CPE348: Introduction to Computer Networks

Lecture #12: Chapter 3.5

Jianqing Liu
Assistant Professor of Electrical and Computer Engineering, University of Alabama in Huntsville

jianqing.liu@uah.edu
http://jianqingliu.net

Routing

Forwarding, Switching or Routing

- Forwarding:
  - one device sending a datagram to the next one in the path to the destination
- Switching:
  - moving a datagram from one interface to another within a device
- Routing:
  - a specific process in a layer-3 device to decide what to do with a layer-3 packet

Routing

- Forwarding table vs Routing table
  - Forwarding table
    - It contains the mapping from a network number to an outgoing interface and some MAC information, such as Ethernet Address of the next hop
  - Routing table
    - It contains mapping from network numbers to next hops

Routing

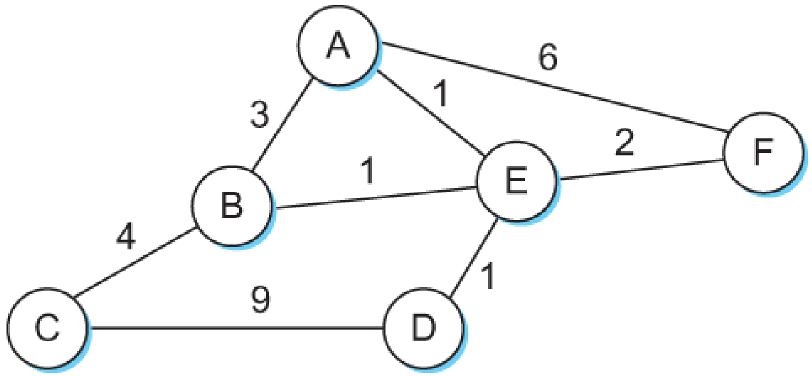
(a)	
Prefix/Length	Next Hop
18/8	171.69.245.10

(b)		
Prefix/Length	Interface	MAC Address
18/8	if0	8:0:2b:e4:b:1:2

Example rows from (a) routing and (b) forwarding tables

Routing

- Network as a Graph



- The basic problem of routing is to find the lowest-cost path between any two nodes

- The term – “cost” – has many different meanings:
  - Latency/delay
  - Number of hops
  - Error probability
  - …

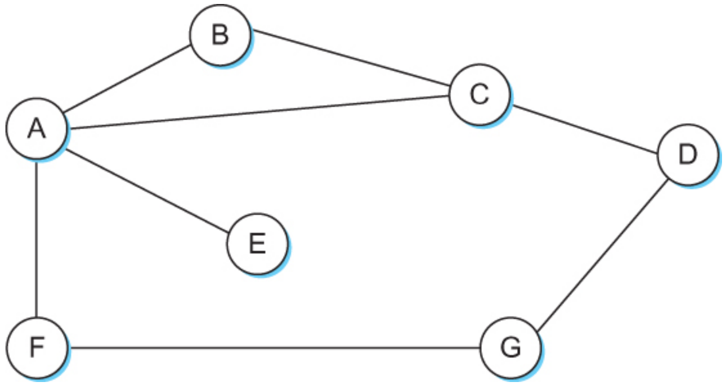
Routing

Is there any algorithm that can calculate the shortest path?

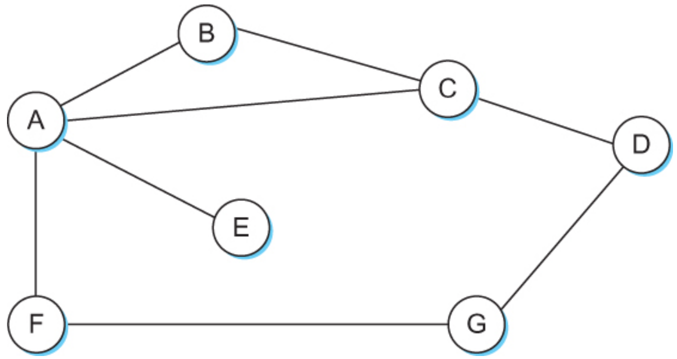
- Centralized approach:
  - A central controller calculates all shortest paths and load them into these routers. Pros & Cons?
- Distributed approach:
  - Two main classes of protocols
    - Distance Vector
    - Link State

Distance Vector Routing Algorithm

- Each node constructs a vector containing the "costs" to all other nodes
- It then distributes that vector to its immediate neighbors
- Open question: how could a node know the cost?



Distance Vector - Example

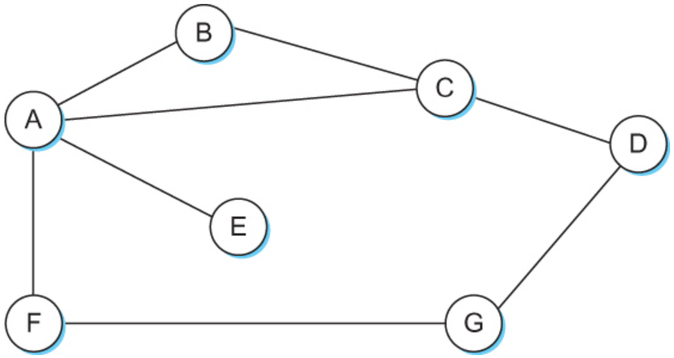


Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Initial distances stored at each node (global view)

Use hop count as the cost

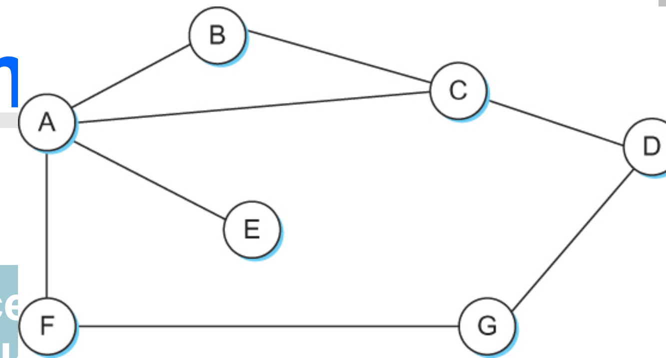
Distance Vector - Example



Destination	Cost	NextHop
B	1	B
C	1	C
D	∞	—
E	1	E
F	1	F
G	∞	—

Initial routing table at node A

Distance Vector - Exam

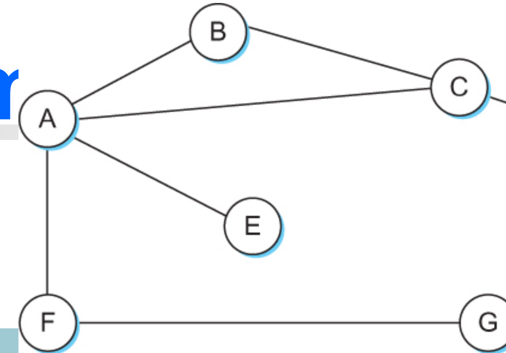


Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	∞	1	1	∞
B	1	0	1	∞	∞	∞	∞
C	1	1	0	1	∞	∞	∞
D	∞	∞	1	0	∞	∞	1
E	1	∞	∞	∞	0	∞	∞
F	1	∞	∞	∞	∞	0	1
G	∞	∞	∞	1	∞	1	0

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	∞
C	1	1	0	1	2	2	2
D	2	2	1	0	∞	2	1
E	1	2	2	∞	0	2	∞
F	1	2	2	2	2	0	1
G	2	∞	2	1	∞	1	0

Distances stored at each node after 1 update (global view)

Distance Vector - Exam

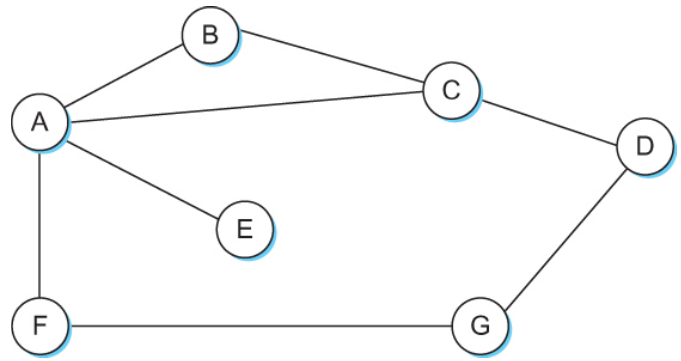


Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	∞
C	1	1	0	1	2	2	2
D	2	2	1	0	∞	2	1
E	1	2	2	∞	0	2	∞
F	1	2	2	2	2	0	1
G	2	∞	2	1	∞	1	0

Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Distances stored at each node after 2 updates (global view)

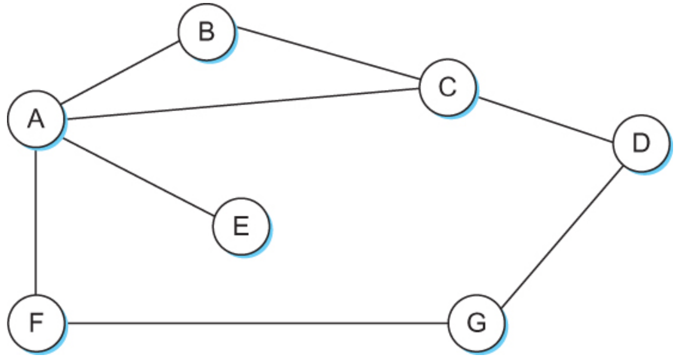
Distance Vector - Example



Information Stored at Node	Distance to Reach Node						
	A	B	C	D	E	F	G
A	0	1	1	2	1	1	2
B	1	0	1	2	2	2	3
C	1	1	0	1	2	2	2
D	2	2	1	0	3	2	1
E	1	2	2	3	0	2	3
F	1	2	2	2	2	0	1
G	2	3	2	1	3	1	0

Final distances stored at each node (global view)

Distance Vector - Example



Destination	Cost	NextHop
B	1	B
C	1	C
D	2	C
E	1	E
F	1	F
G	2	F

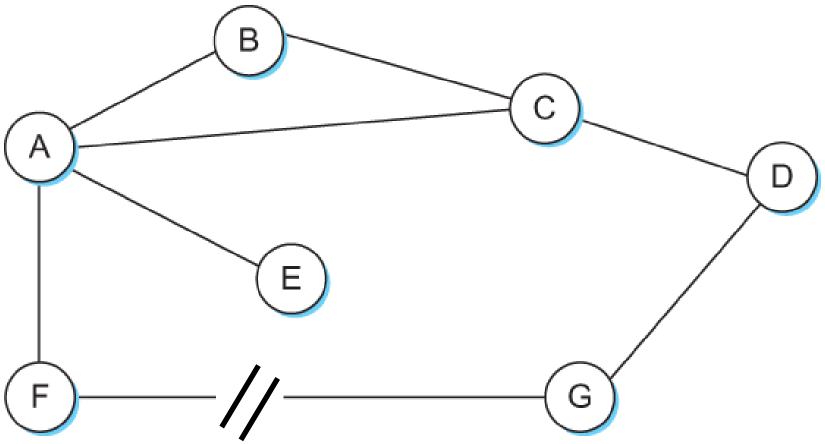
Final routing table at node A

Distance Vector Routing Algorithm

- The distance vector routing algorithm is also called as Bellman-Ford algorithm
- Every T seconds each router sends its table to its neighbor, which then updates its table based on the new information

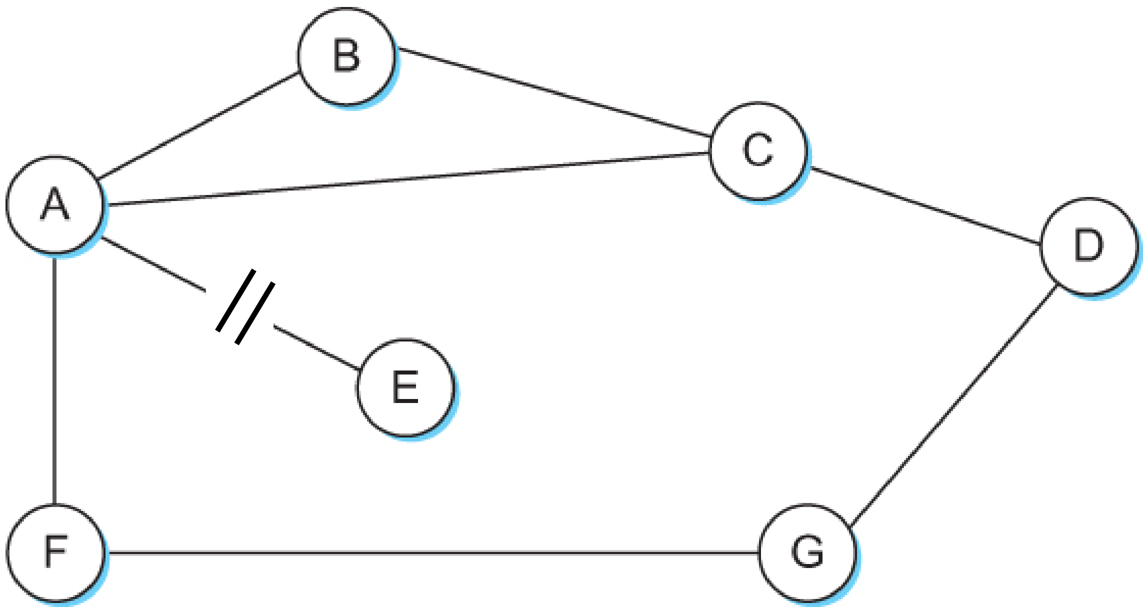
DVR Algorithm – issue

- When a node detects a link failure
  - F detects that link to G has failed
  - F sets distance to G to infinity and sends update to A
  - A sets distance to G to infinity since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A



DVR Algorithm – issue

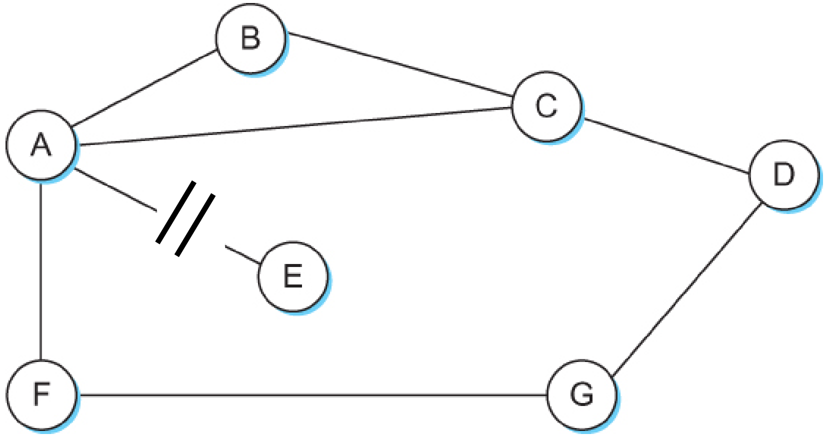
- Another example:
  - Suppose the link from A to E goes down
  - In the next round of updates, A advertises a distance of infinity to E, but B and C advertise a distance of 2 to E



DVR Algorithm – issue

- Depending on the exact timing of events, the following might happen
  - Node B, upon hearing that E can be reached in 2 hops from C, concludes that it can reach E in 3 hops and advertises this to A
  - Node A concludes that it can reach E in 4 hops and advertises this to C
  - Node C concludes that it can reach E in 5 hops; and so on.

Count-to-infinity problem!

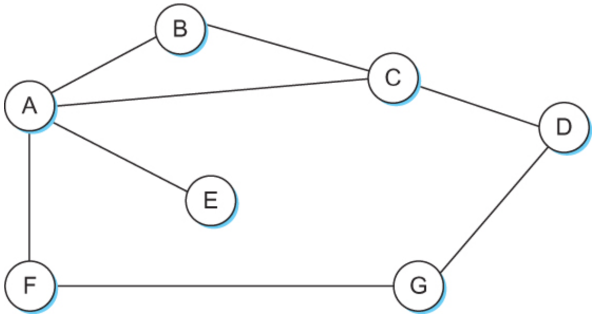


Count-to-infinity Problem – remedy

- Use an upper bound to force stop – i.e. 16

- Another technique to improve the time to stabilize routing is called *split horizon*
  - When a node sends a routing update to its neighbors, it does not send those routes it learned from each neighbor back to that neighbor

For example, if B has the route (E, 2, A) in its table, then it knows it learned this route from A, and so whenever B sends a routing update to A, it does not include the route (E, 2) in that update



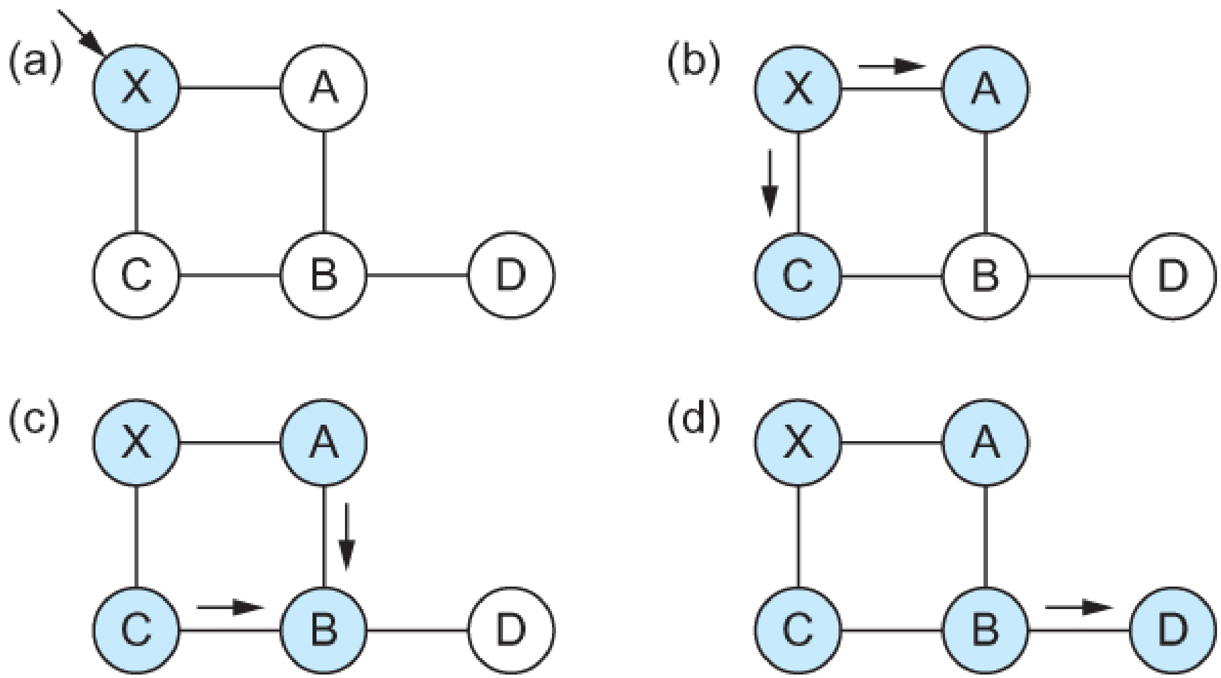
Link State Routing

Strategy: Send to all nodes (not just neighbors) information about directly connected links (not entire routing table).

- Link State Packet (LSP)
  - ID of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet
- Reliable Flooding
  - forward LSP to all nodes but one that sent it
  - start SEQNO = 0 and increment
  - decrement TTL of each stored LSP;
  - discard when TTL=0

Flood Link State Packets

Reliable Flooding

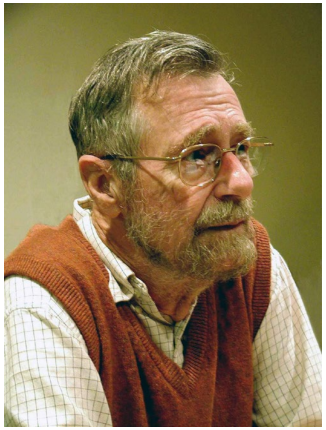


Flooding of link-state packets. (a) LSP arrives at node X; (b) X floods LSP to A and C; (c) A and C flood LSP to B (but not X); (d) flooding is complete

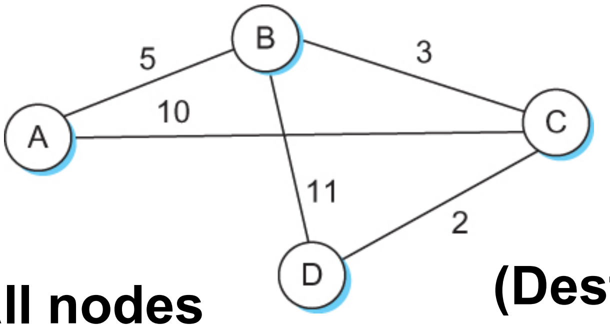
Link State Routing

- Each router computes its routing table from the LSP's it has collected.
  - The calculation is based on the Dijkstra's algorithm a.k.a., *forward search algorithm*

i. Each router maintains two lists, known as **Tentative** and **Confirmed**
ii. Each of these lists contains a set of entries of the form (Destination, Cost, NextHop)
iii. Algorithm converges when **Tentative** list become null



Shortest Path Routing-Forward Search



D has LSP's of all nodes

(Destination, Cost, NextHop)

Step	Confirmed	Tentative	Comments
1	(D,0,–)		Since D is the only new member of the confirmed list, look at its LSP.
2	(D,0,–)	(B,11,B) (C,2,C)	D's LSP says we can reach B through B at cost 11, which is better than anything else on either list, so put it on Tentative list; same for C.
3	(D,0,–) (C,2,C)	(B,11,B)	Put lowest-cost member of Tentative (C) onto Confirmed list. Next, examine LSP of newly confirmed member (C).
4	(D,0,–) (C,2,C)	(B,5,C) (A,12,C)	Cost to reach B through C is 5, so replace (B,11,B). C's LSP tells us that we can reach A at cost 12.
5	(D,0,–) (C,2,C) (B,5,C)	(A,12,C)	Move lowest-cost member of Tentative (B) to Confirmed, then look at its LSP.
6	(D,0,–) (C,2,C) (B,5,C)	(A,10,C)	Since we can reach A at cost 5 through B, replace the Tentative entry.
7	(D,0,–) (C,2,C) (B,5,C) (A,10,C)		Move lowest-cost member of Tentative (A) to Confirmed, and we are all done.

Distance-Vector vs Link-State

- **Distance-Vector (e.g. RIP):**
  - Each node talks with its neighbors only
  - Sends all information it knows - known distances to other nodes
  - Speed of convergence is slower than LS
  - Stabilization may not occur – count to infinity
  - Simple algorithm
- **Link-State (e.g. OSPF):**
  - Each node talks to all other nodes
  - Sends what it knows for sure - State of its directly connected links
  - Stabilizes quickly and it responds rapidly to network changes
  - Low traffic generation
  - Storage at each node is large
  - Uses reliable flooding of packets

Metrics – Cost of Links

- Assign 1 to all links – hop count
- Latency – take into account delay of the link
- Capacity – what is BW of each link
- Current Load – increase cost as load increases
- Queue length (average value between updates)

- Metrics are fixed by administrators – not dynamically changing due to stability issues.

More Practices

- What if you are given a network graph of bilateral links?
- What if a link fails when the algorithm is running?

Appendix – pseudo codes of Dijkstra's algorithm

- Dijkstra's Algorithm - Assume non-negative link weights
  - N: set of nodes in the graph
  - L(i, j): the non-negative cost associated with the edge between nodes i, j $\in$N and L(i, j) = $\propto$ if no edge connects i and j
  - Let s $\in$N be the starting node which executes the algorithm to find shortest paths to all other nodes in N
  - Two variables used by the algorithm
    - M: set of nodes incorporated so far by the algorithm
    - C(n) : the cost of the path from s to each node n
    - The algorithm

```
M = {s}
For each n in N - {s}
    C(n) = L(s, n)
while ( N ≠ M)
  M = M ∪ {w} such that C(w) is the minimum
                              for all w in (N-M)
  For each n in (N-M)
          C(n) = MIN (C(n), C(w) + L(w, n))
```

Appendix – pseudo codes of Dijkstra's algorithm

- The algorithm
  1) Initialize the **Confirmed** list with an entry for myself; this entry has a cost of 0
  2) For the node just added to the **Confirmed** list in the previous step, call it node **Next**, select its LSP
  3) For each neighbor (Neighbor) of **Next**, calculate the cost (Cost) to reach this Neighbor as the sum of the cost from myself to Next and from Next to Neighbor
     a) If Neighbor is currently on neither the **Confirmed** nor the **Tentative** list, then add (Neighbor, Cost, Nexthop) to the **Tentative** list, where Nexthop is the direction I go to reach Next
     b) If Neighbor is currently on the **Tentative** list, and the Cost is less than the currently listed cost for the Neighbor, then replace the current entry with (Neighbor, Cost, Nexthop) where Nexthop is the direction I go to reach Next
     c) If Neighbor on **Confirmed** list, then ignore it
  4) If the **Tentative** list is empty, stop. Otherwise, pick the entry from the **Tentative** list with the lowest cost, move it to the **Confirmed** list, and return to Step 2.