量，每个费用估计 $D_x(y)$ 收敛到 $d_x(y)$，$d_x(y)$ 为从结点 x 到结点 y 的实际最低费用路径的费用［Bersekas 1991］！

距离向量（DV）算法

在每个结点 x：

```
1  Initialization:
2     for all destinations y in N:
3        Dx(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        Dw(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector Dx = [Dx(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       Dx(y) = minv{c(x,v) + Dv(y)}
15
16    if Dx(y) changed for any destination y
17       send distance vector Dx = [Dx(y): y in N] to all neighbors
18
19 forever
```

在该 DV 算法中，当结点 x 发现它的直接相连的链路费用变化或从某个邻居接收到一个距离向量的更新时，它就更新其距离向量估计。但是为了一个给定的目的地 y 而更新它的转发表，结点 x 真正需要知道的不是到 y 的最短路径距离，而是沿着最短路径到 y 的下一跳路由器邻居结点 $v^*(y)$。如你可能期望的那样，下一跳路由器 $v^*(y)$ 是在 DV 算法第 14 行中取得最小的邻居 v。（如果有多个取得最小的邻居 v，则 $v^*(y)$ 能够是其中任何一个最小的邻居。）因此，对于每个目的地 y，在第 13～14 行中，结点 x 也决定 $v^*(y)$ 并更新它对目的地 y 的转发表。

前面讲过 LS 算法是一种全局算法，在于它要求每个结点在运行 Dijkstra 算法之前，首先获得该网络的完整信息。DV 算法是分布式的，它不使用这样的全局信息。实际上，结点具有的唯一信息是它到直接相连邻居的链路费用和它从这些邻居接收到的信息。每个结点等待来自任何邻居的更新（第 10～11 行），当接收到一个更新时计算它的新距离向量（第 14 行）并向它的邻居分布其新距离向量（第 16～17 行）。许多类似 DV 的算法在实践中被用于多种路由选择协议中，包括因特网的 RIP 和 BGP、ISO IDRP、Novell IPX 和早期的 ARPAnet。

图 4-30 举例说明了 DV 算法的运行，应用场合是该图顶部有三个结点的简单网络。算法操作以同步的方式显示出来，其中所有结点同时从其邻居接收报文，计算其新距离向量，如果距离向量发生了变化则通知其邻居。学习完这个例子后，你应认识到算法以异步方式也能正确运行，异步方式中可在任意时刻出现结点计算与更新的产生/接收。

该图最左边一列显示了这 3 个结点各自的初始**路由选择表**（routing table）。例如，位于左上角的表是结点 x 的初始路由选择表。在一张特定的路由选择表中，每行是一个距离向量——特别是每个结点的路由选择表包括了它的距离向量和它的每个邻居的距离向量。因此，在结点 x 的初始路由选择表中的第一行是 $\boldsymbol{D}_x = [D_x(x),\ D_x(y),\ D_x(z)] = [0,\ 2,\ 7]$。在该表的第二和第三行是最近分别从结点 y 和 z 收到的距离向量。因为在初始化时结点 x 还没有从结点 y 和 z 收到任何东西，所以第二行和第三行表项中被初始化为无穷大。