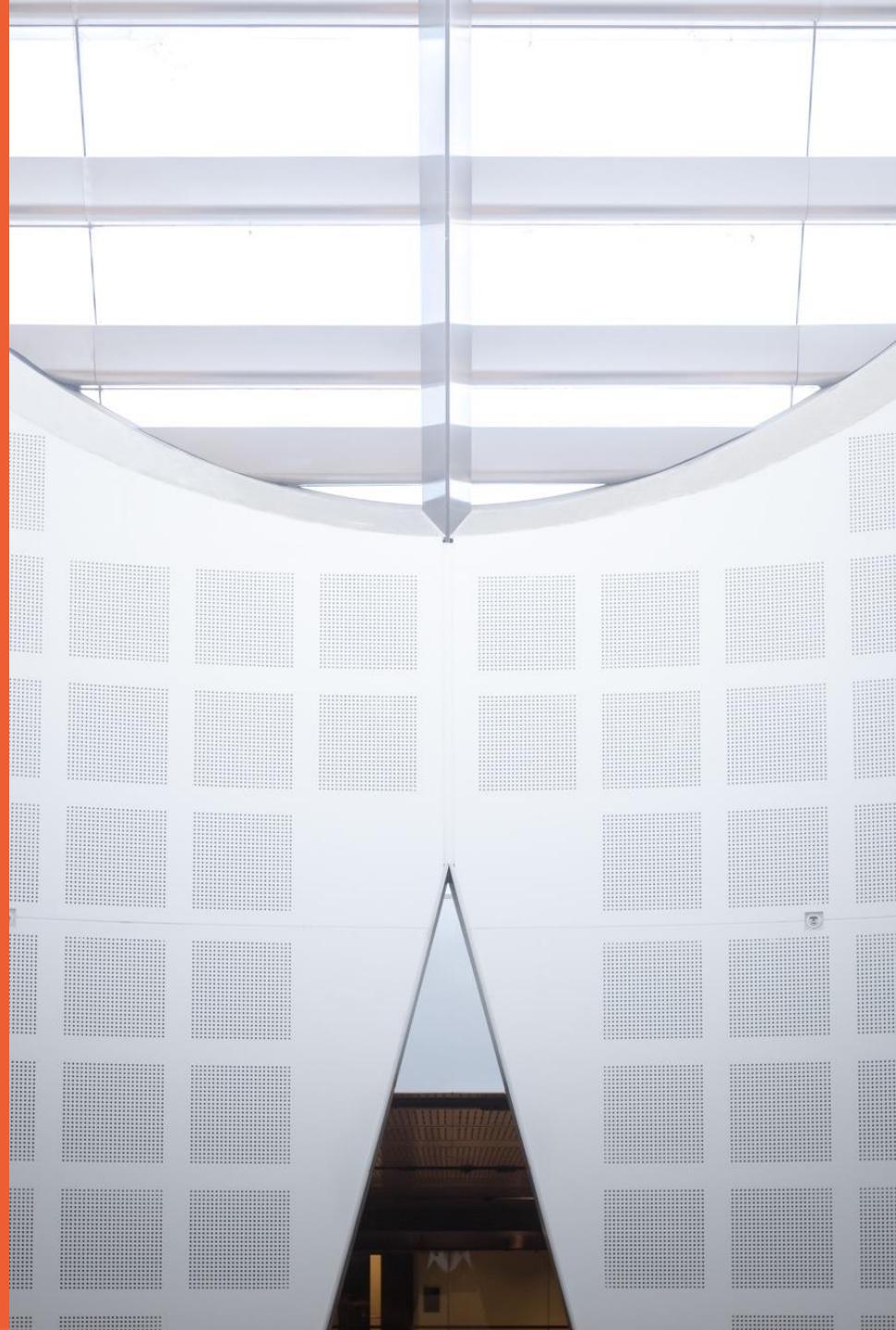


COMP9121: Design of Networks and Distributed Systems

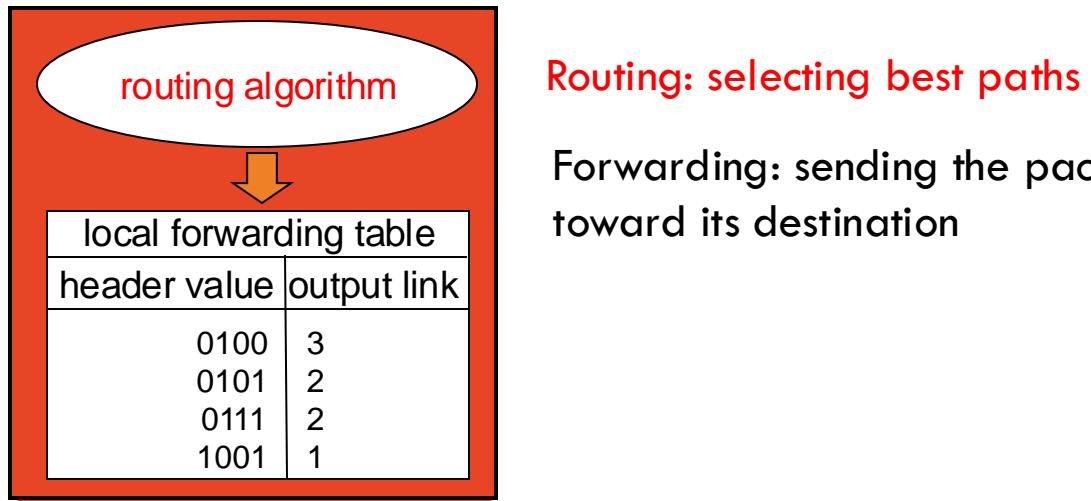
Week 5: Network Layer 2
Wei Bao
School of Computer Science



THE UNIVERSITY OF
SYDNEY

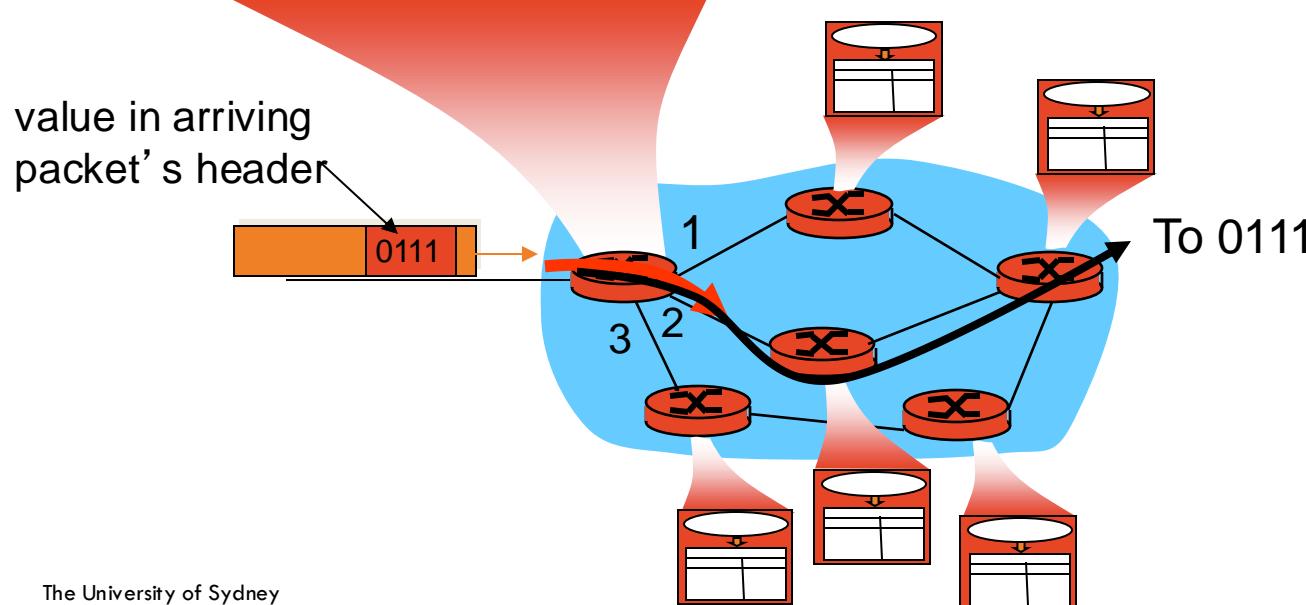


Routing and forwarding



Routing: selecting best paths in a network.

Forwarding: sending the packet to the next-hop toward its destination



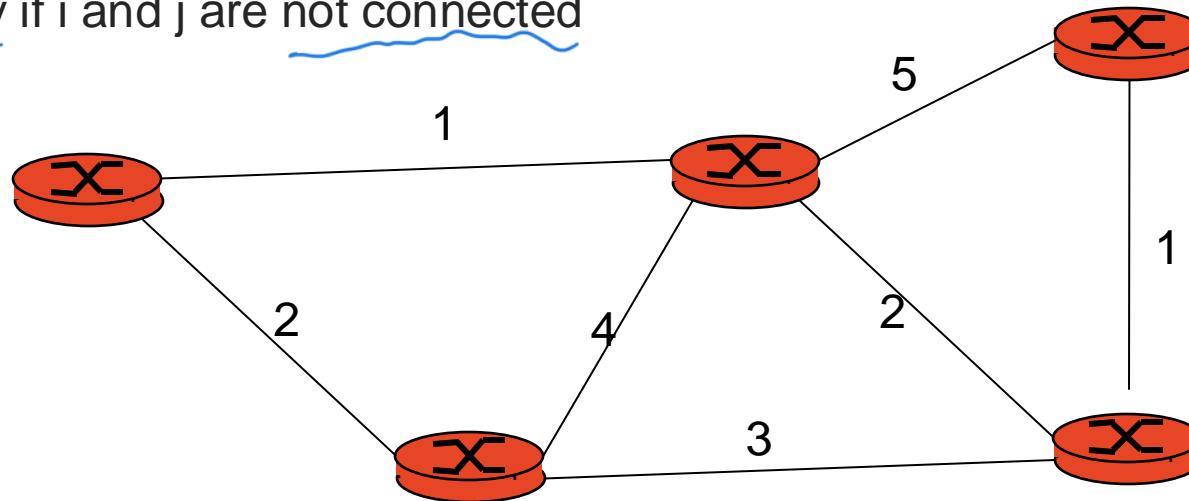
Link Cost

We need to find shortest path

i: i-th node (router).

C_{ij} : link cost between i and j

C_{ij} = infinity if i and j are not connected



Aim: Shortest path.

Classification

两种方法，分别适用两种 Case

Global or decentralized information?

Global:

- all routers have complete topology, link cost info
- “link state” algorithms ①

Decentralized:

- router only knows physically-connected neighbors, link costs to neighbors
- “distance vector” algorithms ②

Link State Algorithm: Dijkstra

Link-State Algorithm

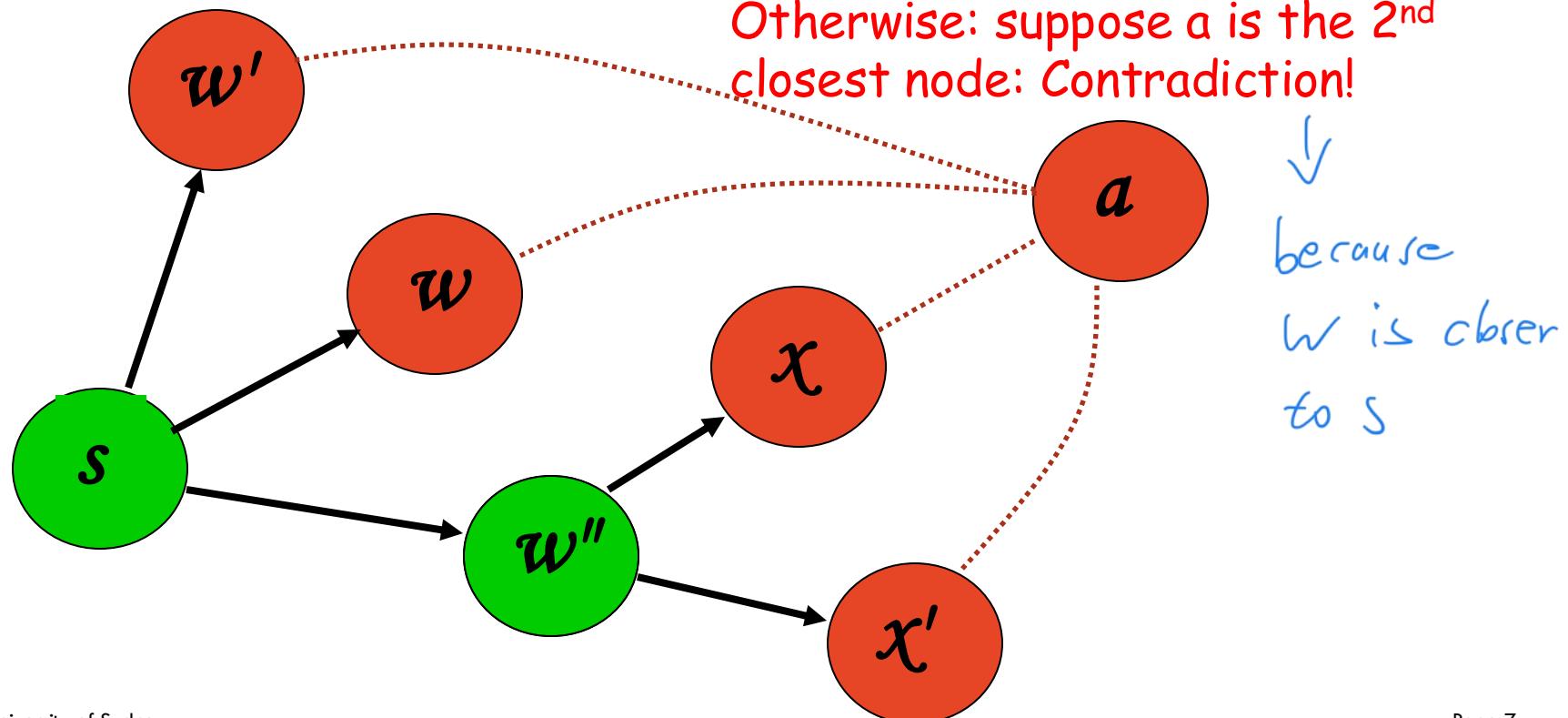
- Basic idea: two step procedure
 - 1 Each source node gets a **map** of all nodes and link costs of the entire network
 - 2 Find the **shortest path** on the map from the source node to all destination nodes
- Step 1: Broadcast of link-state information
 - Every node i in the network broadcasts to every other node in the network:
 - ID's of its neighbors: \mathcal{N}_i =set of neighbors of i
 - Distances to its neighbors: $\{C_{ij} \mid j \in \mathcal{N}_i\}$
 - **Flooding** is a popular method of broadcasting packets
- Step 2: Dijkstra

Dijkstra Algorithm: Finding shortest paths in order

Find shortest paths from source s to all other destinations

- ① Closest node to s is 1 hop away
- ② 2^{nd} closest node to s is 1 hop away from s or w''

Otherwise: suppose a is the 2^{nd} closest node: Contradiction!



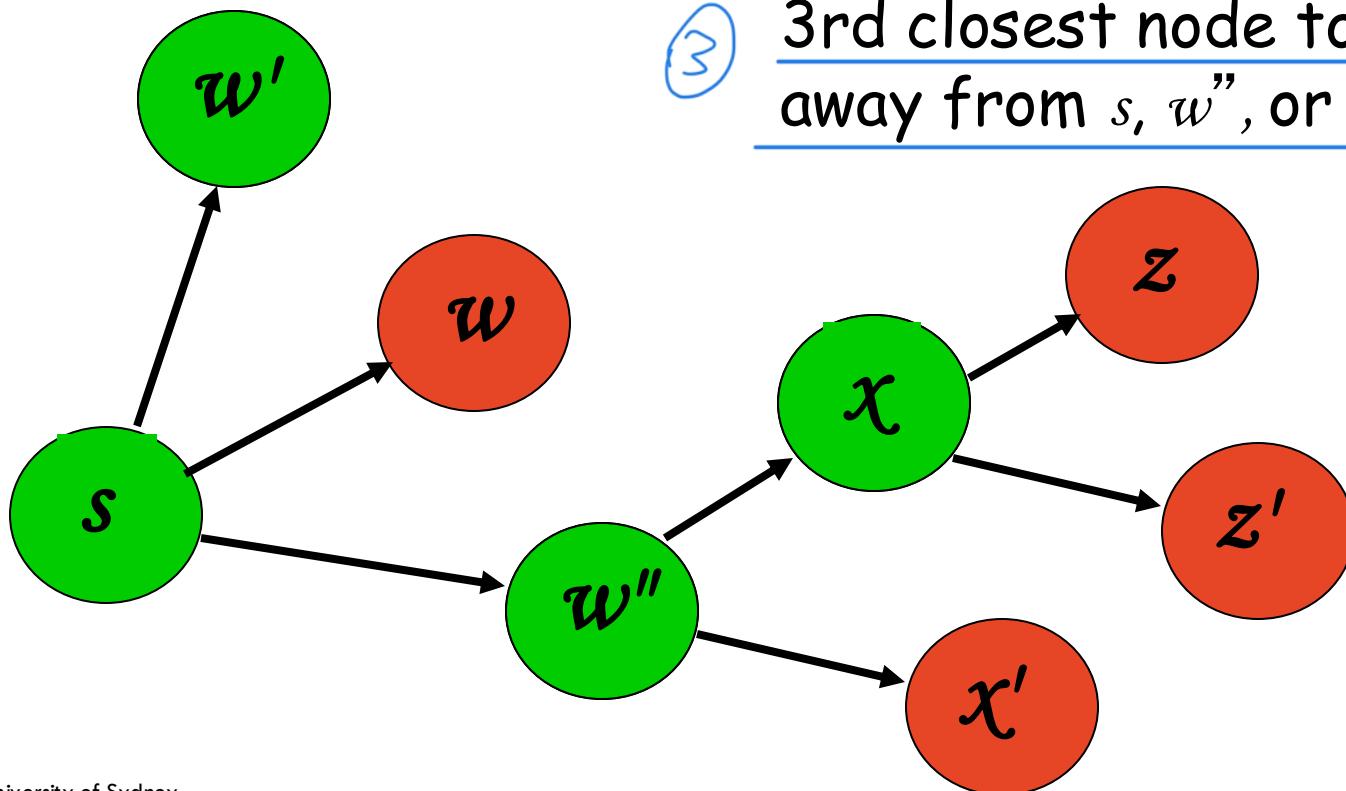
Dijkstra Algorithm: Finding shortest paths in order

Find shortest paths from source s to all other destinations

Closest node to s is 1 hop away

2nd closest node to s is 1 hop away from s or w'

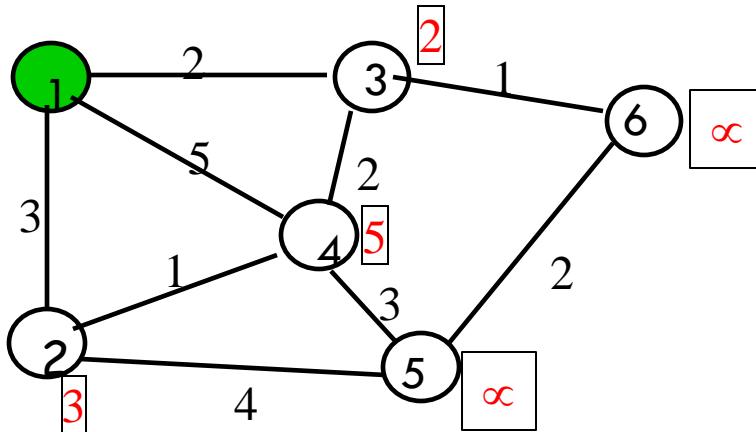
③ 3rd closest node to s is 1 hop away from s , w'' , or x



Dijkstra's algorithm

step 10

- C_{ij} : distance/cost from i to j
- D_j : current shortest distance from s to j (From Start to Current)
- N : set of nodes for which shortest path already found
- **Initialization:** (Start with source node s)
 - $N = \{s\}$, $D_s = 0$, “ s is distance zero from itself”
 - $D_i = C_{sj}$ for all $j \neq s$, distances of directly-connected neighbors



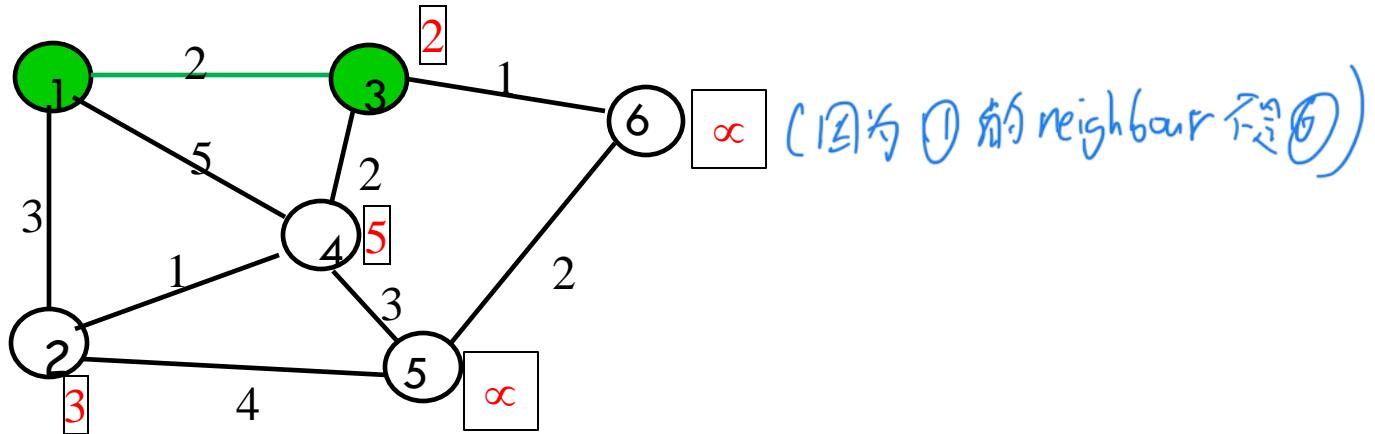
Dijkstra's algorithm

Step②

– Step A: (Find next closest node i)

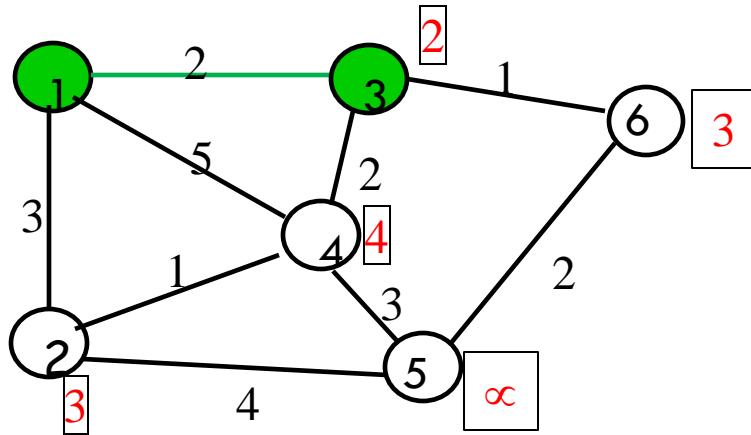
- Find $i \notin N$ such that
- $D_i = \min D_j$ for $j \notin N$
- Add i to N
- If N contains all the nodes, stop

因为已经找到 internet 中所有的 Node



Dijkstra's algorithm

- (node i was added to N last step)
 - Other nodes may find a better way via the newly added
 - Step B: (update minimum costs)
 - For each node $j \notin N$
 - $D_j = \min(D_i, D_i + C_{ij})$
- Minimum distance from s to j through node i in N*



Dijkstra's algorithm

- N : set of nodes for which shortest path already found
- **Initialization:** (*Start with source node s*)
 - $N = \{s\}$, $D_s = 0$, “ s is distance zero from itself”
 - $D_j = C_{sj}$ for all $j \neq s$, distances of directly-connected neighbors

Step A: (*Find next closest node i*)

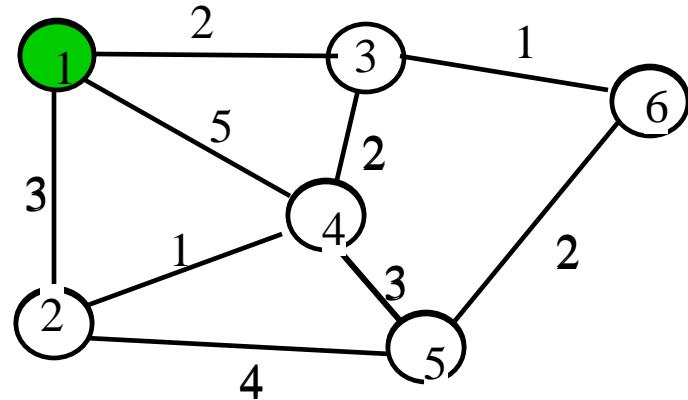
- Find $i \notin N$ such that
- $D_i = \min D_j$ for $j \notin N$
- Add i to N
- If N contains all the nodes, stop

if true

Step B: (*update minimum costs*)

- For each node $j \notin N$
- $D_j = \min (D_j, D_i + C_{ij})$
- Go to Step A

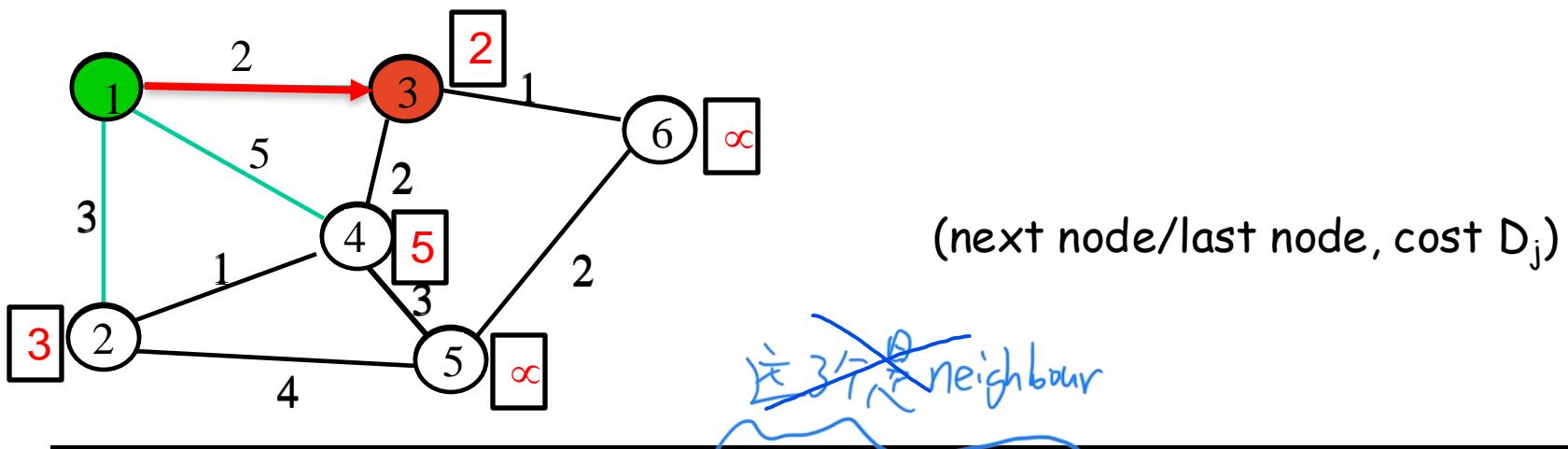
Dijkstra's algorithm: Example



Find the shortest path from 1 to all other nodes.

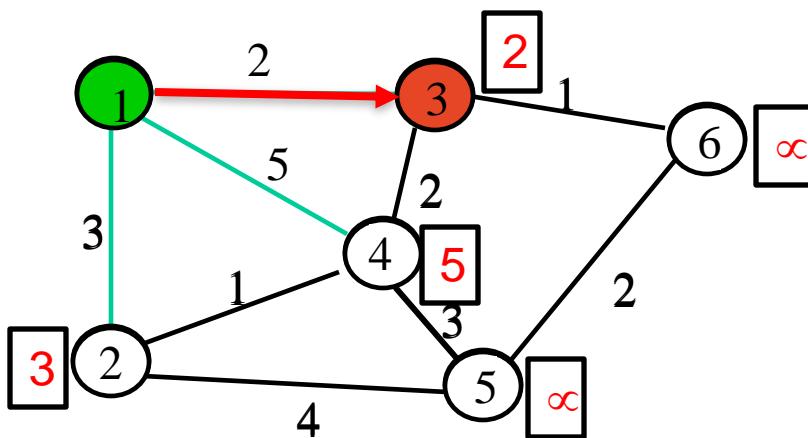
Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial						
1						
2						
3						
4						
5						

Dijkstra's algorithm: Example



Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1						
2						
3						
4						
5						

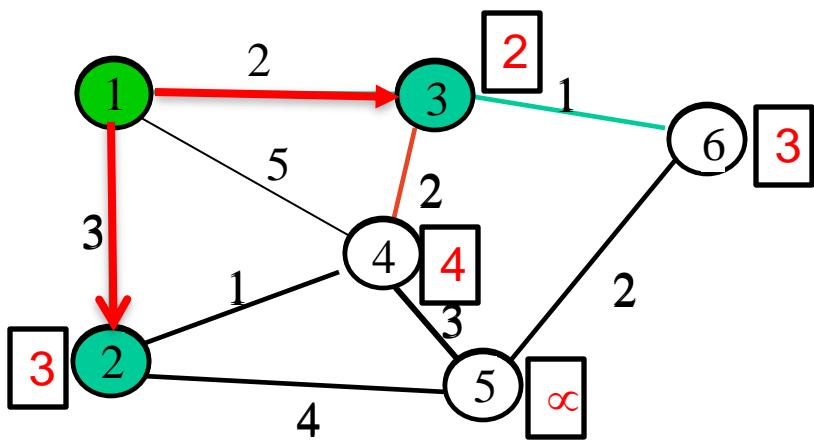
Dijkstra's algorithm: Example



Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1, 3}	: 找到从 start 到 N_1 和 N_3 所有 neighbour 的最短距离				
2						
3						
4						
5						

这 N_3 是因为 N_3 离 N_1 最近

Dijkstra's algorithm: Example

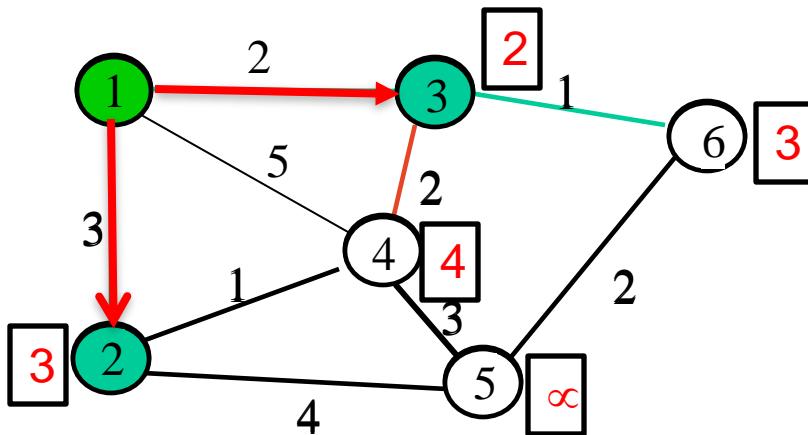


$N_1 \rightarrow N_4 \cancel{\rightarrow} 5$
 $N_1 \rightarrow N_3 \rightarrow N_4 \cancel{\rightarrow} 4$

Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2						
3						
4						
5						

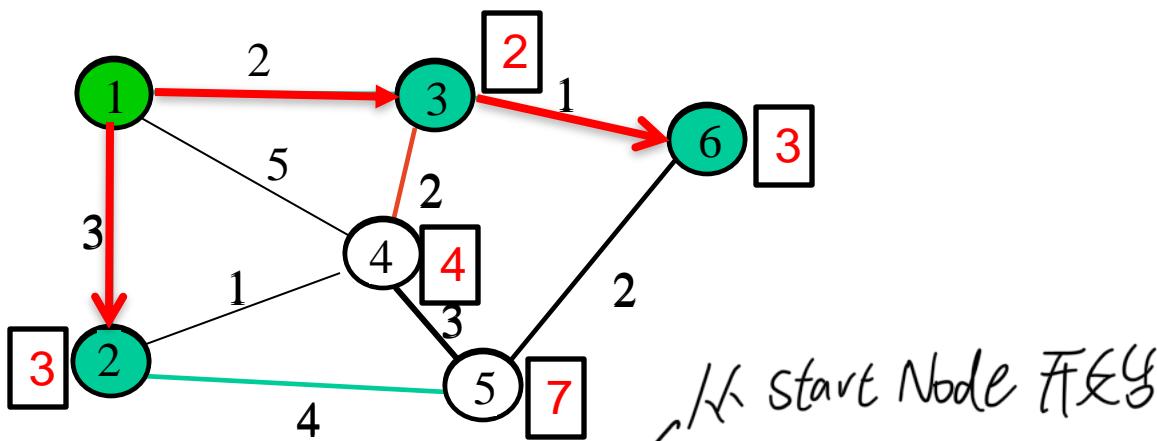
因为 N_2 和 N_6 的距离都是 3, 任何都可以

Dijkstra's algorithm: Example



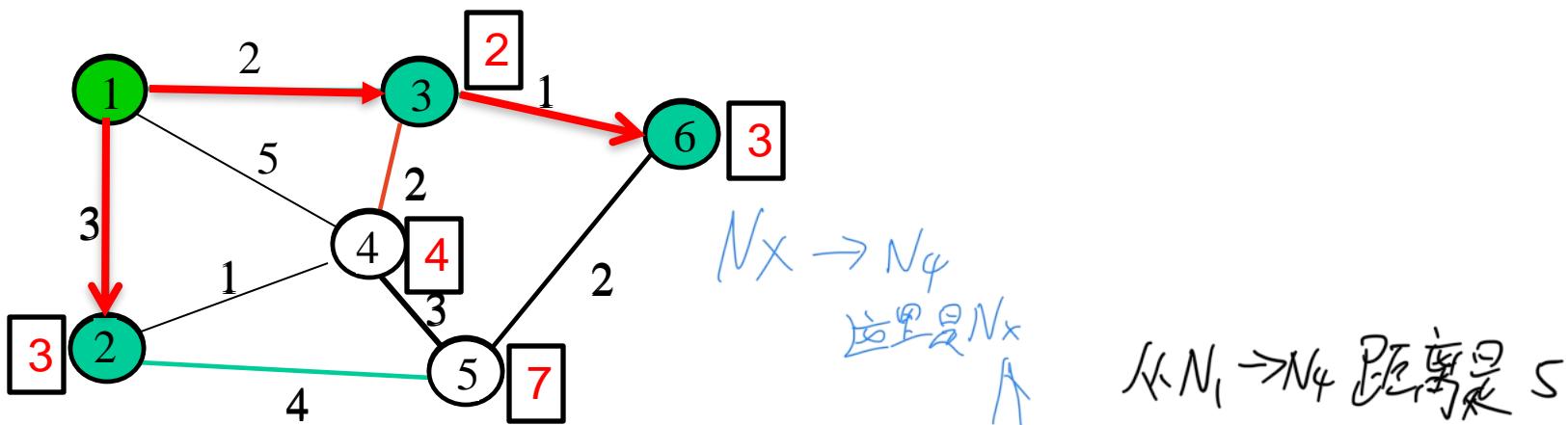
Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2	{1,2,3}					
3		→ 这 N_2 是因为除了 N_3 , N_2 是离 N_1 最近的				
4				→ rd	← least	
5						

Dijkstra's algorithm: Example



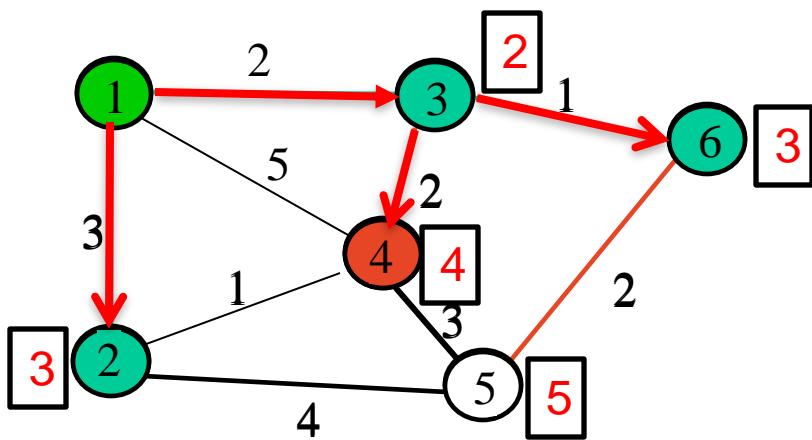
Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2	{1,2,3}			(3,4)	(2,7)	(3,3)
3						
4						
5						

Dijkstra's algorithm: Example



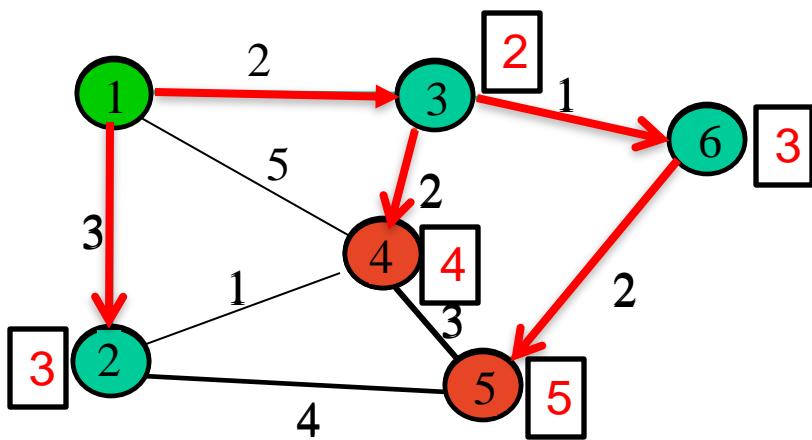
Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2	{1,2,3}			(3,4)	(2,7)	(3,3)
3	{1,2,3,6}					
4						
5						

Dijkstra's algorithm: Example



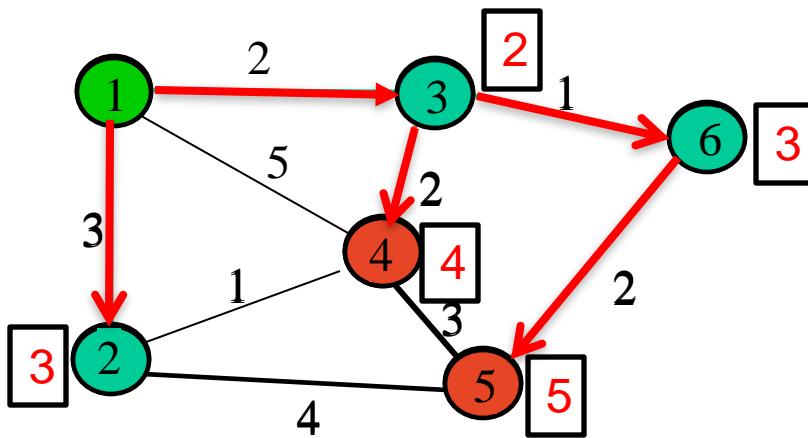
Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2	{1,2,3}			(3,4)	(2,7)	(3,3)
3	{1,2,3,6}			(3,4)	(6,5)	
4						
5						

Dijkstra's algorithm: Example



Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2	{1,2,3}			(3,4)	(2,7)	(3,3)
3	{1,2,3,6}			(3,4)	(6,5)	
4	{1,2,3,4,6}				(6,5)	

Dijkstra's algorithm: Example



Iteration	Tree	N_2	N_3	N_4	N_5	N_6
Initial	{1}	(1,3)	(1,2)	(1,5)	(-1,∞)	(-1,∞)
1	{1,3}	(1,3)		(3,4)	(-1,∞)	(3,3)
2	{1,2,3}			(3,4)	(2,7)	(3,3)
3	{1,2,3,6}			(3,4)	(6,5)	
4	{1,2,3,4,6}				(6,5)	
5	{1,2,3,4,5,6}	(1,3)	(1,2)	(3,4)	(6,5)	(3,3)

$N_6 \rightarrow N_3 \rightarrow N_1$

$N_5 \rightarrow N_6 \rightarrow N_3 \rightarrow N_1$

Reaction to Failure

- If a link fails,
 - Router sets link distance to infinity & floods the network with an update packet
 - All routers immediately update their link database & recalculate their shortest paths
 - Recovery very quick
- But watch out for old update messages
 - Add time stamp or sequence # to each update message
 - Check whether each received update message is new
 - If new, add it to database and broadcast
 - If older, send update message on arriving link

Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- each iteration: need to check all nodes, w, not in N
- operations: $O(n^2)$
- more efficient implementations possible: $O(n \log n)$
 - Using a heap to find the min.

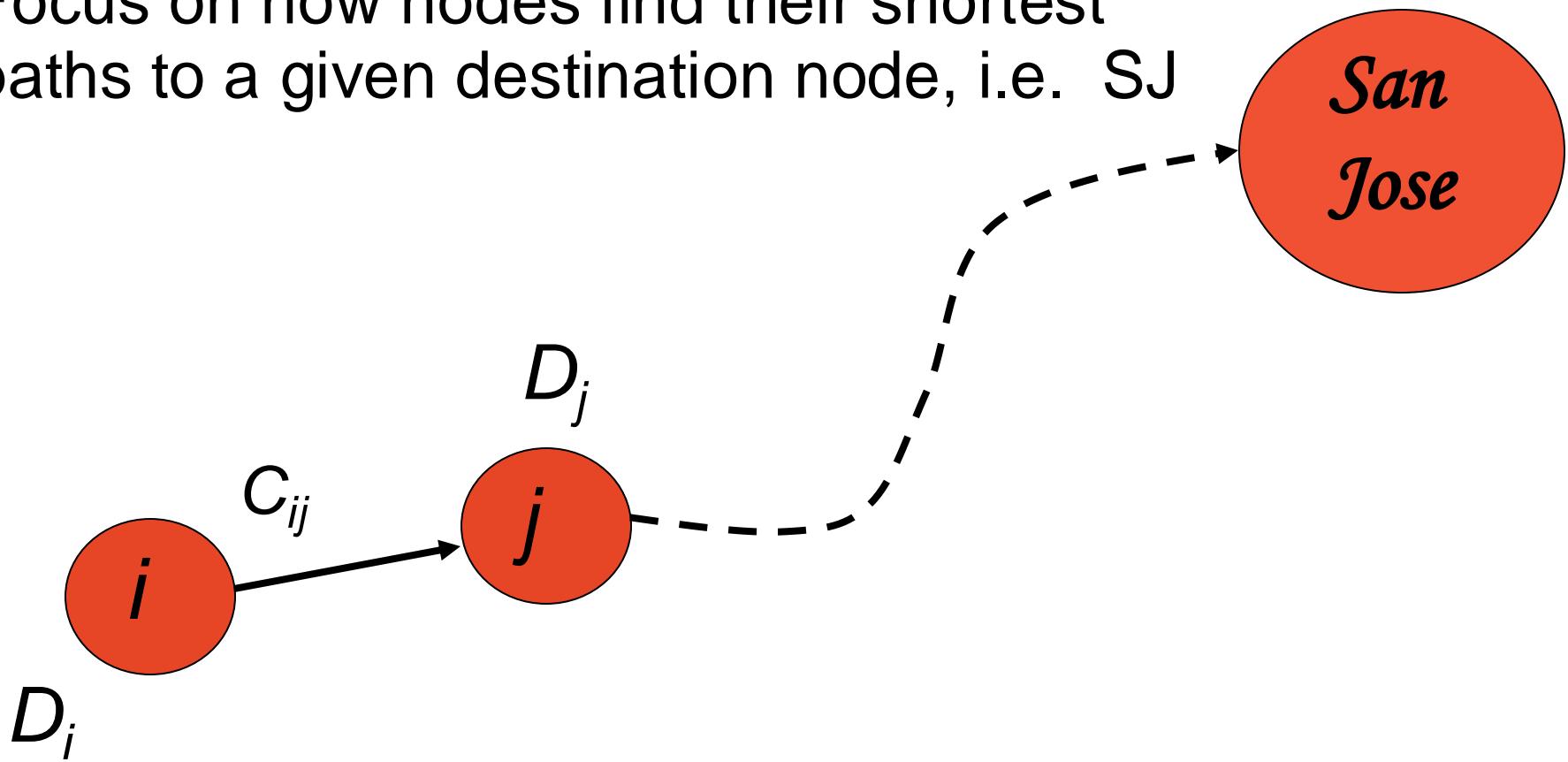
Important Concepts

- Link-state is a two-step procedure
 - Learn the map (by flooding)
 - Find the shortest path (use Dijkstra)
- LS is centralized!

Distance Vector Algorithm

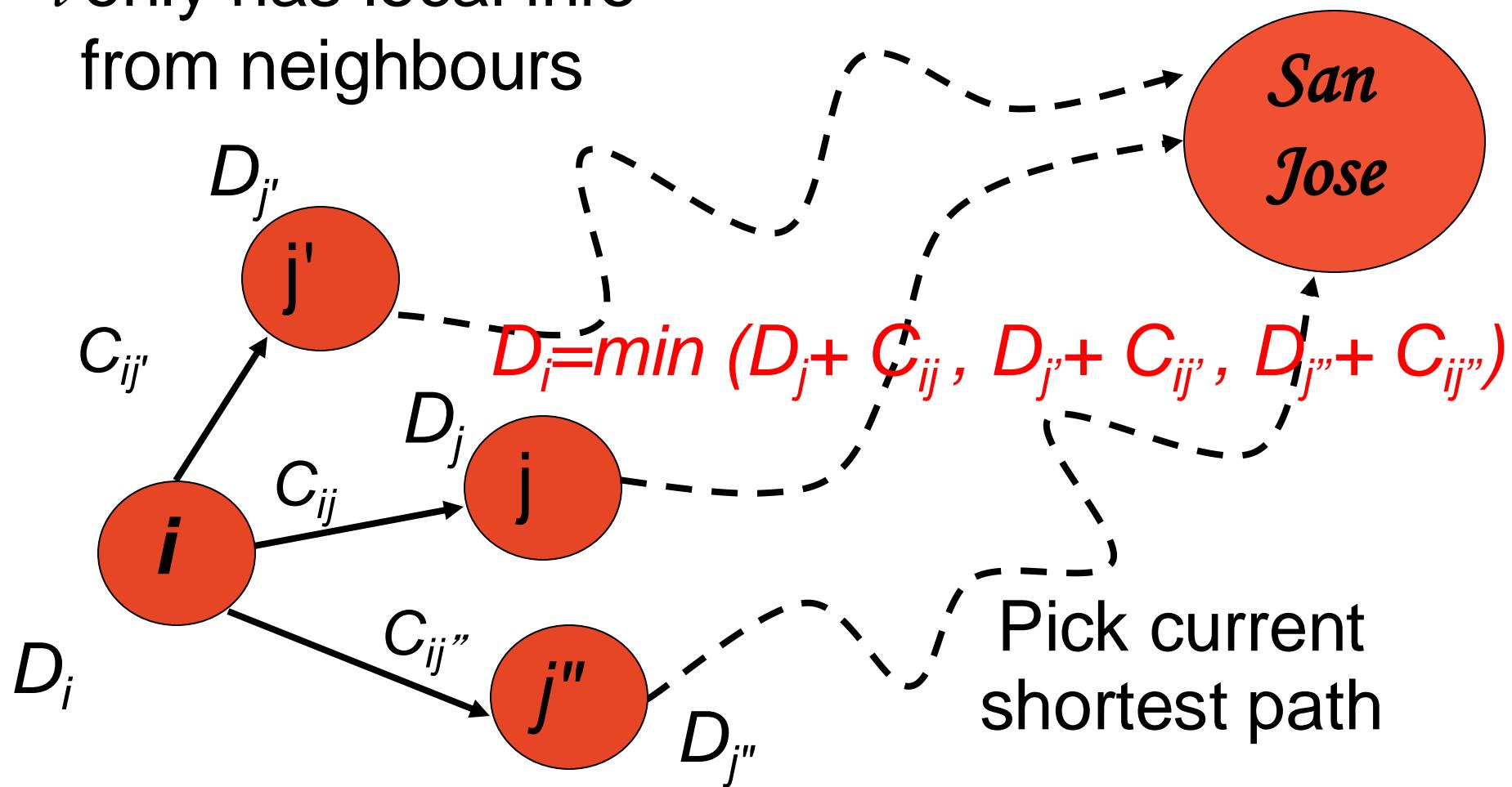
Shortest Path to SJ

Focus on how nodes find their shortest paths to a given destination node, i.e. SJ



But we don't know the shortest paths

i only has local info
from neighbours



Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

$D_x(y) := \text{cost of shortest path from } x \text{ to } y$

Then

$$D_x(y) = \min_v \{ C(x,v) + D_v(y) \}$$

where min is taken over all neighbors v of x

we only know shortest distance between neighbour and current neighbour know shortest distance between neighbour and destination Node

(for recursive
step)

y=6

Destination node 6

$$D_1 = \min \{ 3+D_2, 2+D_3, 5+D_4 \} \quad x=1 \quad y=6, \quad n=\{2, 3, 4\}$$

$$D_2 = \min \{ 3+D_1, 1+D_4, 4+D_5 \} \quad x=2 \quad y=6 \quad n=\{1, 4, 5\}$$

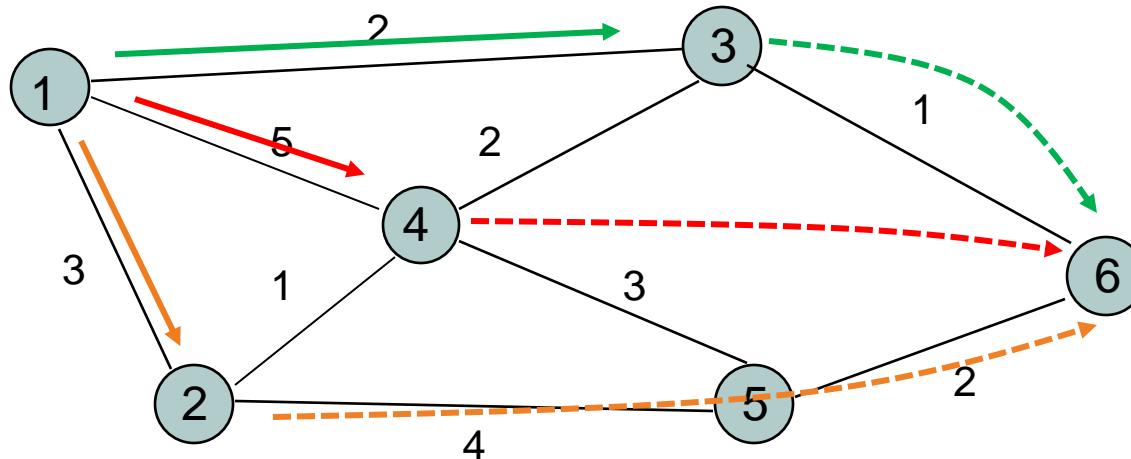
$$D_3 = \min \{ 2+D_1, 2+D_4, 1 \}$$

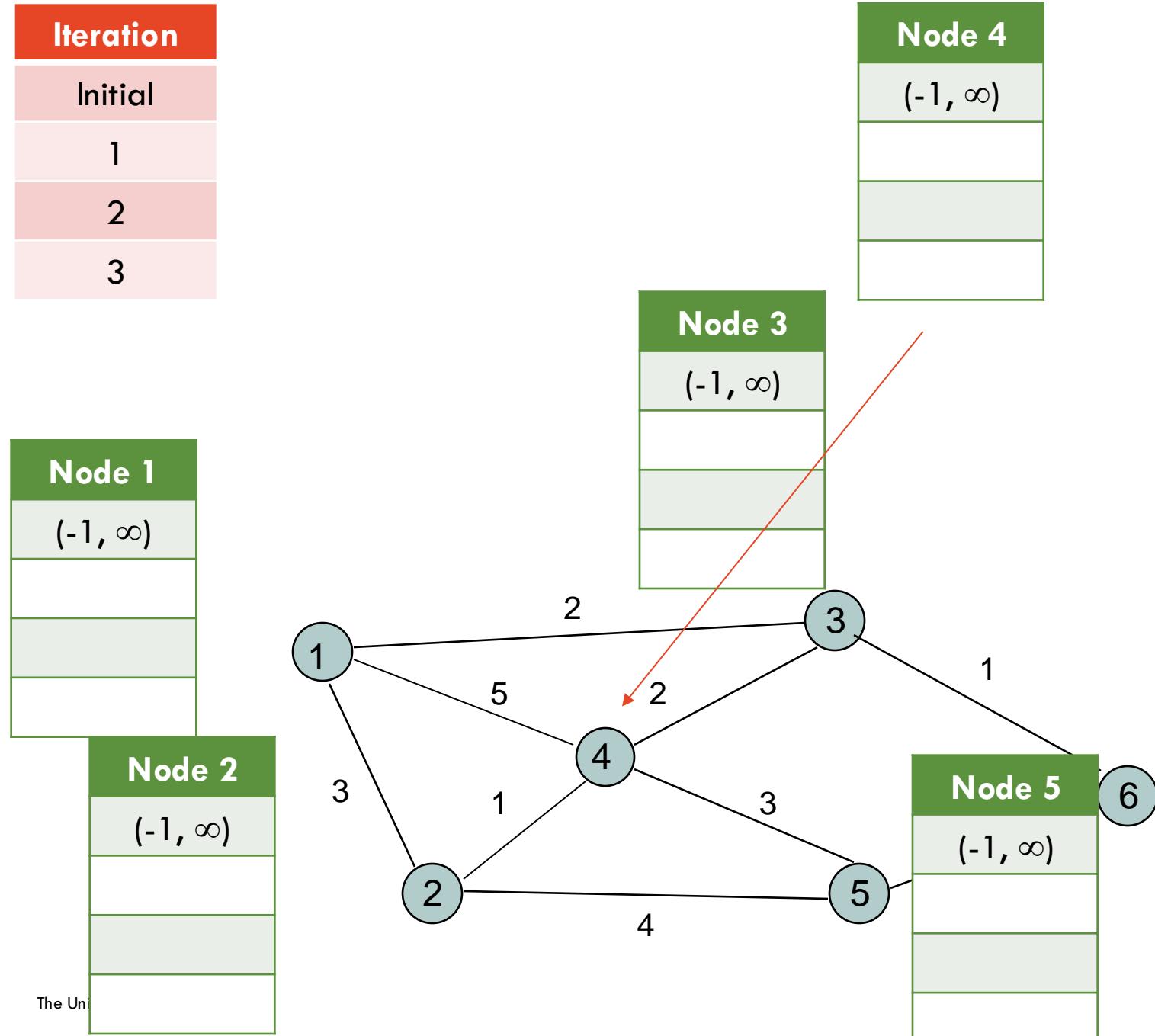
$$D_4 = \min \{ 5+D_1, 1+D_2, 2+D_3, 3+D_5 \}$$

$$D_5 = \min \{ 4+D_2, 3+D_4, 2 \}$$

How to solve: Use an iterative procedure!

Use this Bellman-ford equation every round, until convergence.





*San
Jose*

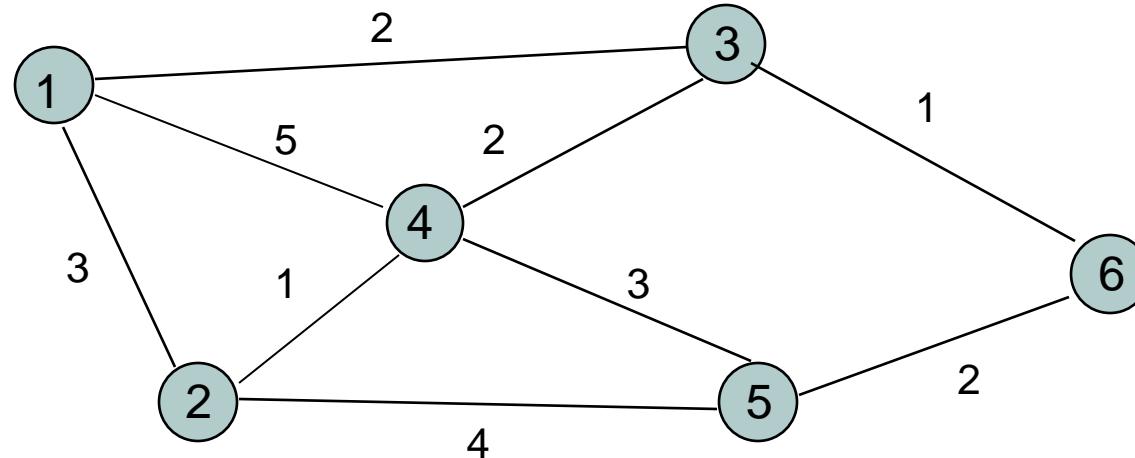
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)				
1					
2					
3					

{

{

Table entry
@ node 1
for dest SJ

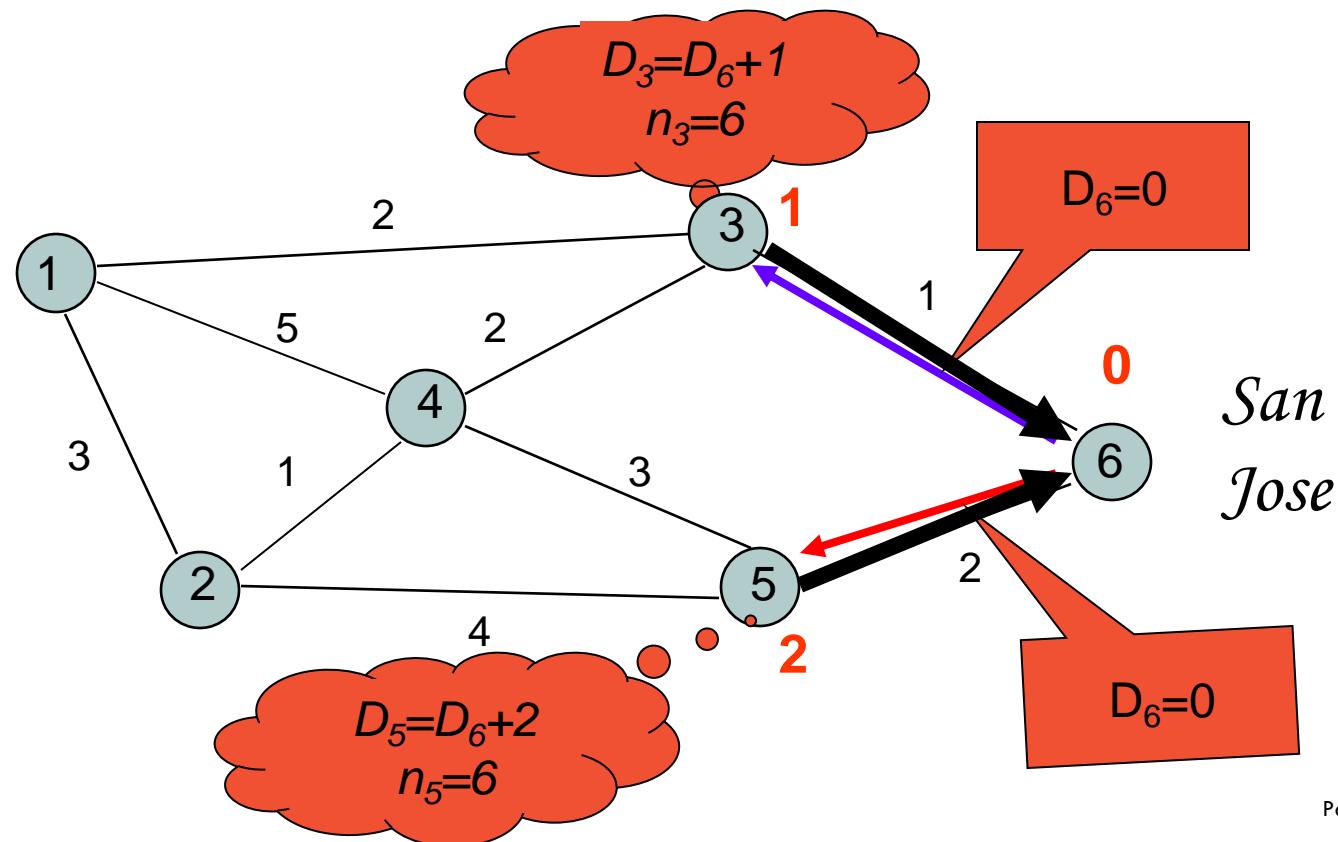
Table entry
@ node 3
for dest SJ



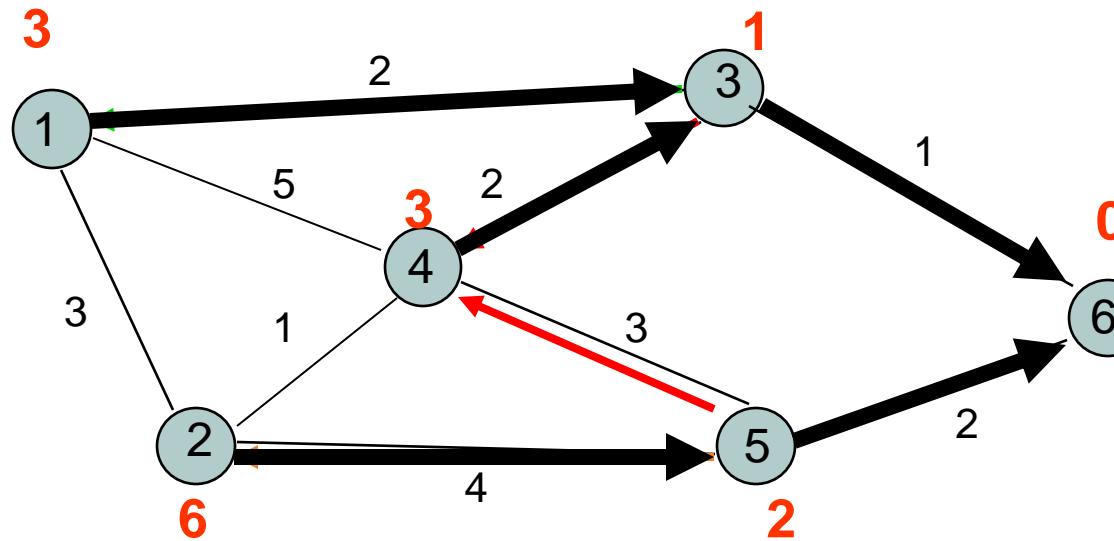
San
Jose

→ 所有次數都變小

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)				
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6, 2)
2					
3					



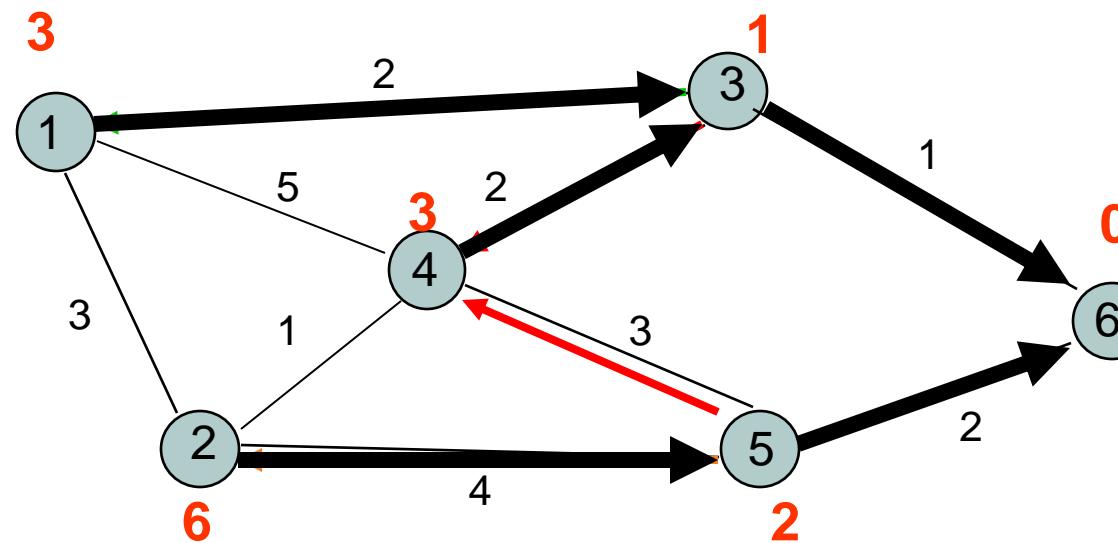
Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)				
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6, 2)
2	(3, 3)	(5, 6)	(6, 1)	(3, 3)	(6, 2)
3					



San
Jose

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)				
1	(-1, ∞)	(-1, ∞)	(6, 1)	(-1, ∞)	(6, 2)
2	(3, 3)	(5, 6)	(6, 1)	(3, 3)	(6, 2)
3					

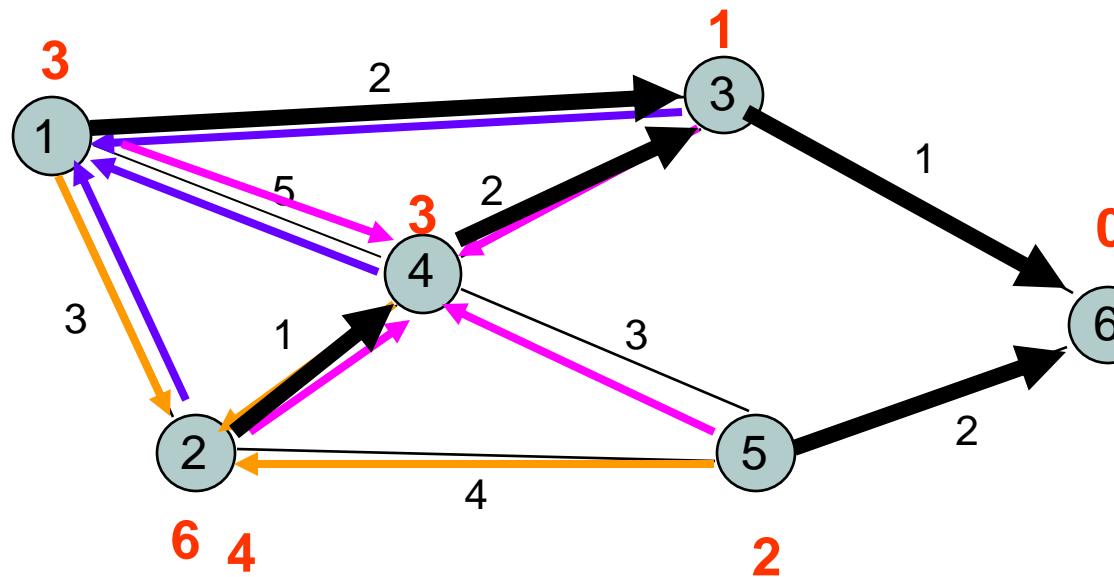
Iter 2 只能用 Iter 1 中的信息



San
Jose

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(-1, ∞)				
1	(-1, ∞)	(-1, ∞)	(6,1)	(-1, ∞)	(6,2)
2	(3,3)	(5,6)	(6, 1)	(3,3)	(6,2)
3	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)

\hookrightarrow if $b = 4$
in node 4



San
Jose

Distance vector algorithm

Basic idea:

- From time-to-time (ex. 30 sec), each node sends its own distance vector estimate to neighbours
- Distance vector: (Current shortest distance to 1, Current shortest distance to 2, ... Current shortest distance to N)
If we have multiple destination
- When a node x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v \{C(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

- Under minor, natural conditions, the estimate $D_x(y)$ converges to the actual least cost $d_x(y)$

Example

Consider the following network in which distance vector routing is used. Each node in this network sends its routing table using a vector of size 6 where each entity of the vector represents the cost to the corresponding node, i.e. (Cost-to-Node1, Cost-to-Node2, Cost-to-Node3, Cost-to-Node4, Cost-to-Node5, Cost-to-Node6).

The following cost vectors have just arrived at router 1 from its neighbours:

from 2: (2, 0, 2, 7, 4, 1);
from 3: (3, 2, 0, 3, 1, 4);
from 5: (1, 3, 1, 4, 0, 3).

The link costs between node 1 and nodes 2, 3, and 5 are 2, 3, and 1, respectively. What is the routing table at node 1?

Cost Next Node

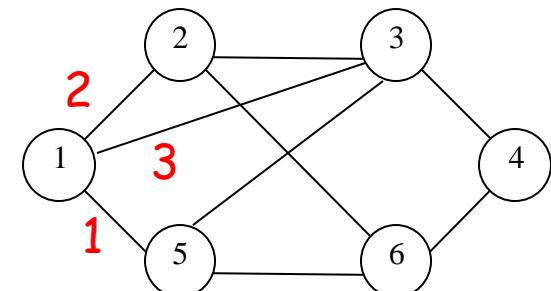
For Node 2: $\min \{2+0, 3+2, 1+3\} = 2$ 2

For Node 3: $\min \{2+2, 3+0, 1+1\} = 2$ 5

For Node 4: $\min \{2+7, 3+3, 1+4\} = 5$ 5

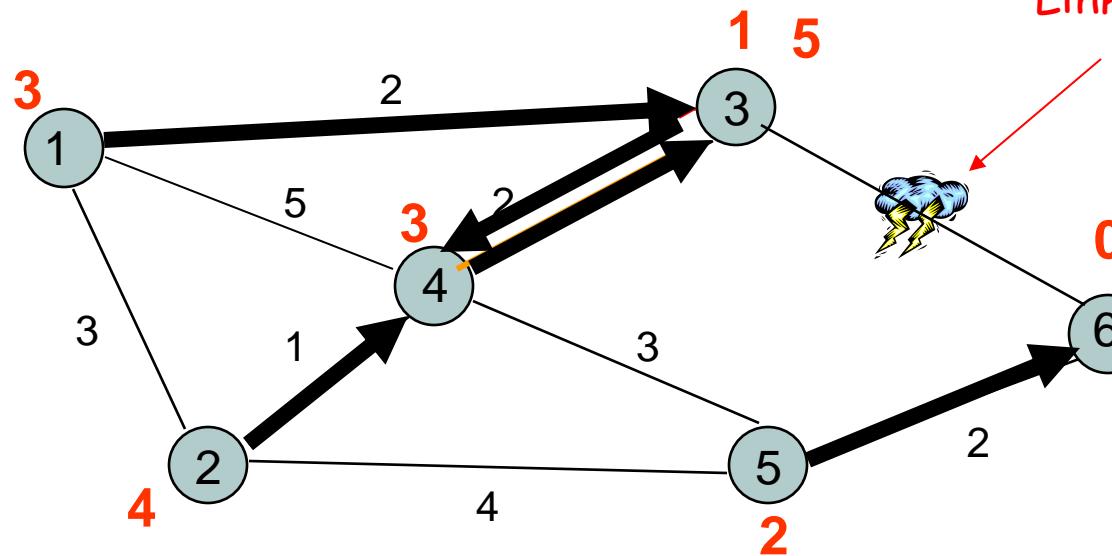
For Node 5: $\min \{2+4, 3+1, 1+0\} = 1$ 5

For Node 6: $\min \{2+1, 3+4, 1+3\} = 3$ 2



Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6,1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4,5)	(3,3)	(6,2)
2					
3					

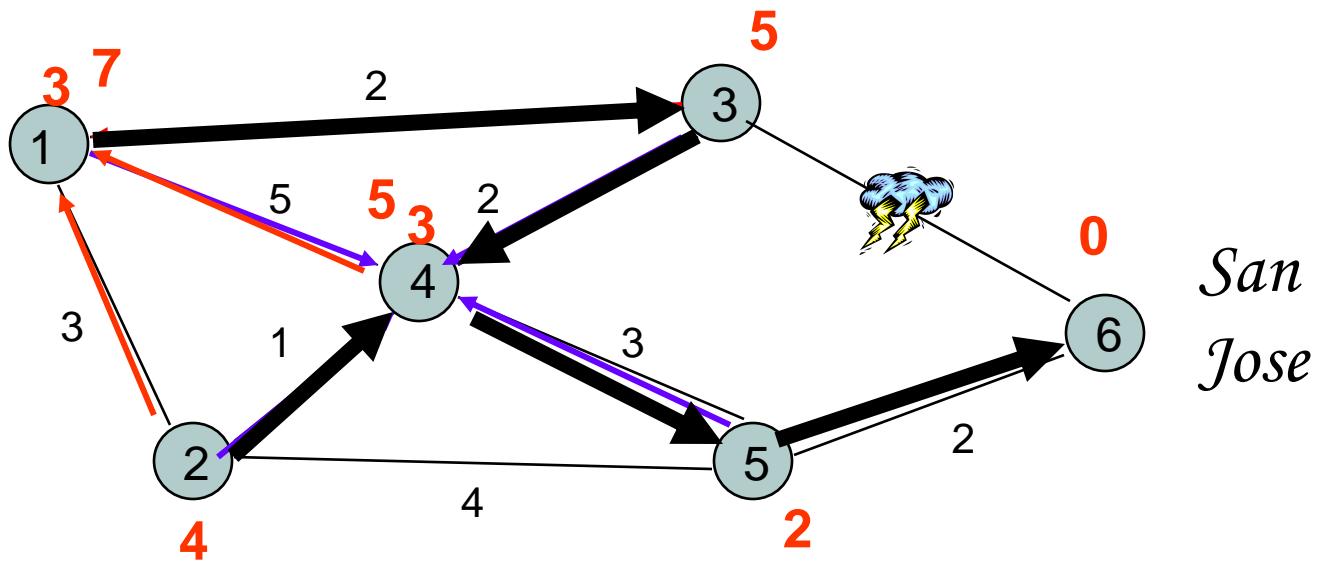
What happen if
Link cost changes



San
Jose

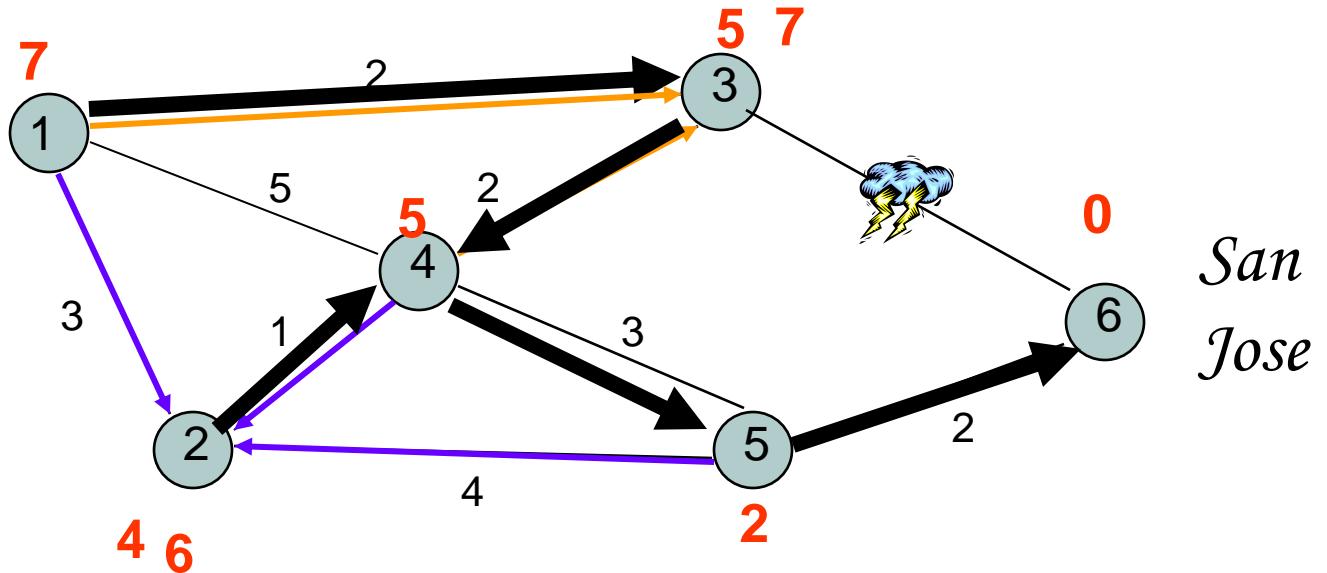
Network disconnected; Loop created between nodes 3 and 4

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6, 1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4, 5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4, 5)	(5,5)	(6,2)
3					



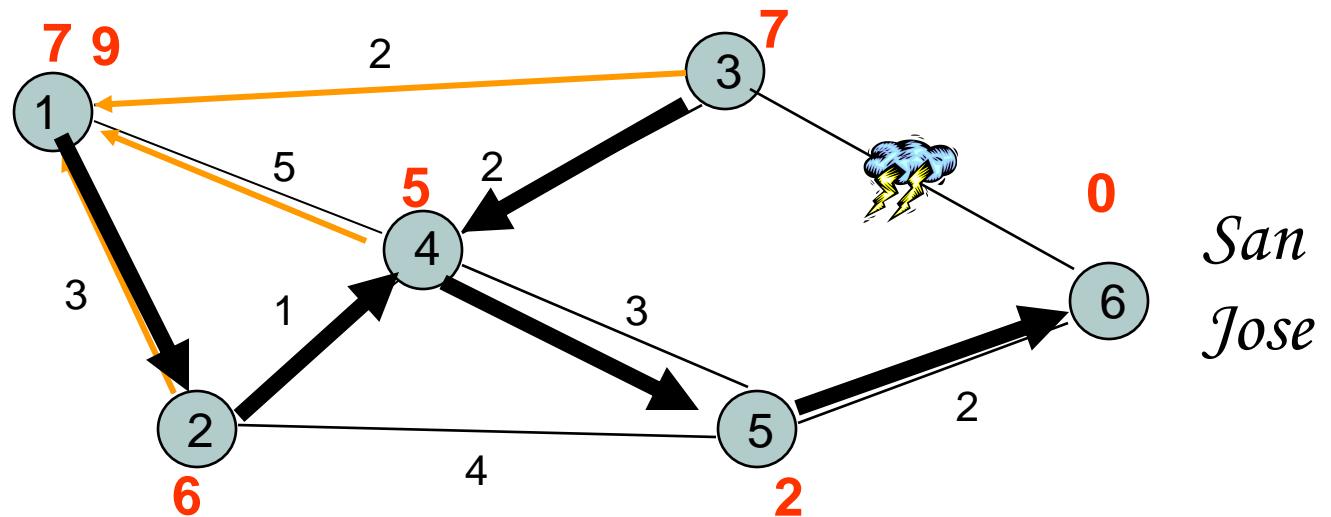
Node 4 could have chosen 2 as next node because of tie

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
Initial	(3,3)	(4,4)	(6,1)	(3,3)	(6,2)
1	(3,3)	(4,4)	(4,5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4,5)	(5,5)	(6,2)
3	(3,7)	(4,6)	(4,7)	(5,5)	(6,2)



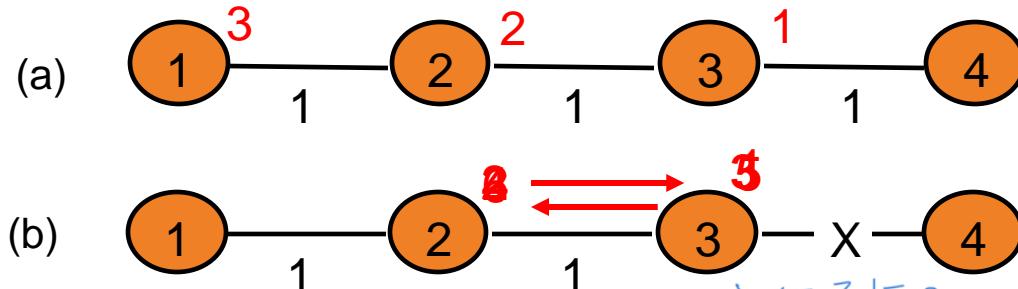
Node 2 could have chosen 5 as next node because of tie

Iteration	Node 1	Node 2	Node 3	Node 4	Node 5
1	(3,3)	(4,4)	(4,5)	(3,3)	(6,2)
2	(3,7)	(4,4)	(4,5)	(2,5)	(6,2)
3	(3,7)	(4,6)	(4,7)	(5,5)	(6,2)
4	(2,9)	(4,6)	(4,7)	(5,5)	(6,2)



Node 1 could have chosen 3 as next node because of tie

Counting to Infinity Problem



Nodes believe best path is through each other

(Destination is node 4)

Update	Node 1	Node 2	Node 3
Before break	(2,3)	(3,2)	(4,1)
After break	(2,3)	(3,2)	(2,3)
1	(2,3)	(3,4)	(2,3)
2	(2,5)	(3,4)	(2,5)
3	(2,5)	(3,6)	(2,5)
4	(2,7)	(3,6)	(2,7)
5	(2,7)	(3,8)	(2,7)
...

Problem: Bad News Travels Slowly

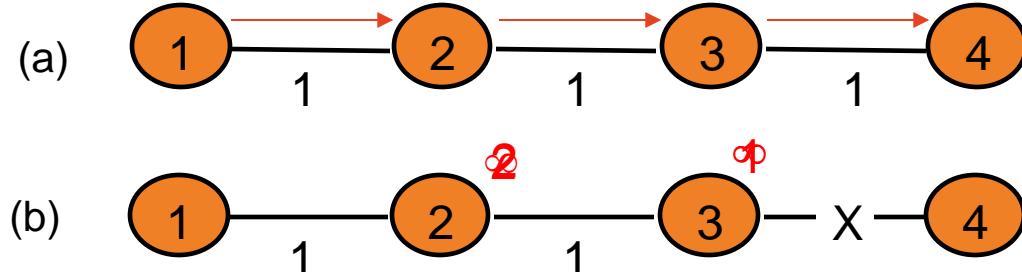
类似于单向箭头

通过发 ∞

Remedies

- **Split Horizon** *也就是说 A learn B, 那么 A 就不会再 teach B*
 - Do not report route to a destination to the neighbor from which route was learned
- **Poisoned Reverse**
 - Report route to a destination to the neighbor from which route was learned, but with infinite distance

Split Horizon with Poison Reverse



Nodes believe best path is through each other

Don't learn from 1

Don't learn from 2

Update	Node 1	Node 2	Node 3	
Before break	(2, 3)	(3, 2)	(4, 1)	
After break	(2, 3)	(3, 2)	(-1, ∞)	Node 2 advertises its route to 4 to node 3 as having distance infinity; node 3 finds there is no route to 4
1	(2, 3)	(-1, ∞)	(-1, ∞)	Node 1 advertises its route to 4 to node 2 as having distance infinity; node 2 finds there is no route to 4
2	(-1, ∞)	(-1, ∞)	(-1, ∞)	Node 1 finds there is no route to 4

Important Concepts

- *Link State is centralized*
 - Dijkstra
- *Distance Vector is decentralized*
 - Split Horizon/Poison Reverse