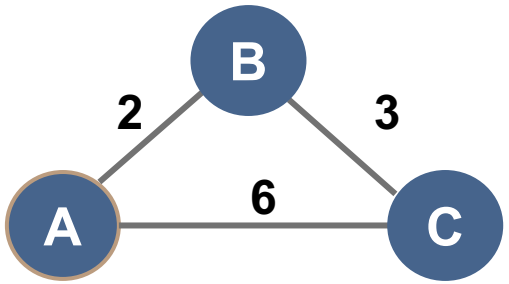
# 距离矢量路由算法

VS.

# 算法过程

# 距离矢量算法的数据结构

| 主要数据结构 | 节点信息资源 | 距离矢量算法 |
| --- | --- | --- |
| • 每个节点维护一张经过所有邻居到全部目的地的距离表（距离矩阵）。 | • 与其直接相连的链路(本地链路)的成本<br>• 来自邻接节点的路由信息 | • 用估算延迟作为性能指标<br>• 基于Bellman-Ford算法 |




① 计算距离矩阵

计算从本地出发到达所有目的地的全部可达路径，需要本地链路信息和邻居的路由信息。

② 生成路由表

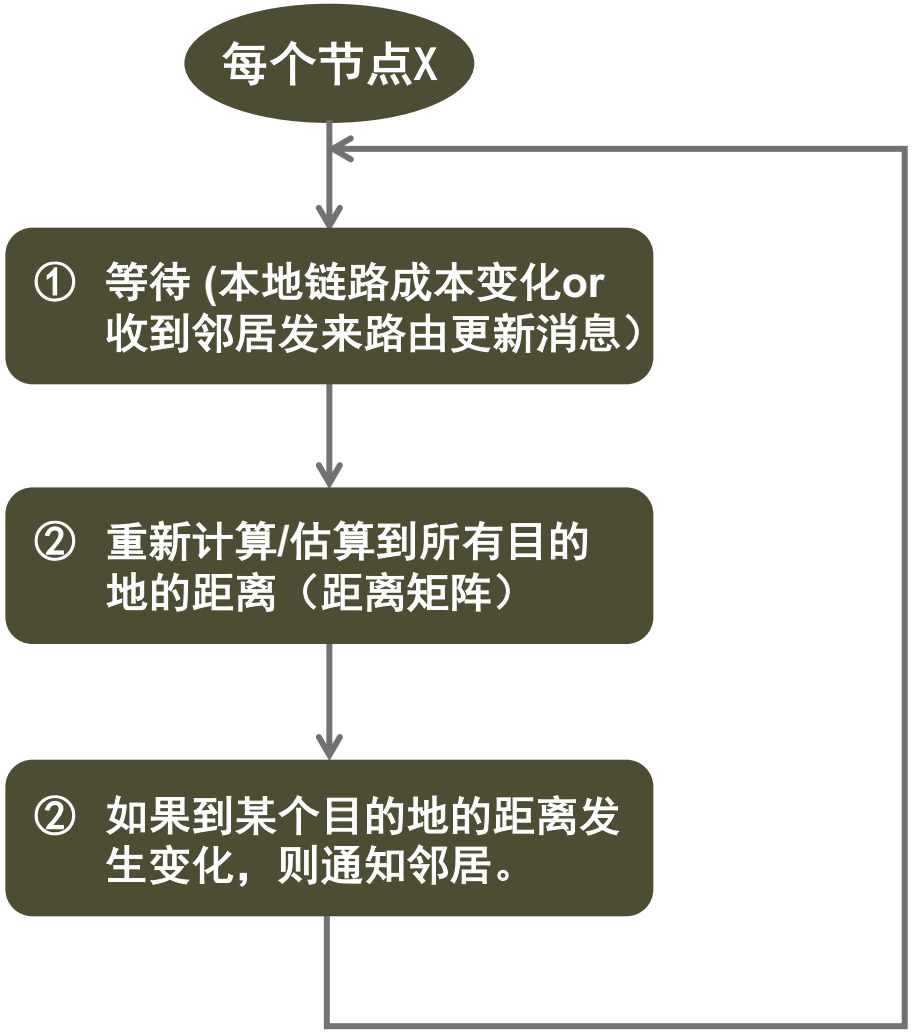
# 距离矢量算法过程

## 迭代的条件

- 本地链路成本发生变化
- 收到邻居发来的距离更新消息

## 迭代更新路由

- 每个节点仅当自己的距离发生变化时才通知邻居
- 如果需要其邻居再通知邻居的邻居



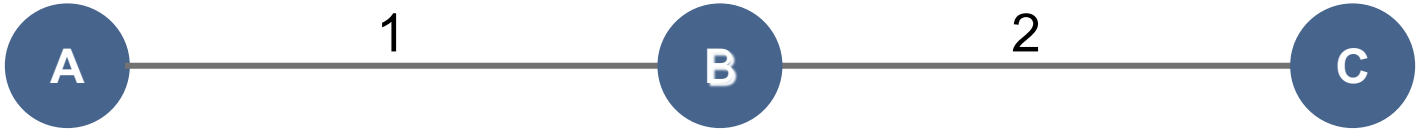
# 距离矢量算法示例

假设：如图所示的网络中路由器A,B,C运行基于距离矢量路由算法的路由协议

考察：A,B,C路由表的变化

- A,B,C各自独立运行路由协议，一旦满足迭代条件就重新计算路由。
- 注意：路由表是根据距离矩阵计算获得的

- t0：路由稳定后的各节点路由表




A路由表

| 目的地. | 下一跳 | 度量 |
|---|---|---|
| A | A | 0 |
| B | B | 1 |
| C | B | 3 |

B路由表

| 目的地. | 下一跳 | 度量 |
|---|---|---|
| A | A | 1 |
| B | B | 0 |
| C | C | 2 |

C路由表

| 目的地. | 下一跳 | 度量 |
|---|---|---|
| A | A | 3 |
| B | B | 2 |
| C | C | 0 |



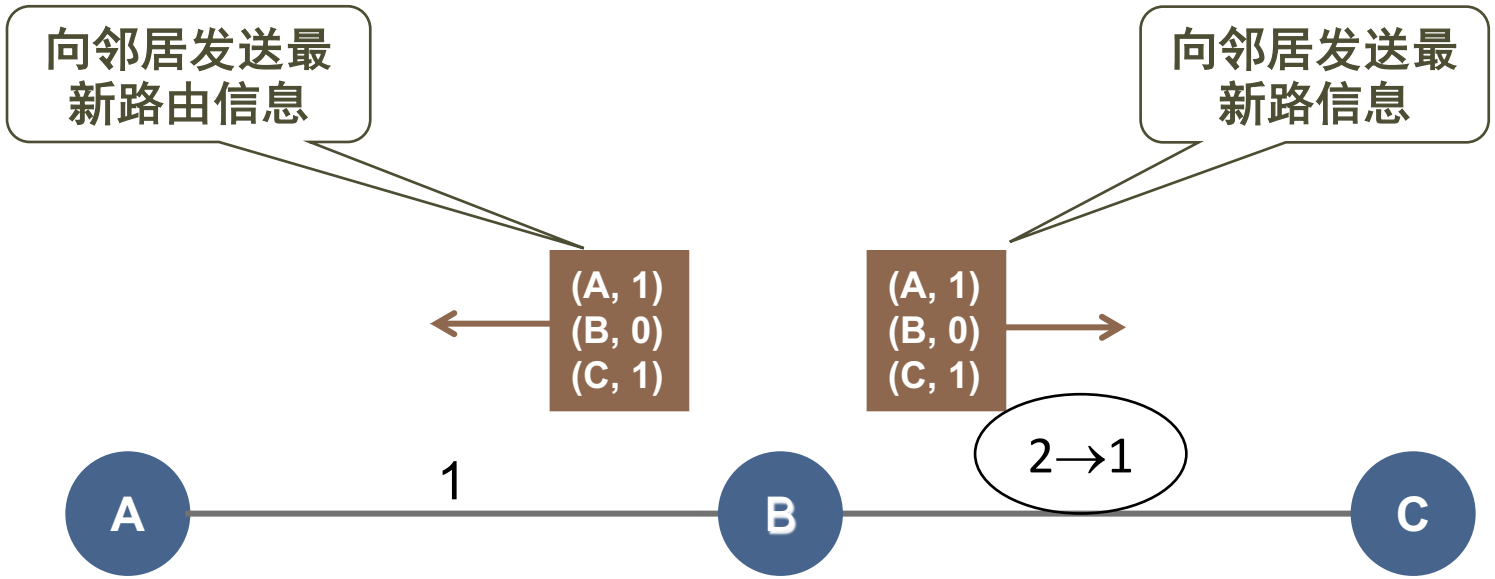
注：From Tobias Schwaegli

# 向邻居发送最新路由信息

- t1：B-C的链路成本发生变化

B重新计算路由，并将计算结果发送给自己的邻居（B,C）。

节点B感知到链路成本发生了变化，重新计算所有经过该条链路出发的路径长度。



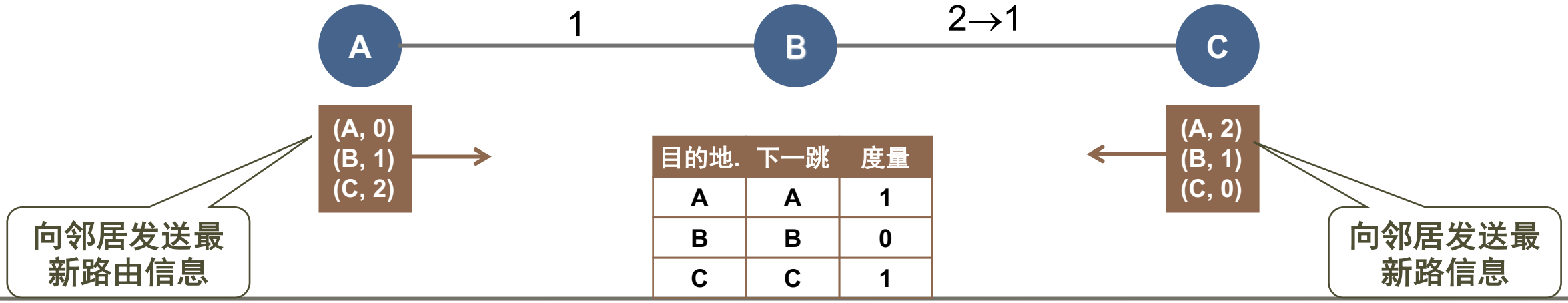



注：From Tobias Schwaegli

# 迭代时重新计算路由

- A和C收到来自B的路由信息后，发现B到目的地C的路径长度发生了变化，于是
  - ① 重新计算路由
  - ② 发现到B的路由有了变化，把最新路由发给邻居（B）

- B收到来自A和C的路由信息后，检查经过邻居到达B和C的路径是否比已有的更好
  - ① A到C的距离是2，加上B-A链路成本1，经过A到达C的路径长度为3，比已有的差，不更改到C的路由
  - ② 同理，B放弃经过C抵达A的路由，维持到A的原路由



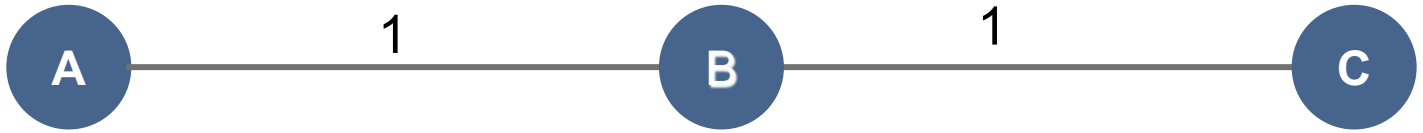



注：From Tobias Schwaegli

# 算法收敛的条件

- t2：路由稳定后的各节点路由表

当所有节点（A, B, C）把最新路由表发送给邻居后，没有导致邻居更新路由，至此路由达到稳定，算法收敛。




A路由表

| 目的地. | 下一跳 | 度量 |
|---|---|---|
| A | A | 0 |
| B | B | 1 |
| C | B | 2 |

B路由表

| 目的地. | 下一跳 | 度量 |
|---|---|---|
| A | A | 1 |
| B | B | 0 |
| C | C | 1 |

C路由表

| 目的地. | 下一跳 | 度量 |
|---|---|---|
| A | A | 2 |
| B | B | 1 |
| C | C | 0 |



注：From Tobias Schwaegli