5.3.4 附录 E 最短路径算法——Dijkstra 算法

在路由选择算法中都要用到求最短路径算法。最出名的求最短路径算法有两个，即 Bellman-Ford 算法和 Dijkstra 算法。这两种算法的思路不同，但得出的结果是相同的。我们在下面只介绍 Dijkstra 算法，它的已知条件是整个网络拓扑和各链路的长度。

应注意到，若将已知的各链路长度改为链路时延或费用，这就相当于求任意两结点之间具有最小时延或最小费用的路径。因此，求最短路径的算法具有普遍的应用价值。

下面以图 E-1 的网络为例来讨论这种算法，即寻找从源结点到网络中其他各结点的最短路径。为方便起见，设源结点为结点 1。然后一步一步地寻找，每次找一个结点到源结点的最短路径，直到把所有的点都找到为止。

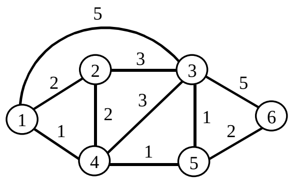


图 E-1 求最短路径算法的网络举例

令 $D(v)$为源结点(记为结点 1)到某个结点 v 的距离，它就是从结点 1 沿某一路径到结点 v 的所有链路的长度之和。再令 $l(i, j)$为结点 i 至结点 j 之间的距离。整个算法只有以下两个部分：

(1) 初始化

令 N 表示网络结点的集合。先令 $N = \{1\}$。对所有不在 N 中的结点 v，写出

$$D(v)=\begin{cases} l(1,v) & \text{若结点}v\text{与结点1直接相连} \\ \infty & \text{若结点}v\text{与结点1不直接相连} \end{cases}$$

在用计算机进行求解时，可以用一个比任何路径长度大得多的数值代替∞。对于上述例子，可以使 $D(v) = 99$。

(2) 寻找一个不在 N 中的结点 w，其 $D(w)$值为最小。把 w 加入到 N 中。然后对所有不在 N 中的结点 v，用$[D(v), D(w) + l(w, v)]$中的较小的值去更新原有的 $D(v)$值，即：

$$D(v) \leftarrow \text{Min}[D(v), D(w) + l(w, v)] \tag{E-1}$$

(3) 重复步骤(2)，直到所有的网络结点都在 N 中为止。

表 E-1 是对图 E-1 的网络进行求解的详细步骤。可以看出，上述的步骤(2)共执行了 5 次。表中带圆圈的数字是在每一次执行步骤(2)时所寻找的具有最小值的 $D(w)$ 值。当第 5 次执行步骤(2)并得出了结果后，所有网络结点都已包含在 N 之中，整个算法即告结束。

表 E-1 **计算图 E-1 的网络的最短路径**

步骤	N	$D(2)$	$D(3)$	$D(4)$	$D(5)$	$D(6)$
初始化	{1}	2	5	1	∞	∞
1	{1, 4}	2	4	①	2	∞
2	{1, 4, 5}	2	3	1	②	4
3	{1, 2, 4, 5}	②	3	1	2	4
4	{1, 2, 3, 4, 5}	2	③	1	2	4
5	{1, 2, 3, 4, 5, 6}	2	3	1	2	④

现在我们对以上的最短路径树的找出过程进行一些解释。

因为选择了结点 1 为源结点，因此一开始在集合 N 中只有结点 1。结点 1 只和结点 2, 3 和 4 直接相连，因此在初始化时，在 $D(2)$，$D(3)$和 $D(4)$下面就填入结点 1 到这些结点相应的距离，而在 $D(5)$和 $D(6)$下面填入∞。

下面执行步骤 1。在结点 1 以外的结点中，找出一个距结点 1 最近的结点 w，这应当是 $w = 4$，因为在 $D(2)$，$D(3)$和 $D(4)$中，$D(4) = 1$，它的之值最小。于是将结点 4 加入到结点集合 N 中。这时，我们在步骤 1 这一行和 $D(4)$这一列下面写入①，数字 1 表示结点 4 到结点 1 的距离，数字 1 的圆圈表示结点 4 在这个步骤加入到结点集合 N 中了。

接着就要对所有不在集合 N 中的结点（即结点 2, 3, 5 和 6）逐个执行（E-1）式。

对于结点 2，原来的 $D(2) = 2$。现在 $D(w) + l(w, v) = D(4) + l(4, 2) = 1 + 2 = 3 > D(2)$。因此结点 2 到结点 1 距离不变，仍为 2。

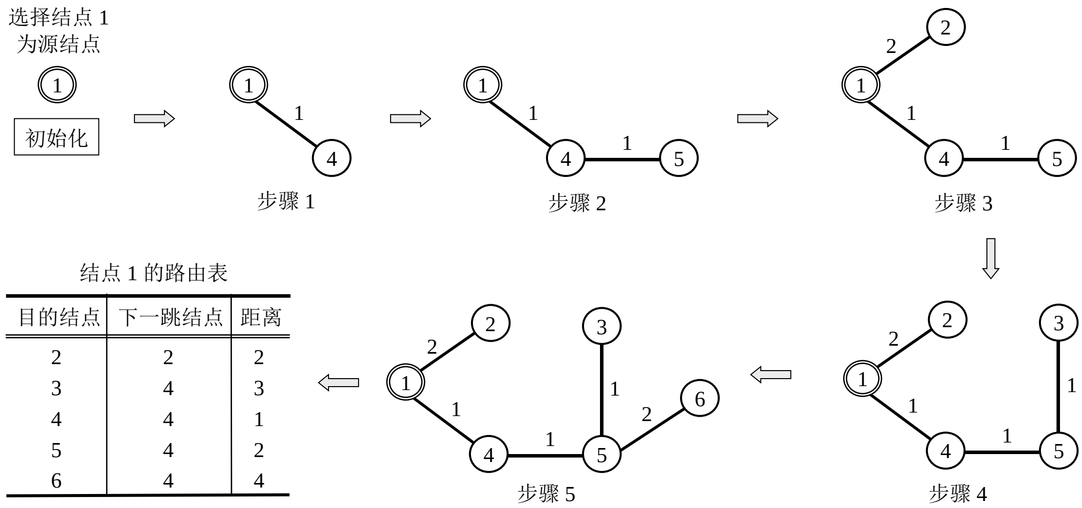
对于结点 3，原来的 $D(3) = 5$。现在 $D(w) + l(w, v) = D(4) + l(4, 3) = 1 + 3 = 4 < D(3)$。因此结点 3 到结点 1 的距离要更新，从 5 减小到 4。

对于结点 5，原来的 $D(5) = \infty$。现在 $D(w) + l(w, v) = D(4) + l(4, 5) = 1 + 1 = 2 < D(5)$。因此结点 5 到结点 1 的距离要更新，从∞减小到 2。

对于结点 6，现在到结点 1 的距离仍为∞。

步骤 1 的计算到此就结束了。

下面执行步骤 2。在结点 1 和 4 以外的结点中，找出一个距结点 1 最近的结点 w。现在有两个结点（结点 2 和 5）到结点 1 的距离一样，都是 2。我们选择结点 5（当然也可以选择结点 2，最后得出的结果还是一样的）。以后的详细步骤这里就省略了，读者可以自行完成剩下的步骤。



结点 1 的路由表

目的结点	下一跳结点	距离
2	2	2
3	4	3
4	4	1
5	4	2
6	4	4

图 E-2　用 Dijkstra 算法求出最短路径树的各个步骤和在结点 1 的路由表

最后就得出以结点 1 为根的最短路径树。图 E-2 给出了各步骤执行后的结果。从最短路径树可清楚地找出从源结点(结点 1)到网内任何一结点的最短路径。图 E-2 还给出了在结点 1 的路由表。此路由表指出对于发往某个目的结点的分组，**从结点 1 发出后的下一跳结点**（在算法中常称为“后继结点”）和距离。当然，像这样的路由表，在所有其他各结点中都有一个。但这就需要分别以这些结点为源结点，重新执行算法，然后才能找出以这个结点为根的最短路径树和相应的路由表。