5.3 最短路由算法

通信工程学院 信息科学研究所

最短路由

- 许多实际的路由算法都是基于最短路径这一概念。
- 这里首先要明确最短的含义，它取决于对链路长度的定义。长度通常是一个正数，它可以是物理距离的长短、时延的大小、各个节点队列长度等等。如果长度取1，则最短路由即为最小跳数（中转次数）的路由。其次，链路的长度随着时间可能是变化的，它取决于链路拥塞情况。
- 最短路由算法的理论基础是图论。

BWC
Xidian Univ.

图论复习

- 每一个网络都可以抽象成一个图。一个图*G*由一个非空的节点集合*N*和节点间的链路A组成，即 $G=(N,A)$ 。
- 链路可以是有方向的，也可以是无方向的。如果节点*i*和*j*之间仅有 $i \to j$ 的链路，则称该链路是有方向的（或单向链路）。如果节点*i*和*j*之间同时有 $i \to j$ 及 $j \to i$ 的链路，则称该链路是无方向的（或双向链路）。
- 方向图与无方向图

- 关联（Incident）：它表示链路与节点的关系。
- 方向性行走（Walk）：是一个节点的序列，该序列中关联的链路，是*G*中的一个链路。
- 方向性Path：指无重复节点的方向性Walk。
- 方向性环（Cycle）：指开始节点和目的节点相同的方向性Path。
- 强连通方向图：指对于每一对节点*i*，*j*都有一条方向性路径。
- 连通的方向图：指如果方向图对应的无方向图是连通的，则该方向图是连通的。

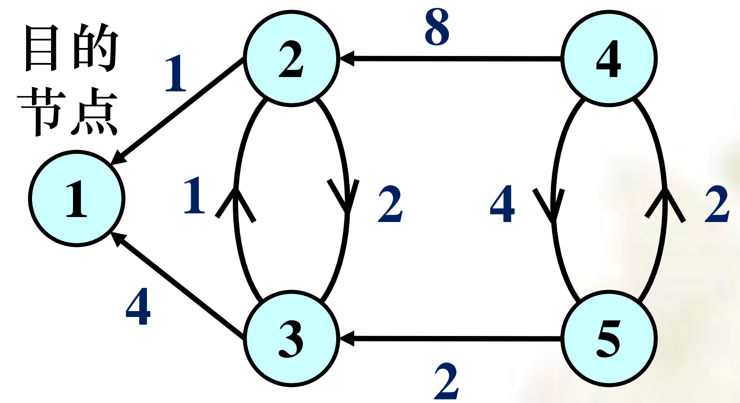
- 给每条链路指定一个实数作为其长度，则一条方向性路径 $p=(i,j,k,\ldots,l,m)$ 的长度就是各链路长度之和，即 $d_{ij}+d_{jk}+\cdots+d_{lm}$ 。
- **最短路径问题**就是寻找从*i*到*m*的最小长度方向性路径。
- 根据长度的不同定义，寻找最短路径的算法有不同的含义。

集中式最短路径算法

- 讨论三种标准的集中式最短路径算法:
  - Bellman-Ford算法
  - Dijkstra算法
  - Floyd-Warshall算法。
- **Bellman-Ford**算法和**Dijkstra**算法是点对多点的最短路径算法
- **Floyd-Warshall**算法则是多点对多点的最短路径算法。

1. Bellman-Ford算法

- 典型的Bellman-Ford算法（简记为**B-F算法**）是一种**集中式**的**点到多点**的**路由算法**，即寻找网络中一个节点到其它所有节点的路由。
- 假定节点1是“**目的节点**”，我们要寻找网络中其它所有的节点到目的节点1的最短路径。
- 用 d_{ij} 表示节点*i*到节点*j*的长度。



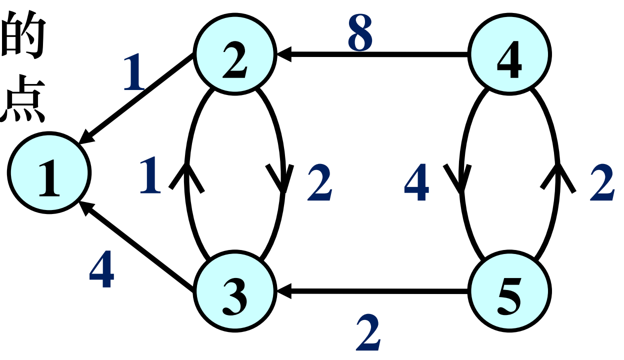
- **定义：最短（$\le h$）行走（Walk）是指在下列约束条件下从给定节点*i*到目的节点1的最短Walk。**
  - ① 该行走（Walk）中最多包括*h*条链路，即Walk中包含的链路数至多为*h*条。
  - ② 该行走（Walk）仅经过目的节点1一次。
- **最短（$\le h$）行走Walk长度用 D_i^h 表示。**
- **对所有的*h*，令$D_1^h = 0$。B-F算法的核心思想是通过下面的公式进行迭代，即**

$$D_i^{h+1} = \min_j [d_{ij} + D_j^h] \qquad i \ne 1$$

- 下面给出从*h*步Walk中寻找最短路由的算法。
  - 第一步：初始化。即对所有*i* $(i \neq 1)$ 令 $D_i^0 = \infty$。
  - 第二步：对所有的节点*j*（$j \neq i$），先找出使用一条链路的最短（$h \leq 1$）的Walk长度；
  - 第三步：对所有的节点*j*（ $j \neq i$ ），再找出使用两条链路的最短（$h \leq 2$ ）的Walk长度；
  - 依次类推：如果对所有*i*有：$D_i^h = D_i^{h-1}$（即继续迭代下去以后不会再有变化），则算法在*h*次迭代后结束。

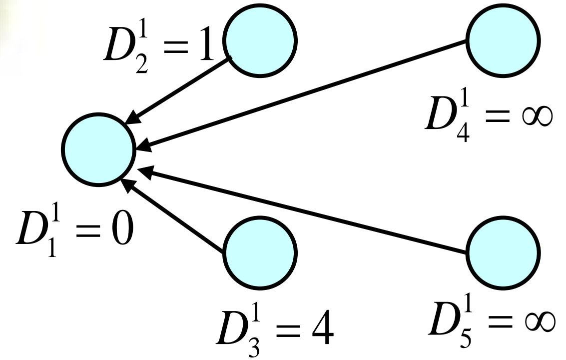
BWC Xidian Univ.

$$D_i^{h+1} = \min_j [d_{ij} + D_j^h] \qquad i \neq 1$$

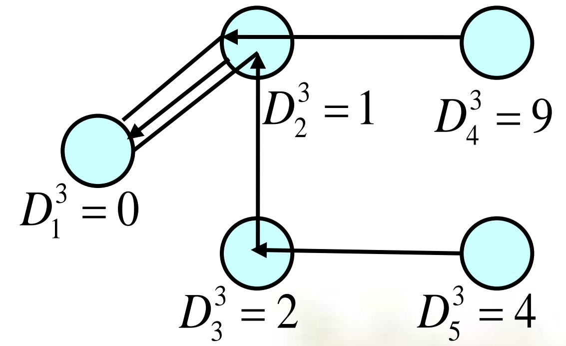


最短路径问题：链路长度如图中的标定所示

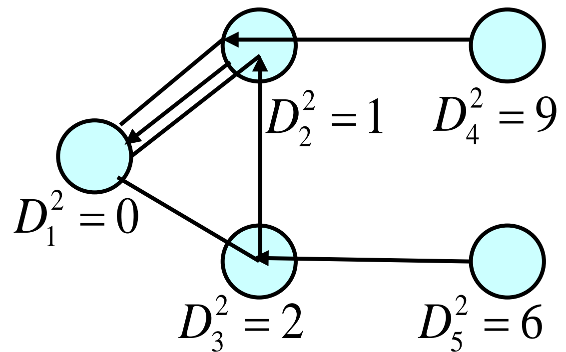
例5.2 请描述图5-8中节点4到节点1的路由迭代过程。



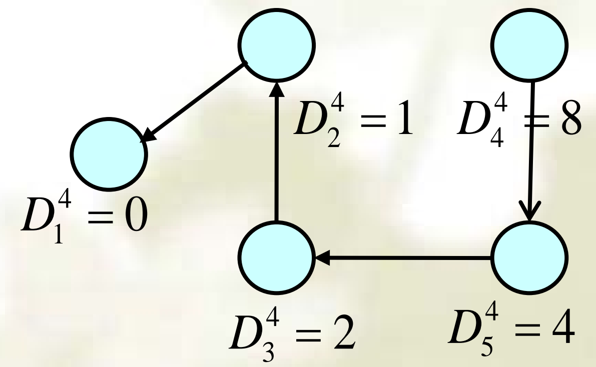
(b) 最多使用1条链路的最短路径



(d) 最多使用3条链路的最短路径



(c) 最多使用2条链路的最短路径



(e) 最终的最短路径

- 最短Walk长度等于最短路径长度的充分必要条件

定理1：对于式（5-1）的B-F算法（初始条件：对所有$i \neq 1$，有$D_i^0 = \infty$），有：

- (1) 由该算法产生的D_i^h等于最短（$\leq h$）Walk长度
- (2) 当且仅当所有不包括节点1的环具有非负的长度，算法在有限次迭代后结束。此外，如果算法在最多$k \leq N$次迭代后结束，则结束时D_i^h就是从*i*到1的最短路径长度。

- 定理1中(1)阐明了与最短（$\le h$）**Walk**的关系。(2)阐明了算法何时结束，结束时所得的结果是否是最短路径。

证明：我们采用归纳法证明（1）。

- ① 因为 $D_i^1 = d_{i1}$，所以显然有 D_i^1 等于最短（≤ 1）的**Walk**长度；
- ② 假定 D_i^h 是等于最短（$\le h$）的**Walk**长度，求证 D_i^{h+1} 是等于最短（$\le h+1$）的**Walk**长度。

证明：

- 从*i*到1的最短（$\le h+1$）**Walk**包含的链路数有两种情况：一种情况是链路数小于**h+1**，在此情况下，有**Walk**长度等于 D_i^h；另一种情况是链路数等于**h+1**。综上，有

最短（$\le h+1$）**Walk**长度 $= \min\{D_i^h, D_i^{h+1}\} = \min\{D_i^h, \min_{j\neq 1}[d_{ij} + D_j^h]\}$

- 根据 D_i^h 等于最短（$\le h$）**Walk**的假设，对所有的 $k \le h$ 有 $D_j^k \le D_j^{k-1}$。因此有

$$D_i^{h+1} = \min_j[d_{ij} + D_j^h] \le \min_j[d_{ij} + D_j^{h-1}] = D_i^h$$

最短（$\le h+1$）**Walk**长度 $= D_i^{h+1}$

证明：

- （2）如果B-F算法在*h*次迭代后结束，即有

$$D_i^k = D_i^h \quad \text{对所有i和} \quad k \geq h \qquad （5\text{-}5）$$

- 则我们不可能通过添加更多的链路来减少最短的Walk长度。（否则，算法没有结束。）
- 也就是不可能存在一个负长度的（不包括目的节点）环。因为这样的负长度的任意大次数的重复将使Walk的长度任意的小，这与式（5-5）相矛盾。

证明：

- 相反，假定所有的不包括1的环具有非负的长度。从最短（$\leq h$）Walk中删除这些环，我们会得到长度相同或更短的路径。
- 因此，对每一个*i*和*h*，总存在一条从*i*到1的最短（$\leq h$）Walk，其相应的最短的路径长度等于 D_i^h。
- 由于路径中没有环，路径可能包括最多*N-1*条链路。因此 $D_i^N = D_i^{N-1}$，对所有*i*成立。即算法在最多*N*次迭代后结束。

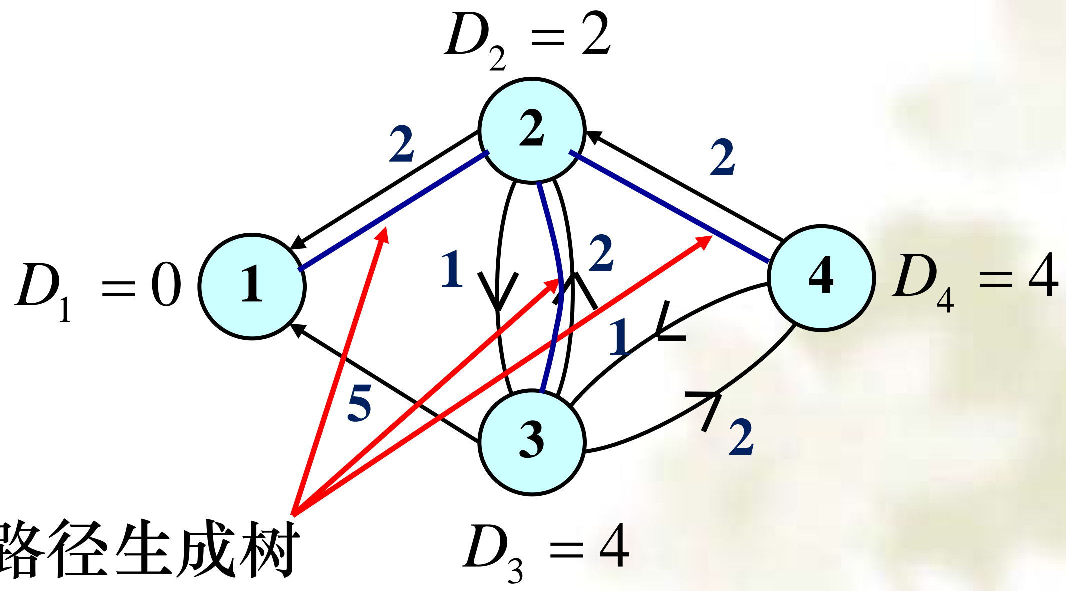
最短路与Bellman方程

- 假定所有不包括节点1的环具有非负的长度，用D_i表示从节点i到达目的节点1的最短路径长度。根据前面的讨论，当B-F算法结束时，有

$$D_i = \min_j [d_{ij} + D_j] \quad 对所有 i \neq 1 \qquad (5\text{-}6)$$
$$D_1 = 0$$

- 该式称为**Bellman方程**。它表明从节点i到达目的节点1的最短路径长度，等于i到达该路径上第一个节点的链路长度，加上该节点到达目的节点1的最短路径长度。从该方程出发，只要所有不包括1的环具有正的长度（而不是0长度）的情况下，我们就可以很容易地找到最短路径（而不是最短路径长度）。

- 具体方法如下：
- 对于每一个节点 $i \neq 1$，选择一条满足 $D_i = \min_j [d_{ij} + D_j]$ 的最小值的链路 (i, j_i)，利用这些**N-1**条链路组成一个子图，则*i* 沿该子图到达目的节点**1**的路径即为最短路径。



- 利用上面的构造方法，可以证明：如果没有0长度（或负长度）的环，则Bellman方程式（5-6）（它可以看成一个含有*N*个未知数的*N*个方程的系统）有唯一解。（如果有不包括节点1的环的长度为0，则Bellman方程不再具有唯一解。注意：路径长度唯一，并不意味着路径唯一。）
- 利用该结论，可以证明：即使初始条件 D_i^0，$i \neq 1$ 是任意数（而不是 $D_i^0 = \infty$），B-F算法都能正确工作，对于不同的节点，迭代的过程可以以任意顺序并行进行。

2. Dijkstra算法

- Dijkstra算法也是一种典型的点对多点的路由选择算法，即通过迭代，寻找某一节点到网络中其它所有节点的最短路径。
- Dijkstra算法通过对路径的长度进行迭代，从而计算出到达目的节点的最短路径。
- 其基本思想是按照路径长度增加的顺序来寻找最短路径。

- 假定所有链路的长度均为非负。显然有：到达目的节点1的最短路径中最短的肯定是节点1的最近的邻节点所对应的单条链路。
- 由于链路长度非负，所以任何多条链路组成的路径的长度都不可能短于第一条链路的长度。
- 最短路径中下一个最短的肯定是节点1的下一个最近的邻节点所对应的单条链路，或者是通过前面选定的节点的最短的有两条链路组成的路径，依次类推。

- Dijkstra算法通过逐步标定到达目的节点路径长度的方法来求解最短的路径。
- 设每个节点*i*标定的到达目的节点1的最短路径长度估计为D_i。如果在迭代的过程中，D_i已变成一个确定的值，称节点*i*为永久标定的节点，这些永久标定的节点的集合用***P***表示。
- 在算法的每一步中，在***P***以外的节点中，必定是选择与目的节点1最近的节点加入到集合***P***中。

- 具体的**Dijkstra**算法如下：
- ① 初始化，即 $P=\{1\}$，$D_1=0$，$D_j=d_{j1}$，$j\neq 1$ 。（如果 $(j,1)\notin A$ ，则 $d_{j1}=\infty$ ）。
- ② 寻找下一个与目的节点最近的节点，即求使下式成立的*i*，$i\notin P$

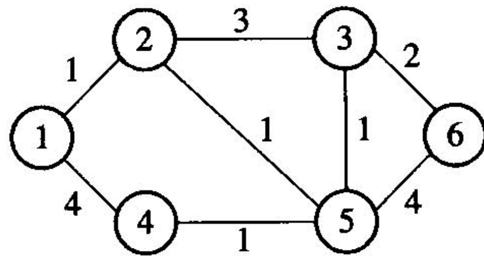
$$D_i=\min_{j\notin P} D_j$$

置 $P=P\cup\{i\}$。如果***P***包括了所有的节点，则算法结束。

- ③ 更改标定值，即对所有的 $j\notin P$，置

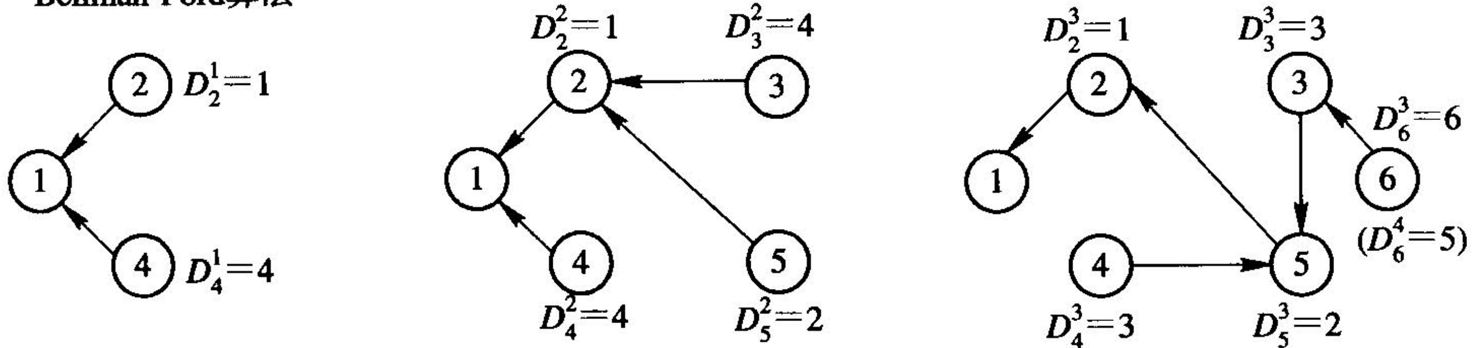
$$D_j=\min_i[D_j, d_{ji}+D_i]$$

返回第②步。



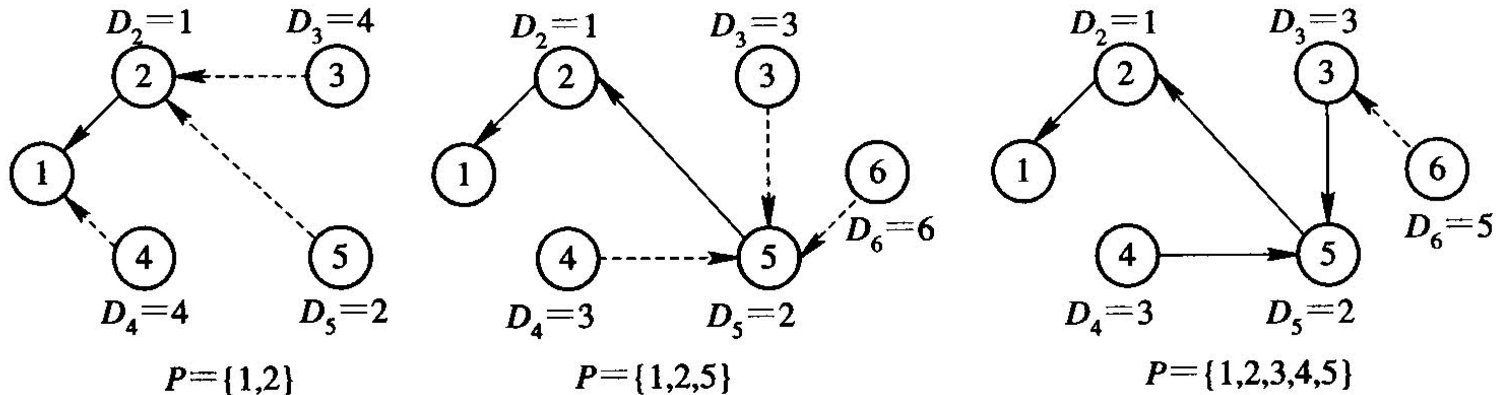
(a)

Bellman-Ford算法



(b)

Dijkstra算法



(c)

BWC Xidian Univ.

- 在图中，还给出了B-F算法的迭代过程。很显然，在最坏的情况下，Dijkstra算法的复杂度为$O(N^2)$，而B-F算法的复杂度为 $O(N^3)$。即Dijkstra算法的复杂度要低于B-F算法。同时，从Dijkstra算法的讨论过程中，我们可以看到：
- ① $D_i \le D_j$，对所有$i \in P, j \notin P$ 。
- ② 对于每一个节点*j*，D_j是从*j*到目的节点1的最短距离。该路径使用的所有节点（除*j*以外）都属于*P*。

$$D_j = \min_i [D_j, d_{ji} + D_i]$$

3. Floyd-Warshall算法（F-W算法）

- B-F算法和Dijkstra算法都是求解所有节点到一个特定的目的节点之间的最短路径，F-W算法则是多点对多点的路由选择算法。F-W算法是寻找所有节点对之间的最短路径。其基本思想是在 $i \rightarrow j$ 的路径之间通过添加中间节点来减小路径长度。
- 在F-W算法中，假定链路的长度可以是正或负，但不能具有负长度的环。
- F-W算法开始时，以单链路（无中间节点）的距离作为最短路径的估计。然后，在仅允许节点1作为中间节点的情况下，计算最短路径，接着，在允许节点1和节点2作为中间节点的情况下计算最短距离，依次类推。

- 其具体描述如下：
- 令 D_{ij}^{n} 是可以用**1**、**2…*n***作为中间节点的从***i***到***j***的最短路径长度，则算法开始时 $D_{ij}^{0}=d_{ij}$，对所有***i***，***j***，$i \neq j$。
- 对于 $n=0,1,\ldots,N-1$，有

$$D_{ij}^{n+1}=\min[D_{ij}^{n},D_{i(n+1)}^{n}+D_{(n+1)j}^{n}] \quad \text{对所有} i \neq j$$

- 上式是已知***i***到***j***的最短路径 D_{ij}^{n}（以**1**，**2…*n***作为中间节点）的条件，如何计算在***i***到***j***的最短路径上可添加节点***n*+1**后的最短路径长度。

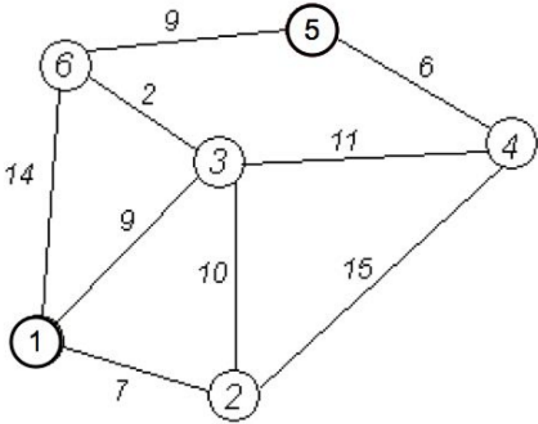
$$D_{ij}^{n+1} = \min[D_{ij}^{n}, D_{i(n+1)}^{n} + D_{(n+1)j}^{n}] \quad 对所有 i \neq j$$

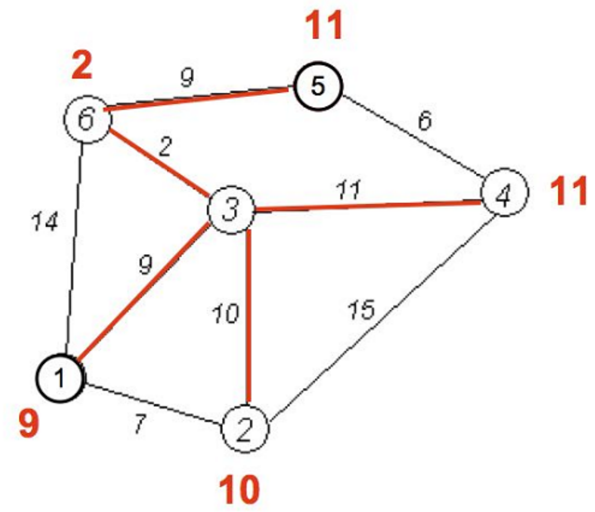
- 在允许添加节点**n**+1的情况下，有两种可能性：
- 一种是最短的路径将包含节点**n**+1，此时的路径长度为 $D_{i(n+1)}^{n} + D_{(n+1)j}^{n}$
- 另一种可能是节点**n**+1不包括在最短路径中，此时路径长度等同于用1，2…**n**作为中间节点的路径长度。因此，最终的最短路径长度应取上述两种可能情况下的最小值，即有式（5-9）成立。
- **F-W**算法的计算复杂度和**B-F**算法的一样，都是 $O(N^3)$ 。

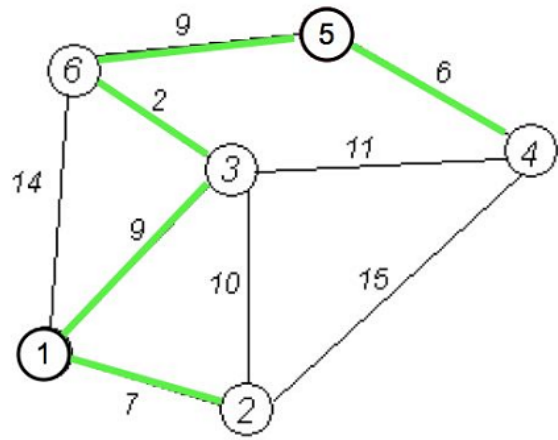
- 前面讨论的三种最短路由算法的构造方法，都是通过迭代的过程求得最终结果。但其主要差别是迭代的内容不同。
- 在B-F算法中，迭代的是路径中的链路数，即使用一条，两条，…，直到N-1条链路。
- 在Dijkstra算法中迭代的是路径的长度，即最短长度，次短长度，…。
- 而在F-W算法中，是对路径的中间节点进行的迭代，即一个中间节点，两个中间节点，…。

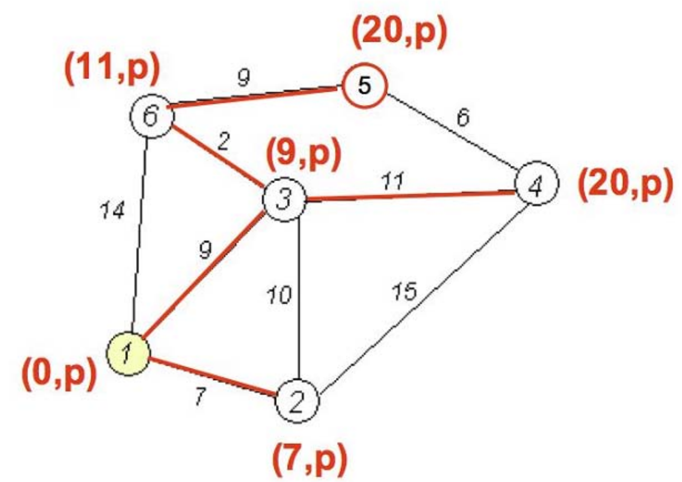
BWC Xidian Univ.

MST and SPT









分布式最短路径算法

- **集中式**路由算法是指网络的路由是由**路由控制中心**计算的，该中心周期性收集各链路的状态，经过路由计算后周期性地向各网络节点提供路由表。
- **分布式**路由是指网络中所有节点通过相互交换路由信息，独立地计算到达各节点的路由。

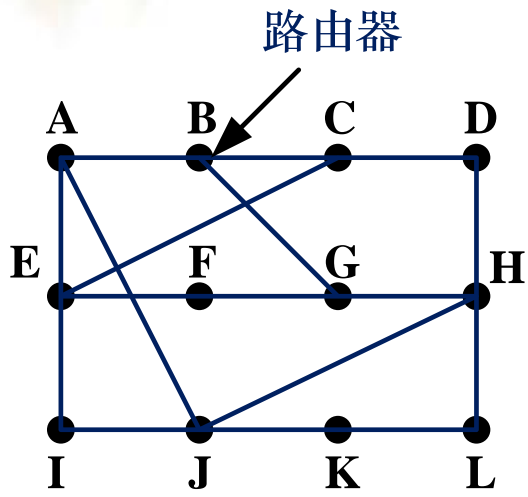
BWC
Xidian Univ

分布式最短路径算法

- 这种路由选择策略是每个节点周期性的从相邻的节点获得网络状态信息，同时也将本节点做出的决定周期性的通知周围的各节点，以使这些节点不断的根据网络新的状态更新其路由选择。
- 整个网络的路由选择经常处于一种动态变化的状态。各个节点的路由表相互作用，是这种路由选择算法的特点。
- 当网络状态发生变化时，必然会影响到许多节点的路由表。因此，要经过一定的时间以后，各路由表中的数据才能达到稳定的数值。
- 分布式路由选择算法的核心思想是各个节点独立的计算最短路径。
- 典型的分布式最短路径选择算法有距离矢量路由算法和链路状态路由算法。

一．距离矢量路由算法

- 距离矢量路由算法（Distance Vector Routing）算法是B-F算法的具体实现。
- 在距离矢量路由表中，每个路由器维护一张路由表，该表中记录了到网络中其它所有节点的路由信息。包括
  - 到该目的节点的下一跳节点（即本节点通过哪个邻节点到达指定的目的节点），
  - 到达该目的节点所需的“距离”的估计值。



目的	A	I	H	K
A	0	24	20	21
B	12	36	31	28
C	25	18	19	36
D	40	27	8	24
E	14	7	30	22
F	23	20	19	40
G	18	31	6	31
H	17	20	0	19
I	21	0	14	22
J	9	11	7	10
K	24	22	22	0
L	29	33	9	9
	JA 时延	JI 时延	JH 时延	JK 时延
	8	10	12	6

从J的4个邻节点接收到的矢量

J到各节点新估计的时延

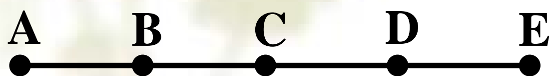
到达目的节点路径上的第一个中继节点

8	A
20	A
28	I
20	H
17	I
30	I
18	H
12	H
10	I
0	-
6	K
15	K

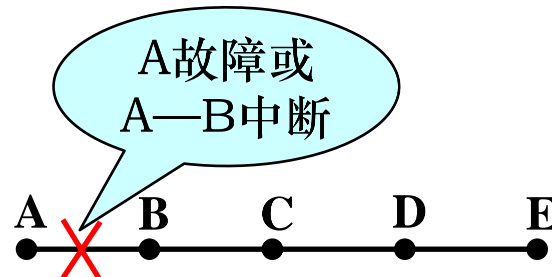
J的新路由表

- 在表中使用的距离量度可以是跳数、时延、某一路径排队的总分组数或者其它类似的量度。每一个节点都确知它的每一个邻节点的距离。
- 如果采用时延作为距离的量度，每个节点应当能够利用一个特殊的“回声”（**ECHO**）分组来直接测量该时延。接收节点收到“回声”分组后，只对它加上时间标记后就立即送回。
- 以时延作为距离的量度。每隔***T***秒，每个节点向它的每个邻节点发送一个路由信息分组，该分组包括了发送节点已知的目的节点的下一跳节点和时延估计值。同样，每一个节点都会收到它所有的邻节点发送来的路由信息分组。

- 1. 计数至无穷问题
- 距离矢量路由算法在理论上是可以正常工作的，但在实际运用中却有很大的缺陷。虽然它能得出正确的结论，但有可能太慢。特别时，它对好消息的反应迅速，但对坏消息却反应迟钝。假定一个节点*i*到达一个目的节点X的最短距离很大。如果下一次收到的路由信息分组中，A突然报告它到目的节点X的时延很短，则*i*会立即将最短路由切换到通过A的链路去往目的节点X。即通过一次信息交换，好信息即被处理。



∞	∞	∞	∞	初始值
1	∞	∞	∞	1次交换后
1	2	∞	∞	2次交换后
1	2	3	∞	3次交换后
1	2	3	4	4次交换后

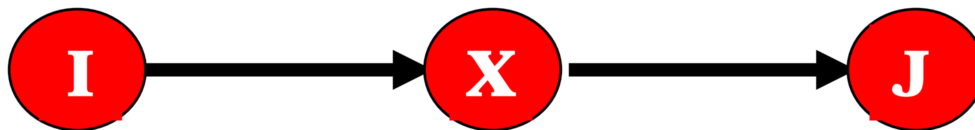


1	2	3	4	初始值
3	2	3	4	1次交换后
3	4	3	4	2次交换后
5	4	5	4	3次交换后
5	6	5	6	4次交换后
7	6	7	6	5次交换后
7	8	7	8	6次交换后
⋮	⋮	⋮	⋮	
∞	∞	∞	∞	

图5-12　计数至无穷问题举例

- 讨论坏消息的传播速度。
  - 从图中可以看出，坏消息传播很慢，没有一个节点会将其距离设置成大于邻节点报告的最小距离值加1，所有的节点都会逐步地增加其距离值，直至无穷大。该问题称为“计数至无穷问题”（count to infinity）问题或“坏消息现象”（bad news phenomenon）。在实际系统中，我们可以将无穷大设置为网络的最大跳数加1。但是当采用时延作为距离的长度时，将很难定义一个合适的时延上界。该时延的上界应足够大，以避免将长时延的路径认为是故障的链路。

- 2. 水平分裂算法
- 理论上已经提出了许多解决计数至无穷问题的办法。但这些办法都比较复杂。这里我们介绍一种“水平分裂算法”（Split Horizon）。水平分裂算法与距离矢量算法的工作过程基本一样，不同之处在于如果节点*I*到达某一目的节点*J*的距离是通过节点*X*得到的，则节点*I*将不会向节点*X*报告有关节点*J*的信息（即节点*I*向节点*X*报告的到达节点*J*的距离为无穷大）。



- 平分割法虽然能够解决一些计数至无穷问题，但有时候也不能正常工作。
  - 初始化A、B到D的距离为2，到C的距离为1
  - 水平分裂法，A和B都会告诉C不可达D。因此，C确定D不可达。
  - A听到B有一条到D的长度为2的路径，认为通过B经过3个节点到D。类似B

二. 链路状态路由算法

- 在1979年之前，**ARPANET**都是采用的距离矢量路由算法。之后，就用链路状态路由算法（**Link State Routing**）取代了距离矢量路由算法。其主要原因有两个。
  - 第一，因为在距离矢量算法中，时延的度量是仅仅是队列的长度，而并没有考虑后来链路带宽的增长；
  - 第二，距离矢量算法的收敛速度比较慢，即使是采用了类似于水平分割这样的技术，也需要耗费过多的时间用于记录信息。

- 链路状态路由算法的思想非常简单，它包括以下五个部分：
  - ① 发现邻节点，并获取它们的地址；
  - ② 测量到达每一个邻节点的时延或成本；
  - ③ 构造一个分组来通告它所知道的所有路由信息；
  - ④ 发送该分组到所有其它节点；
  - ⑤ 计算到所有其它节点的最短路径。
- 事实上，完整的拓扑结构和所有的时延都已经分发到网络中的每一个节点。随后，每个节点都可以用**Dijkstra**算法来求得到其它所有节点的最短路径。

- ① 发现邻节点
  - 当一个路由器启动以后，它的第一个任务就是要知道它的邻节点是谁。具体实现的方法是：
    - 该路由器在每一个输出链路上广播一个特殊的Hello分组，在这些链路另一端的路由器将会发送回一个应答分组，告知它是谁。所有路由器的名字（地址）必须是全球唯一的。

- 当两个或多个路由器通过LAN互连时，如图5-14（a）所示，这时我们把LAN看成一个虚拟的节点N，如图5-14（b）所示。这时A到C的路由就可以看成是ANC。

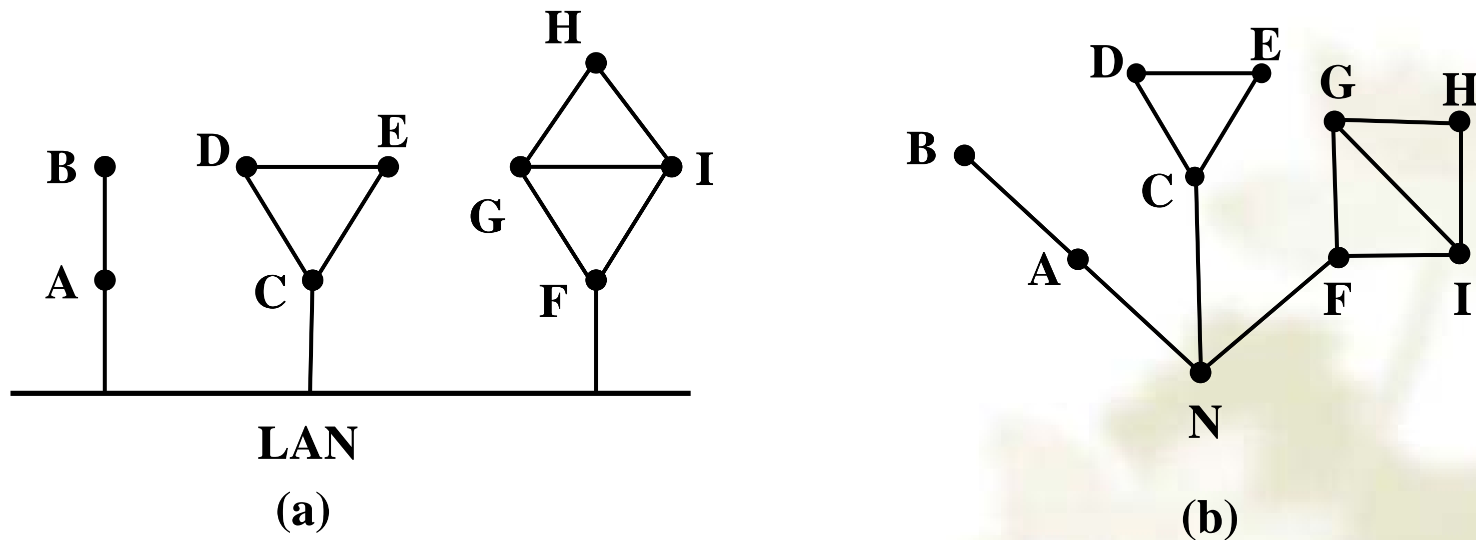
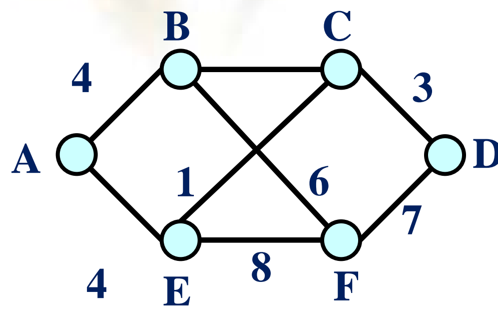


图5-14　节点通过LAN互连时的模型
（a）通过LAN互连的网络　（b）用虚拟节点N来等效的网络

- ② 测量链路时延或成本
  - 链路状态路由算法要求每个路由器确知到达每一个邻节点的时延或对该时延有一个合理的估计。确定该时延的最直接的方法是发送一个特殊的**ECHO**分组给每个邻节点，并要求每个邻节点立即发回该分组。将测量的来回时延除以**2**就可以得到该链路时延的估计。为了得到较好的结果，可测量多次后取平均。
  - 在测量链路时延时，既可以考虑排队的时延（链路的负荷），也可以不考虑排队的时延。考虑排队时延（链路负荷）的优点是可以获得较好的性能，但是可能会引起路由的振荡。

- ③ 构造链路状态分组
  - 每个节点都构造一个自己的链路状态分组，它包括发送节点的标号、该分组的序号和寿命，以及发送节点的邻节点列表及发送节点到这些邻节点的链路时延。
  - 构造链路状态分组是很容易的，困难的是何时构造这些分组。一种方法是周期性地构造这些分组；另一种方法是在链路状态变化（如故障、恢复工作或特性改变）时才构造这些分组。

链路状态分组



(a)

A	
序号	
寿命	
B	4
E	5

B	
序号	
寿命	
A	4
C	2
F	6

C	
序号	
寿命	
B	2
D	3
E	1

D	
序号	
寿命	
C	3
F	7

E	
序号	
寿命	
A	5
C	1
F	7

F	
序号	
寿命	
B	6
D	7
E	8

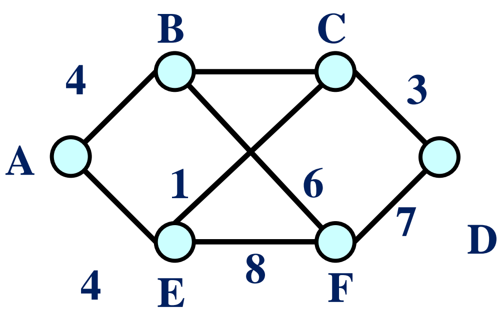
(b)

图5-15　链路状态分组的格式
（a）网络拓扑　（b）链路状态分组的格式

- ④ 分发链路状态分组
  - 该算法的中最具技巧性的部分就是如何可靠的分发链路状态分组。当链路状态分组被发布后，首先得到该分组的路由器将改变其路由选择。同时，别的路由器可能还在使用不同的旧版本的链路信息，这样将导致各节点对当前网络拓扑的看法不一致性，从而计算出的路由可能出现死循环、不可达或其它问题。
  - 链路状态分组分发的最基本方法是采用泛洪（**Flooding**）方式。为防止每个节点处理和中转过时的链路状态分组，在这些分组中引入了序号。每个节点仅中转序号大于已记录的最大序号的分组，为了防止序号出错，在分组中还引入了寿命，寿命每秒递减一次，如果寿命为**0**，则该分组将被丢弃。

- 为了提高传输的可靠性，所有链路状态分组都需要应答。为了处理链路状态分组在泛洪中需要发往哪些邻节点、需要对哪条链路的分组进行应答的问题，每个节点需构造一个分组存储数据结构。该图是图5-15拓扑中节点B的数据结构。图中每一行对应刚刚到达，但还没有完全处理的链路状态分组。由于节点B有三个邻节点A、C和F，所以发送标志指明应当发送给哪个邻节点，应答标志指明应答哪个邻节点。

由于节点B有三个邻节点A、C和F，所以发送标志指明应当发送给哪个邻节点，应答标志指明应答哪个邻节点。



来自C的分组还没有中转，从C发出分组从F到达，将两个分组合并处理。

源节点	序号	寿命	发送标志			ACK标志			数据
			A	C	F	A	C	F	
A	21	60	0	1	1	1	0	0	
F	21	60	1	1	0	0	0	1	
E	21	59	0	1	0	1	0	1	
C	20	60	1	0	1	0	1	0	
D	21	59	1	0	0	0	1	1	

图5-16　链路状态分组的存贮结构（节点B的数据结构）

- ⑤ 计算新的路由
- 当每个节点获得所有的链路状态分组以后，它可以构造一个完整的网络拓扑，此时每个节点就可以运行**Dijkstra**算法来构造到达所有目的的节点的最短路由。
- 链路状态路由算法已广泛用于多种实际网络中，例如**Internet**中的**OSPF**（**open shortest path first**）采用了该算法，**ISO**的无连接网络层协议（**CLNP**）使用的**IS-IS**（**Intermediate System-to-Intermediate System**）协议也是采用该算法。**IS-IS**中交换的信息是用于计算最短路由的网络拓扑图（而不仅仅是链路状态分组），它还可以支持多种网络协议（如**IP**，**IPX**，**AppleTalk**等）。