，链路费用是非对称的，即仅当在链路 (u, v) 两个方向所承载的负载相同时 $c(u, v)$ 与 $c(v, u)$ 才相等。在该例中，结点 z 产生发往 w 的一个单元的流量，结点 x 也产生发往 w 的一个单元的流量，并且结点 y 也产生发往 w 的一个数量为 e 的流量。初始路由选择情况如图 4-29a 所示，其链路费用对应于承载的流量。

当 LS 算法再次运行时，结点 y 确定（基于图 4-29a 所示的链路费用）顺时针到 w 的路径费用为 1，而逆时针到 w 的路径费用（一直使用的）是 $1+e$。因此 y 到 w 的最低费用