



*Corso di*  
***SISTEMI TELEMATICI***  
*a.a. 2009-2010*

**Livello rete: Algoritmi di instradamento**

# Algoritmo di Dijkstra



- L'algoritmo di Dijkstra individua il cammino a lunghezza minima tra un nodo  $s$  e tutti gli altri nodi di un grafo  $G$  procedendo in modo da aumentare progressivamente la distanza
- L'algoritmo procede a passi successivi
  - ✗ al passo  $k$ -esimo sono individuati i  $k$  nodi raggiungibili dal nodo sorgente tramite i cammini a costo minimo
  - ✗ tali  $k$  nodi formano l'insieme  $T$
  - ✗ al passo  $k+1$ -esimo è aggiunto un ulteriore nodo all'insieme  $T$  caratterizzato dal cammino a costo minimo dal nodo  $s$  che transita esclusivamente nei nodi dell'insieme  $T$
  - ✗ l'algoritmo termina quando si sono esplorati tutti i nodi



# Algoritmo di Dijkstra

- Notazioni

**N** : insieme dei nodi del grafo

**s** : nodo sorgente

**T** : insieme dei nodi raggiunti dall'algoritmo

**w(i,j)** : peso (costo) del ramo (i,j)

- $w(i,i) = 0$

- $w(i,j) \geq 0$  se i vertici i e j sono connessi  
direttamente

- $w(i,j) = \infty$  se i vertici i e j non sono connessi  
direttamente

**L(n)** : costo del cammino a costo minimo tra il nodo s ed  
il generico nodo n

# Algoritmo di Dijkstra



- Inizializzazione ( $k=1$ )

$$T = \{s\}$$

$$L(n) = w(s, n) \quad \text{per } n \neq s$$

- Aggiunta di un nodo (passo  $k$ )

- trovare il nodo  $x \notin T$  a distanza minima da  $s$ , cioè tale che:

$$L(x) = \min_{j \notin T} L(j)$$

- aggiungere all'insieme  $T$  il nodo  $x$  ed il ramo incidente a  $x$  che contribuisce a  $L(x)$

- Aggiornamento dei cammini minimi

$$L(n) = \min [L(n), L(x) + w(x, n)]$$

per tutti i valori di  $n \notin T$

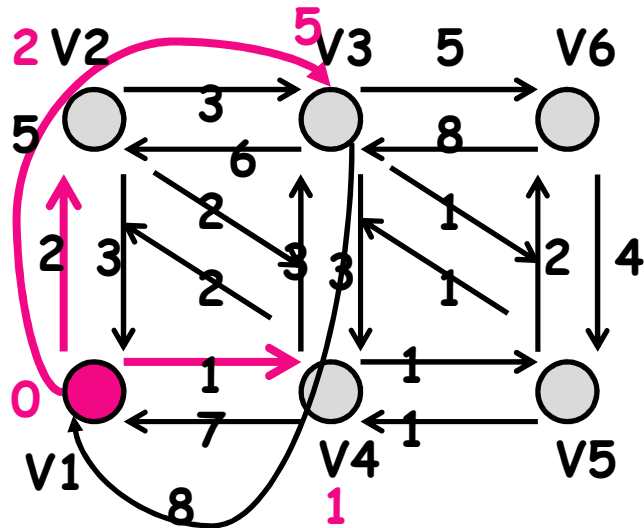


# Algoritmo di Dijkstra

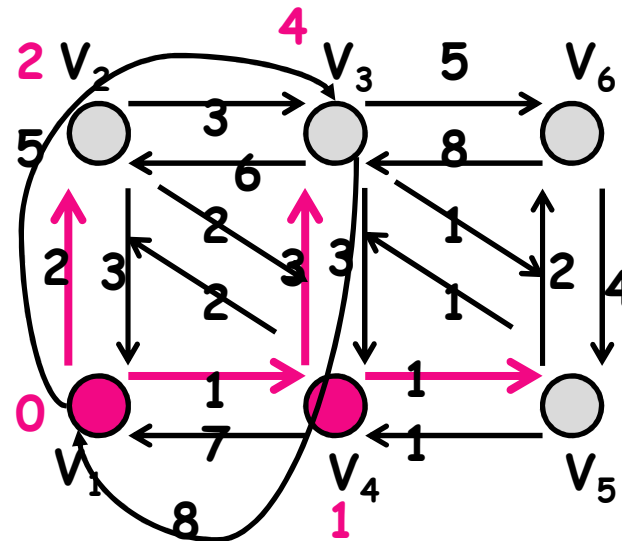
- Si noti che
  - al passo k-esimo viene aggiunto all'insieme T il k-esimo nodo ed è individuato il cammino a costo minimo tra tale nodo ed il nodo sorgente
  - questo cammino transita esclusivamente attraverso i nodi sinora compresi nell'insieme T
- L'algoritmo termina quando tutti i nodi sono stati aggiunti all'insieme T, ovvero  $T=N$



# Algoritmo di Dijkstra: esempio



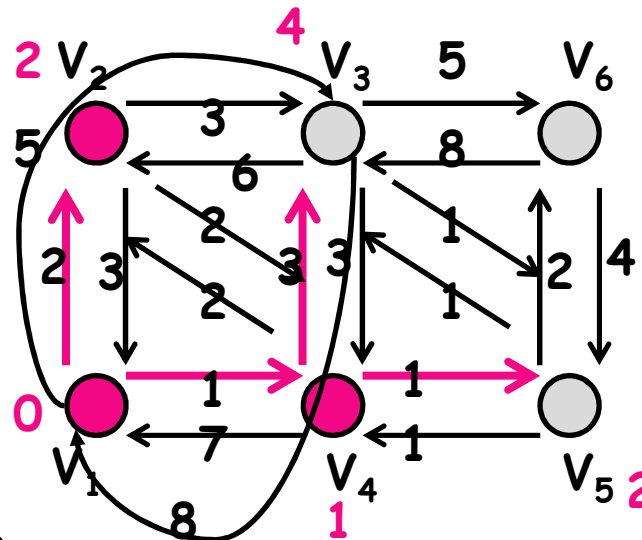
$T = \{1\}$



$T = \{1, 4\}$

$L(2)=2$   $L(4)=1$   
 $L(3)=5$   $L(i)=\infty$   $i=5,6$

$L(2)=2$   $L(3)=4$   
 $L(5)=2$   $L(6)=\infty$

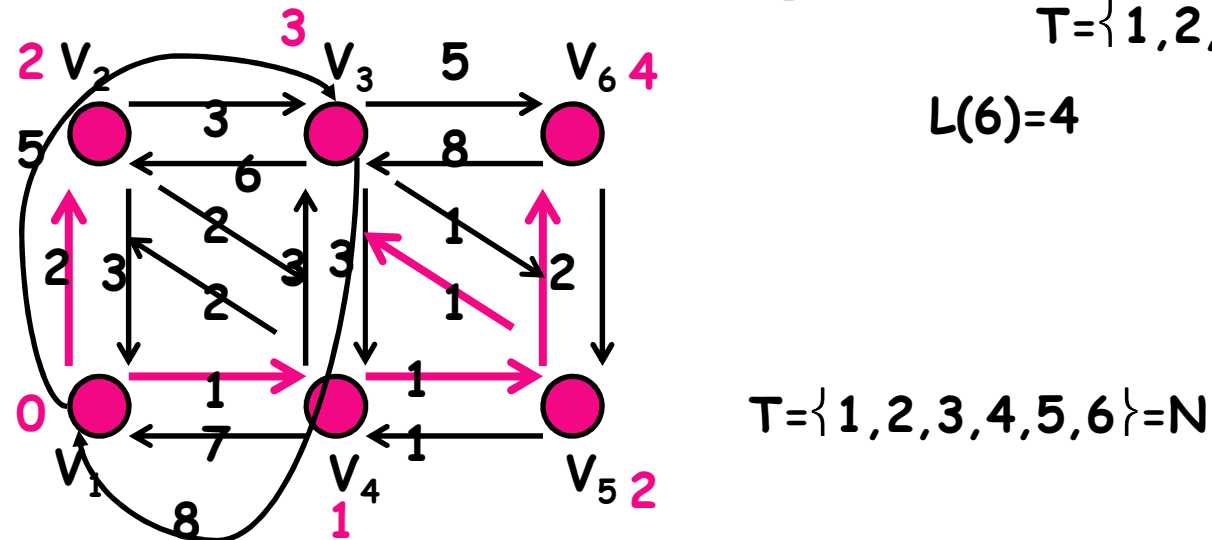
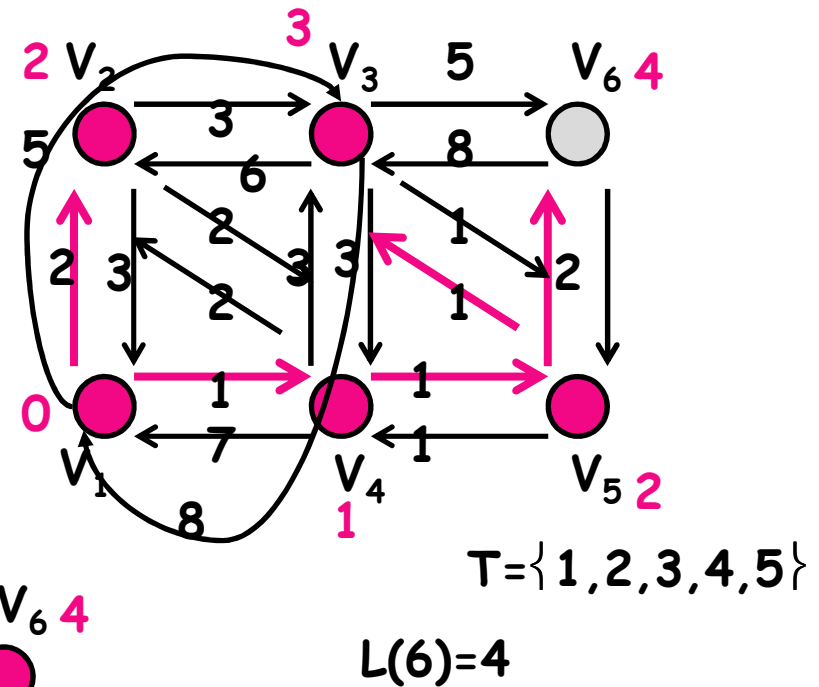
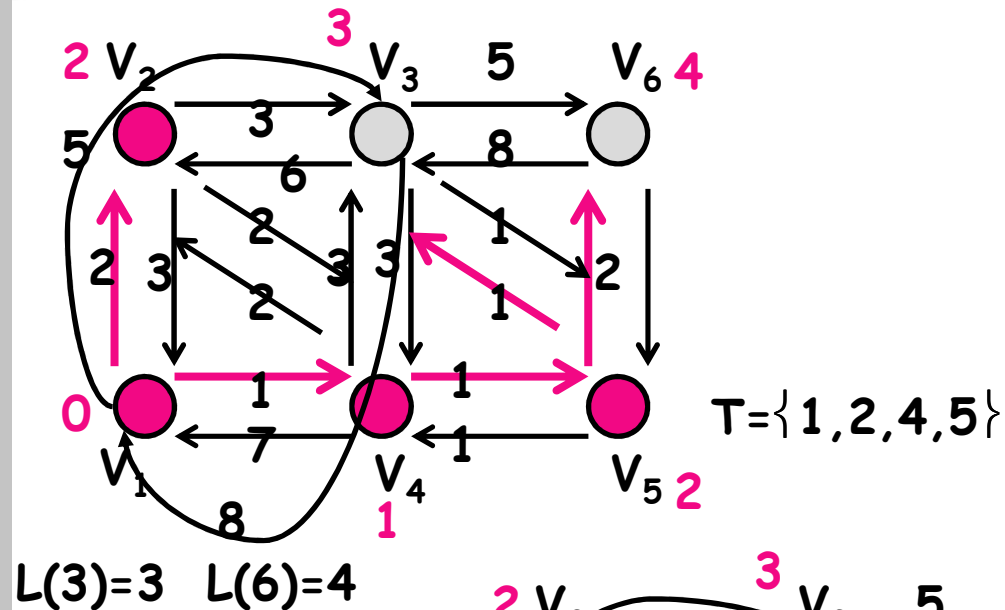


$T = \{1, 2, 4\}$

$L(3)=4$   
 $L(5)=2$   $L(6)=\infty$



# Algoritmo di Dijkstra: esempio



# Algoritmo di Dijkstra



- Al termine
  - $L(n)$  indica il costo del cammino a costo minimo tra i nodi  $s$  ed  $n$
  - l'insieme  $T$  è uno **spanning tree** del grafo di partenza contenente i cammini a costo minimo tra il nodo sorgente e tutti gli altri nodi
- La complessità dell'algoritmo è  $O(V^2)$ 
  - intuitivamente, ci sono  $|V|-1$  iterazioni (ad ogni iterazione si aggiunge un nodo in  $T$ ) e, per ogni iterazione, il numero di operazioni effettuate è proporzionale a  $|V|$





# A Link-State Routing Algorithm

## Dijkstra's algorithm

- net topology, link costs known to all nodes
  - ✗ accomplished via “link state broadcast”
  - ✗ all nodes have same info
- computes least cost paths from one node (“source”) to all other nodes
  - ✗ gives routing table for that node
- iterative: after  $k$  iterations, know least cost path to  $k$  dest.’s

## Notation:

- $w(i,j)$ : link cost from node  $i$  to  $j$ . cost infinite if not direct neighbors
- $L(v)$ : current value of cost of path from source to dest.  $v$
- $p(n)$ : predecessor node along path from source to  $v$ , that is next  $v$
- $T$ : set of nodes whose least cost path definitively known



Università della Calabria D.E.I.S.

# Dijkstra's Algorithm



1 *Initialization:*

2  $T = \{s\}$

3 for all nodes  $n$

4 if  $n$  adjacent to  $s$

5 then  $L(n) = w(s,n)$

6 else  $L(n) = \text{infty}$

7

8 *Loop*

9 find  $n$  not in  $T$  such that  $L(n)$  is a minimum

10 add  $n$  to  $T$

11 update  $L(k)$  for all  $k$  adjacent to  $n$  and not in  $T$ :

12  $L(k) = \min( L(k), L(n) + w(n,k) )$

13 /\* new cost to  $k$  is either old cost to  $k$  or known

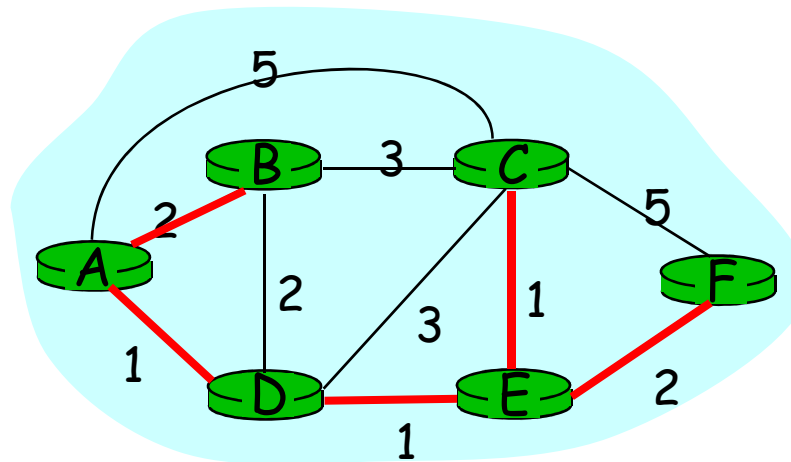
14 shortest path cost to  $n$  plus cost from  $n$  to  $k$  \*/

15 *until all nodes in  $T$*



# Dijkstra's algorithm: example

Step	start N	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
→ 0	A	2,A	5,A	1,A	infinity	infinity
→ 1	AD	2,A	4,D		2,D	infinity
→ 2	ADE	2,A	3,E			4,E
→ 3	ADEB		3,E			4,E
→ 4	ADEBC					4,E
5	ADEBCF					



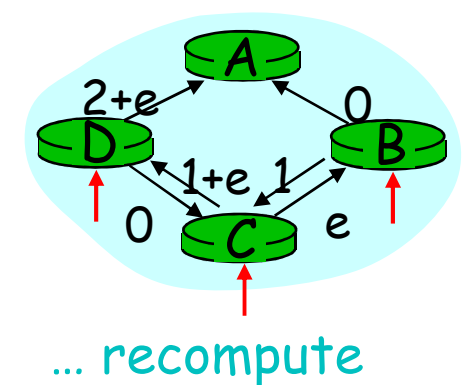
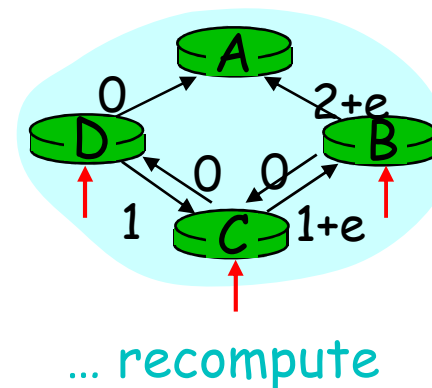
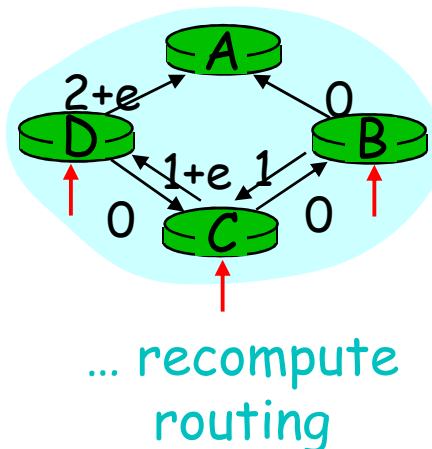
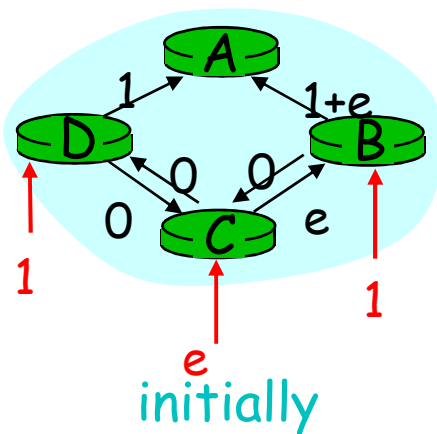
# Dijkstra's algorithm, discussion

Algorithm complexity:  $n$  nodes

- each iteration: need to check all nodes,  $k$ , not in  $T$
- $n*(n+1)/2$  comparisons:  $O(n^2)$
- more efficient implementations possible:  $O(n \log n)$

Oscillations possible:

- e.g., link cost = amount of carried traffic



# Algoritmo di Bellman-Ford

- L'algoritmo di Bellman-Ford individua il cammino a lunghezza minima tra un nodo  $s$  e tutti gli altri nodi di un grafo  $G$  **aumentando progressivamente il numero di rami**
- L'algoritmo procede a passi:
  - al primo passo individua i cammini minimi tra il nodo sorgente e gli altri nodi con il vincolo che i cammini contengano al più 1 ramo
  - al secondo passo si trovano i cammini minimi tra il nodo sorgente e gli altri nodi con il vincolo che i cammini contengano al più 2 rami
  - si itera il procedimento sino al valore massimo di rami possibili in un cammino

# Algoritmo di Bellman-Ford

## • Notazioni

**N** : insieme dei nodi del grafo

**s** : nodo sorgente

**h** : massimo numero di rami in un cammino  
correntemente consentito dall'algoritmo

**w(i,j)** : peso (costo) del ramo (i,j)

-  $w(i,i) = 0$

-  $w(i,j) \geq 0$  se i vertici i e j sono connessi  
direttamente

-  $w(i,j) = \infty$  se i vertici i e j non sono connessi  
direttamente

**$L_h(n)$**  : costo del cammino a costo minimo tra il nodo s  
ed il nodo n, con il vincolo che il numero di rami  
non sia superiore ad h

# Algoritmo di Bellman-Ford

- Inizializzazione

$L_0(n) = \infty$  per tutti i valori di  $n \neq s$

$L_h(s) = 0$  per tutti i valori di  $h$

- Aggiornamento

- per tutti i valori successivi di  $h \geq 0$  e per ogni  $n \neq s$ ,  
calcolare:

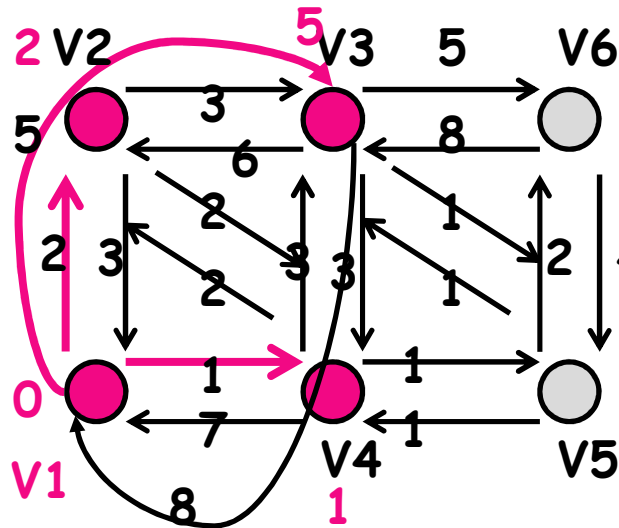
$$L_{h+1}(n) = \min_j [L_h(j) + w(j, n)] \quad (1)$$

- connettere il nodo  $n$  con il nodo predecessore  $j$  che raggiunge il minimo ed eliminare le connessioni di  $n$  con altri nodi predecessori individuati in precedenti iterazioni

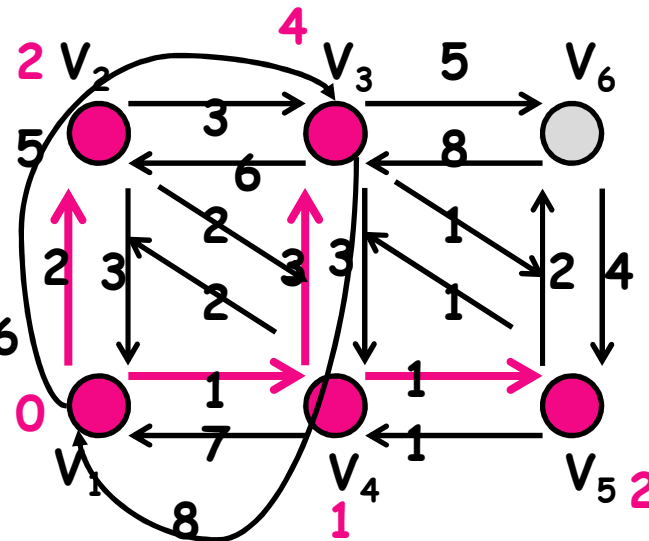
- Termine

- l'algoritmo termina quando all'iterazione  $h+1$  non cambia il valore dei percorsi minimi rispetto all'iterazione precedente, cioè  $L_{h+1}(n) = L_h(n)$  per ogni  $n$

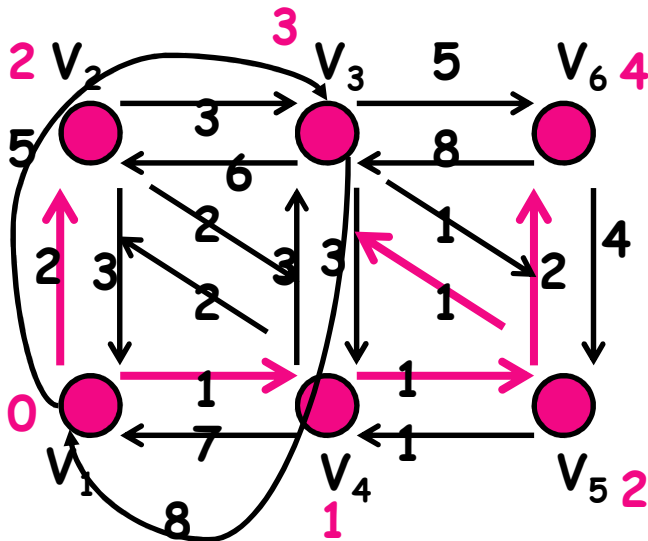
# Algoritmo di Bellman-Ford: esempio



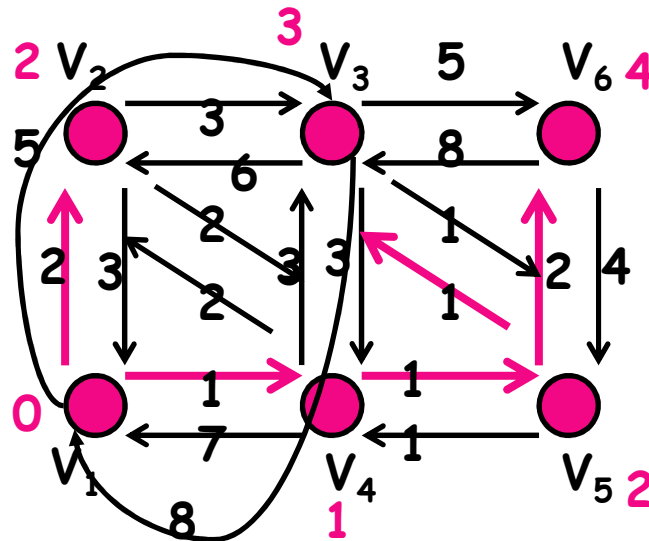
$h=1$   
 $L1(2)=2$   
 $L1(4)=1$   
 $L1(3)=5$   
 $L(i)=\infty \ i=5,6$



$h=2$   
 $L2(2)=2$   
 $L2(4)=1$   
 $L2(3)=4$   
 $L2(5)=2$   
 $L2(6)=\infty$



$h=3$   
 $L3(2)=2$   
 $L3(4)=1$   
 $L3(3)=3$   
 $L3(5)=2$   
 $L3(6)=4$



$h=4$



# Algoritmo di Bellman-Ford

- Al termine
  - si individua uno **spanning tree** del grafo di partenza che contiene i cammini a costo minimo tra il nodo sorgente e tutti gli altri nodi
- La complessità dell'algoritmo è  $O(|V||E|)$ , ovvero nel caso peggiore in cui  $E=|V|^2$ , si ha  $O(|V|^3)$ 
  - intuitivamente, ci sono  $|V|-1$  iterazioni del loop interno (1) e ogni iterazione richiede l'esame dei pesi di ogni ramo

# Bellman-Ford vs. Dijkstra

- I due algoritmi convergono (in condizioni statiche di topologia e costi dei link) alla stessa soluzione
- L'algoritmo di Bellman-Ford, nel caso peggiore, è più complesso; ma in molti casi pratici i due algoritmi si equivalgono
- L'algoritmo di Dijkstra richiede che un nodo conosca l'intera topologia della rete e il peso di tutti i rami
  - ✗ necessità di colloquio tra tutti i nodi
- L'algoritmo di Bellman-Ford richiede la conoscenza dello stato dei rami uscenti da un nodo insieme alle informazioni provenienti dai nodi vicini
  - ✗ necessità di colloquio solo tra nodi adiacenti (implementazione distribuita)



# Interior Gateway Protocols (IGP)



# Interior Gateway Protocols (IGP)

GGP

SPREAD

SPF e OSPF

RIP

HELLO

- Il modo in cui i protocolli IGP determinano le tabelle di instradamento può essere vario ed utilizzare svariati algoritmi, ognuno dei quali ha i suoi pro e contro

# Routing Algorithm classification

## Global or decentralized information?

### Global:

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

### Decentralized:

- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

## Static or dynamic?

### Static:

- routes change slowly over time

### Dynamic:

- routes change more quickly
  - × periodic update
  - × in response to link cost changes

# Protocolli IGP

- La maggior parte dei protocolli IGP si basa su due classi di filosofie di routing:
- Distance-Vector Routing
  - si basa sull'algoritmo di Bellman-Ford
  - più semplice
  - impegna meno risorse sul router
  - meno efficiente
  - adatto a reti piccole
- Link-State Routing
  - si basa sull'algoritmo di Dijkstra
  - molto più complesso
  - molto più efficiente
  - impegna più risorse
  - adatto a reti grandi

# Protocolli di routing

	Algoritmo	Protocollo
Link State	Dijkstra (SPF)	OSPF
Distance Vector Tradizionale	Bellman-Ford	GGP, RIP IGRP (Cisco)
Distance Vector Avanzato	DUAL	EIGRP (Cisco)

# Distance Vector

- I protocolli della classe Distance Vector (DV) si basano sull'algoritmo di **Bellman-Ford distribuito**
- Facciamo l'ipotesi di un funzionamento sincrono di tutti i nodi della rete:
- Ogni nodo  $s$  potrebbe eseguire, in modo sincrono a tutti gli altri nodi, le seguenti operazioni:
  - calcola  $L_1(n)$  ( $n \neq s$ ) ed invia il risultato ai nodi vicini
  - calcola  $L_2(n)$  ( $n \neq s$ ) ed invia il risultato ai nodi vicini
  - ...
  - calcola  $L_h(n)$  ( $n \neq s$ ) ed invia il risultato ai nodi vicini
- Cioè tutti i nodi del grafo eseguono in parallelo ogni iterazione dell'algoritmo di B.F. (incrementando  $h$  di 1)
- La procedura terminerebbe in al più  $N$  operazioni



# Algoritmo di Bellman-Ford distribuito

- Problema: la condizione di sincronia tra i nodi è difficilmente raggiungibile
  - quando e come si avvia l'algoritmo?
  - come si interrompe l'algoritmo se le condizioni di rete cambiano prima del suo termine?
- I protocolli DV utilizzano la versione **asincrona** dell'algoritmo di B.F.
  - ogni nodo  $s$  esegue in modo asincrono il calcolo
$$L(n) = \min_{j \in N(s)} [L(j) + w(j, n)]$$
usando l'ultima stima di  $L(j)$  ricevuta dai nodi vicini  $N(s)$
  - comunica periodicamente il risultato ai nodi vicini  $N(s)$

# Algoritmo di Bellman-Ford distribuito

- L'algoritmo distribuito converge a soluzione se la dinamica delle variazioni è più lenta della velocità di convergenza dell'algoritmo
- Possibilità del "Bad News Phenomenon"
  - variazioni particolari per cui la convergenza è molto lenta
  - il numero di iterazioni può essere molto elevato

# Distance Vector

- Ogni router mantiene, oltre alla tabella di instradamento, una struttura dati, detta distance vector, per ogni sua linea attiva
- Il distance vector contiene informazioni ricavate dalle tabelle di instradamento dei router collegati all'altro estremo della linea
- Il distance vector è un insieme di coppie [indirizzo-distanza]
  - ✗ la distanza è espressa tramite metriche classiche, quali numero di hop e costo
- Il calcolo delle tabelle di routing avviene tramite la fusione (merge) di tutti i distance vector associati alle linee attive di un router

# Distance Vector

- Tutte le volte che un router modifica la sua tabella di routing, la invia ai router adiacenti (cioè direttamente connessi alla stessa rete) sotto forma di distance vector
- Ciascun router apprende tramite un protocollo di neighbor greetings le informazioni relative ai nodi adiacenti
- Quando un router memorizza un distance vector nella sua struttura dati locale, verifica se sono presenti variazioni rispetto al distance vector precedentemente memorizzato: in caso affermativo ricalcola le tabelle di instradamento fondendo tutti i distance vector delle linee attive

# Distance Vector: un router





## Distance Vector: un router

**Il protocollo DV prevede che il router operi così:**

- 1) quando riceve un messaggio (distance vector) da un router adiacente confronta ogni coppia (destinazione, costo) col contenuto della sua tabella di routing e:**
  - a) se la destinazione non è in tabella allora crea una nuova entry per la nuova destinazione**
  - b) se la destinazione è in tabella e il next hop è il router adiacente che ha fatto l'annuncio allora aggiorna il costo ed il next hop nella entry**
  - c) se la destinazione è nella tabella e il costo indica un percorso migliore allora aggiorna il costo e il next hop nella entry**
- 2) ad intervalli regolari trasmette ai router adiacenti un messaggio (distance vector) che riporta tutte le coppie (destinazione, costo) contenute nella tabella di routing**

# Distance Vector: notazioni

Ogni nodo mantiene al suo interno tre vettori

$$W_x = \begin{bmatrix} w(x,1) \\ \vdots \\ w(x,M) \end{bmatrix} \quad L_x = \begin{bmatrix} L(x,1) \\ \vdots \\ L(x,N) \end{bmatrix} \quad R_x = \begin{bmatrix} R(x,1) \\ \vdots \\ R(x,N) \end{bmatrix}$$

Vettore dei costi  
uscenti dal nodo

Vettore delle distanze  
minime verso le altre reti

Vettore next hop  
verso le altre reti

- ✱  $M$  : numero di reti a cui il nodo  $x$  è direttamente connesso
- ✱  $w(x,i)$  : costo associato al ramo uscente dal nodo  $x$  verso la rete  $i$   
( $1 \leq i \leq M$ )
- ✱  $N$  : numero di reti nella configurazione
- ✱  $L(x,j)$  : stima della lunghezza del cammino minimo dal nodo  $x$  alla rete  $j$  (distance vector)
- ✱  $R(x,j)$  : next-hop router nel cammino a minima lunghezza dal nodo  $x$  alla rete  $j$

# Distance Vector: algoritmo

- Ogni nodo applica l'algoritmo di Bellman-Ford in modo distribuito
- Periodicamente (circa 30s) ogni nodo scambia il suo distance vector  $L_x$  con quelli dei nodi vicini
- Sulla base delle informazioni contenute in tutti i distance vector ricevuti, il nodo  $x$  aggiorna i suoi vettori nel seguente modo:

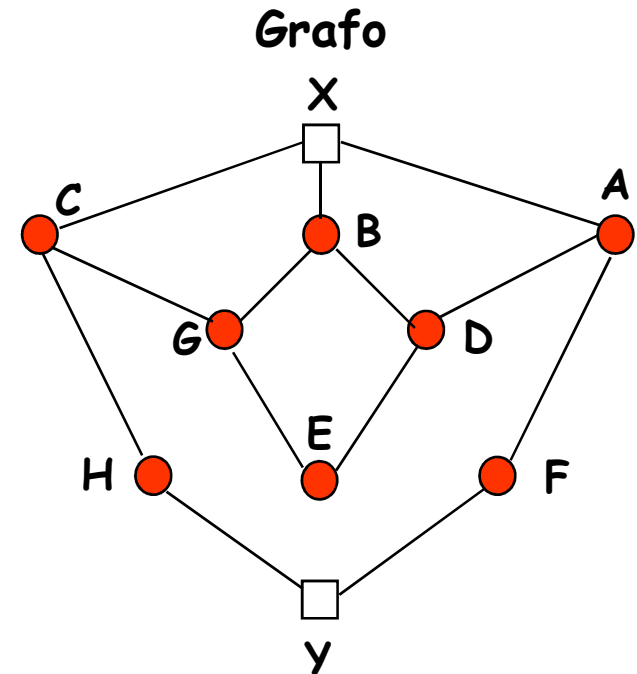
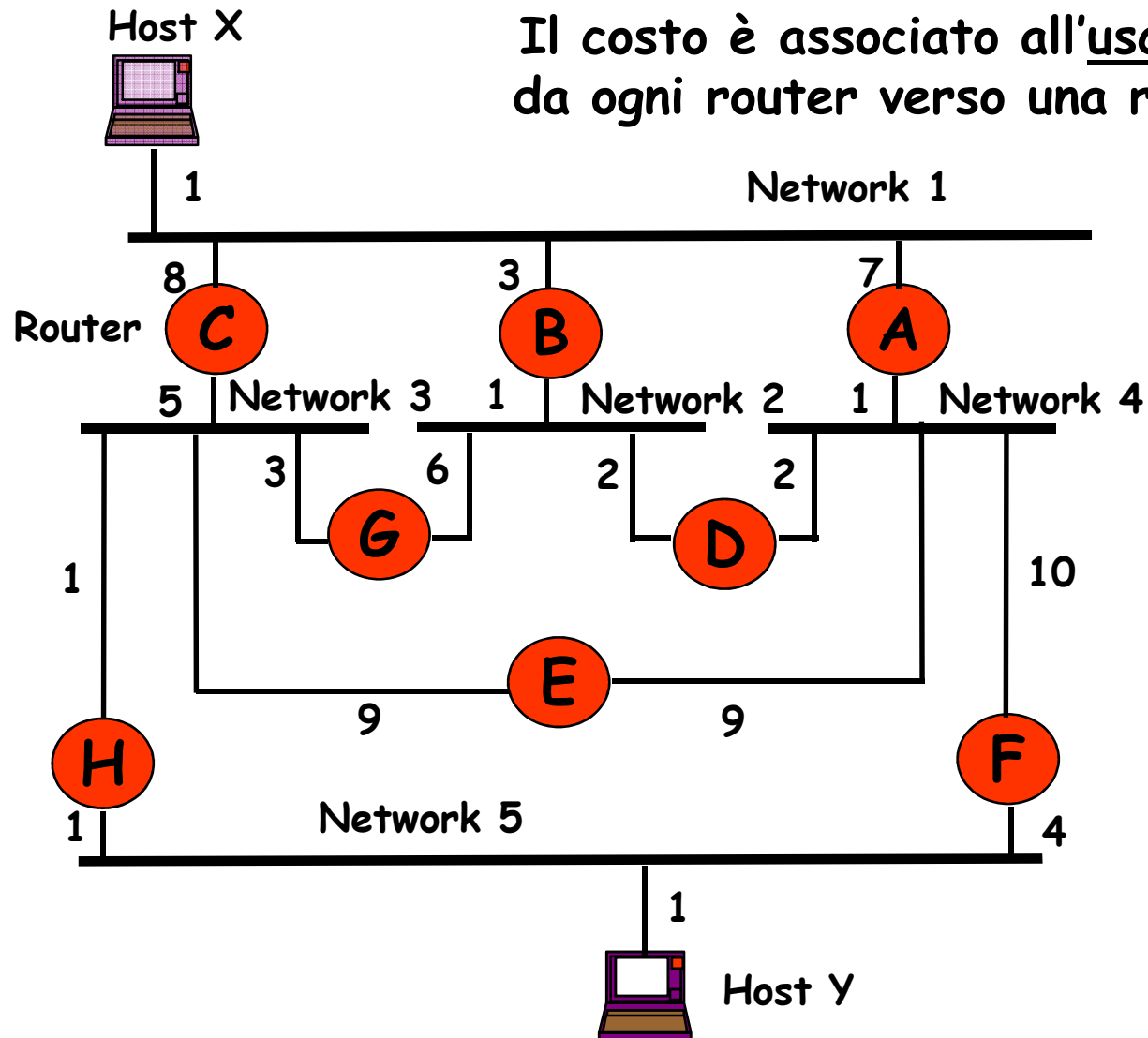
$$L(x,j) = \min_{y \in A} [L(y,j) + w(x, N_{xy})]$$

$$R(x,j) = y$$

- ✕  $A$  : insieme dei nodi vicini al nodo  $x$
- ✕  $N_{xy}$  : rete che interconnette i nodi  $x$  e  $y$



# Distance Vector: esempio



# Distance Vector: esempio

Router A	
Net	R/L
1	D/6
2	D/3
3	D/6
4	-/1
5	F/5

Router B	
Net	R/L
1	-/3
2	-/1
3	G/4
4	D/3
5	G/5

Router C	
Net	R/L
1	-/8
2	B/9
3	-/5
4	A/9
5	H/6

Router D	
Net	R/L
1	B
2	-
3	G
4	-
5	F

Router E	
Net	R/L
1	D/14
2	D/11
3	-/9
4	-/9
5	H/10

Router F	
Net	R/L
1	H/13
2	H/11
3	H/5
4	-/10
5	-/4

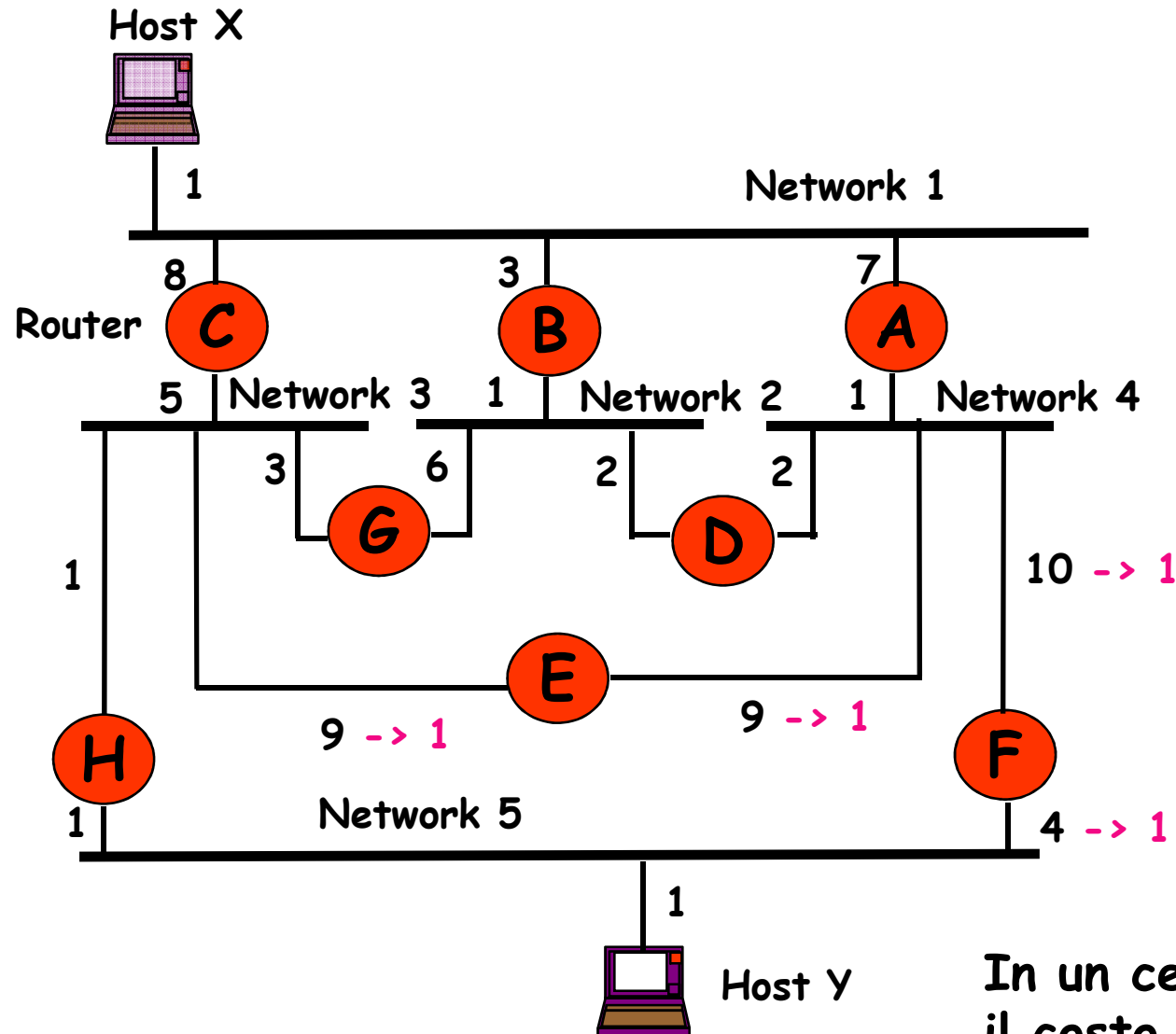
Router G	
Net	R/L
1	B
2	-
3	-
4	D
5	H

Router H	
Net	R/L
1	C
2	G
3	-
4	G
5	-

Host X	
Net	R/L
1	-/1
2	B/2
3	B/5
4	A/2
5	A/6

Tabelle di Routing iniziali

# Distance Vector : esempio



Host X		
Net j	R(X,j)	L(X,j)
1	-	1
2	B	2
3	B	5
4	A	2
5	A	6

Tabella di Routing iniziale

In un certo istante, varia il costo (ritardo) di alcuni link



# Distance Vector : esempio

I router E ed F comunicano ai nodi vicini i nuovi valori di  $L(E,j)$   $L(F,j)$ .  
I nodi vicini all'host X, cioè A, B e C, apprendono del cambiamento, aggiornano i distance vector e li trasmettono al nodo X.

Router A	
Net j	$L(A,j)$
1	6
2	3
3	E/2
4	1
5	F/2

Router B	
Net j	$L(B,j)$
1	3
2	1
3	4
4	3
5	D/4

Router C	
Net j	$L(C,j)$
1	8
2	E/8
3	5
4	E/6
5	6

Distance vector verso l'Host X  
(dopo la variazione)

Host X		
Net	$R(X,j)$	$L(X,j)$
1	-	1
2	B	2
3	A	3
4	A	2
5	A	3

Tabella di Routing finale

# Link State

- Un protocollo di tipo Link state (LS) assume che ogni router dispone della mappa completa della rete su cui calcolare gli instradamenti ottimali utilizzando l'algoritmo di Dijkstra
- La mappa della rete è costruita direttamente dai router tramite l'utilizzo di Link State Packet (LSP)
- Quando un router si attiva, determina il costo dei link su ciascuna delle sue interfacce di rete
- Ogni router, tramite protocolli di neighbor greetings, apprende la propria situazione locale (indirizzi delle sottoreti connesse e dei router direttamente raggiungibili, ovvero dei nodi adiacenti)
- Le informazioni di adiacenza sono incluse in un LSP che viene inviato a tutti gli altri router della rete

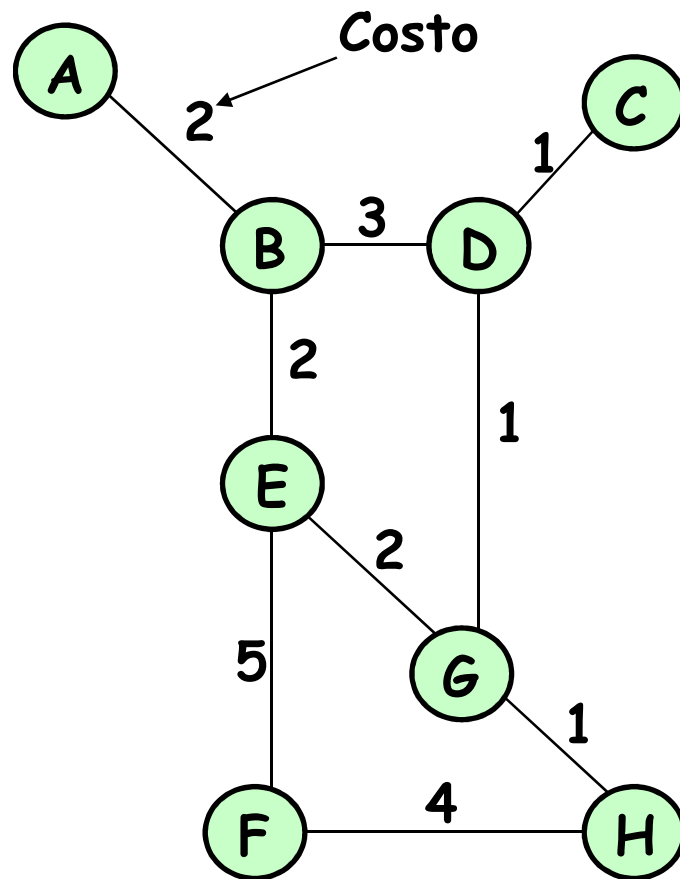
# Link State

- Il LSP generato da ogni router contiene:
  - × identificativo del router che emette il LSP
  - × stato di ogni link connesso al router
  - × identità di ogni vicino connesso all'altro estremo del link
  - × costo del link
  - × numero di sequenza per il LSP, utilizzato per individuare diverse versioni emesse dallo stesso router per scegliere la più recente
- Ogni router contiene un LSP database in cui memorizza il LSP più recente generato da ogni router; grazie a questo database ogni router si costruisce una mappa della rete con i costi associati

# Link State

- Il LSP database, ovvero la mappa logica della rete, non è ancora la tabella di instradamento, ma è un'informazione necessaria e sufficiente affinché un router calcoli le sue tabelle di instradamento
- Ogni router calcola la sua tabella di instradamento, indipendentemente dagli altri router, applicando alla mappa della rete l'algoritmo di Dijkstra, che trasforma una rete magliata in una rete ad albero scegliendo i percorsi a più basso costo, e in cui la radice è il router che sta applicando l'algoritmo
- Da questo albero vengono estratte per ogni destinazione il next-hop e il costo totale del percorso

# LSP Database



## LSP Database

A	B/2		
B	A/2	D/3	E/2
C	D/1		
D	B/3	C/1	G/1
E	B/2	F/5	G/2
F	E/5	H/4	
G	D/1	E/2	H/1
H	F/4	G/1	

(replicato su ogni Router)



# LSP: Tabella di routing

- Ogni router calcola indipendentemente le sue tabelle di routing applicando alla mappa della rete l'algoritmo di Dijkstra o SPF (Shortest Path First)

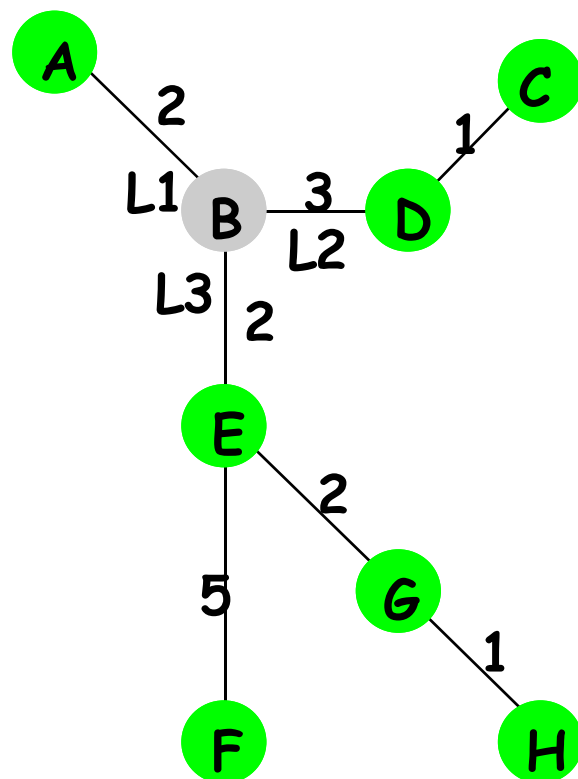


Tabella di  
routing di B

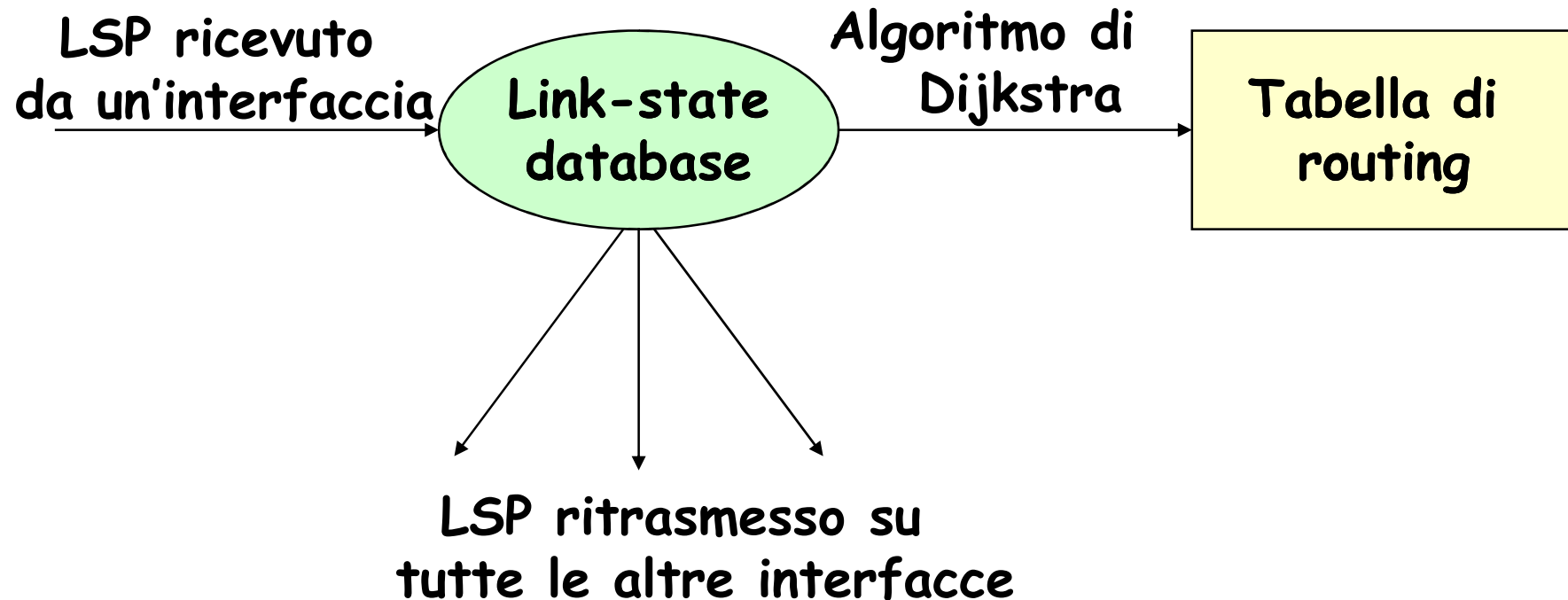
A	L1
C	L2
D	L2
E	L3
F	L3
G	L3
H	L3

# Link State

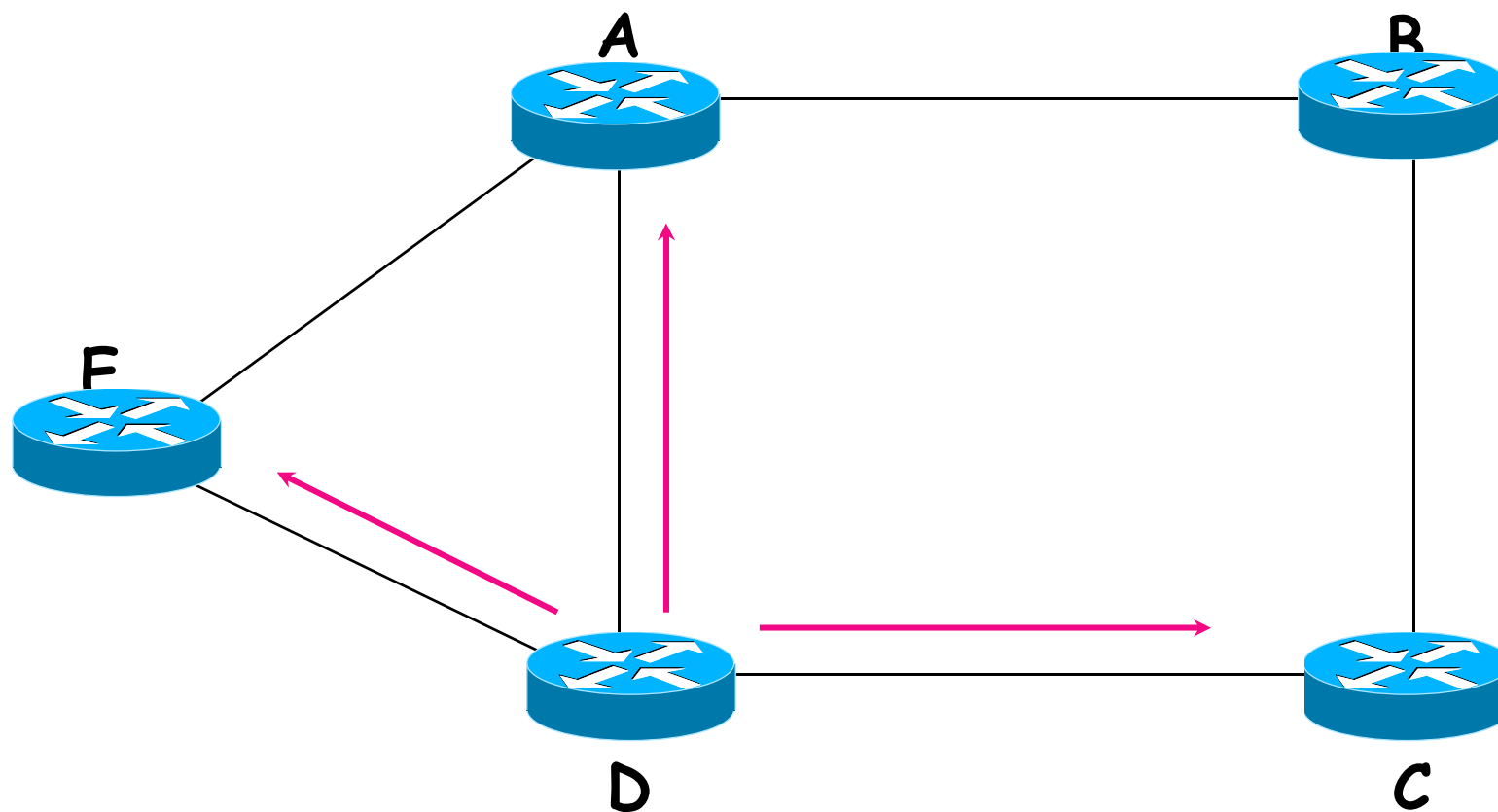
- Si noti la differenza con il distance vector: in quel caso i router cooperano per calcolare le tabelle di instradamento, qui i router cooperano per mantenere aggiornata la mappa della rete, poi ogni router calcola la propria tabella di instradamento in modo autonomo
- In caso di variazioni significative sullo stato dei link, il router invia un nuovo LSP in rete
- Un aspetto critico in protocolli del tipo Link State è la sincronizzazione tra i router; se i router hanno diversi LSP database, allora sono possibili dei loop

# Link State: operazione di un router

- Il LSP è trasmesso in flooding su tutti i link del router
- Il flooding è stato pensato per assicurare la più rapida diffusione e l'aggiornamento di eventuali variazioni di topologia

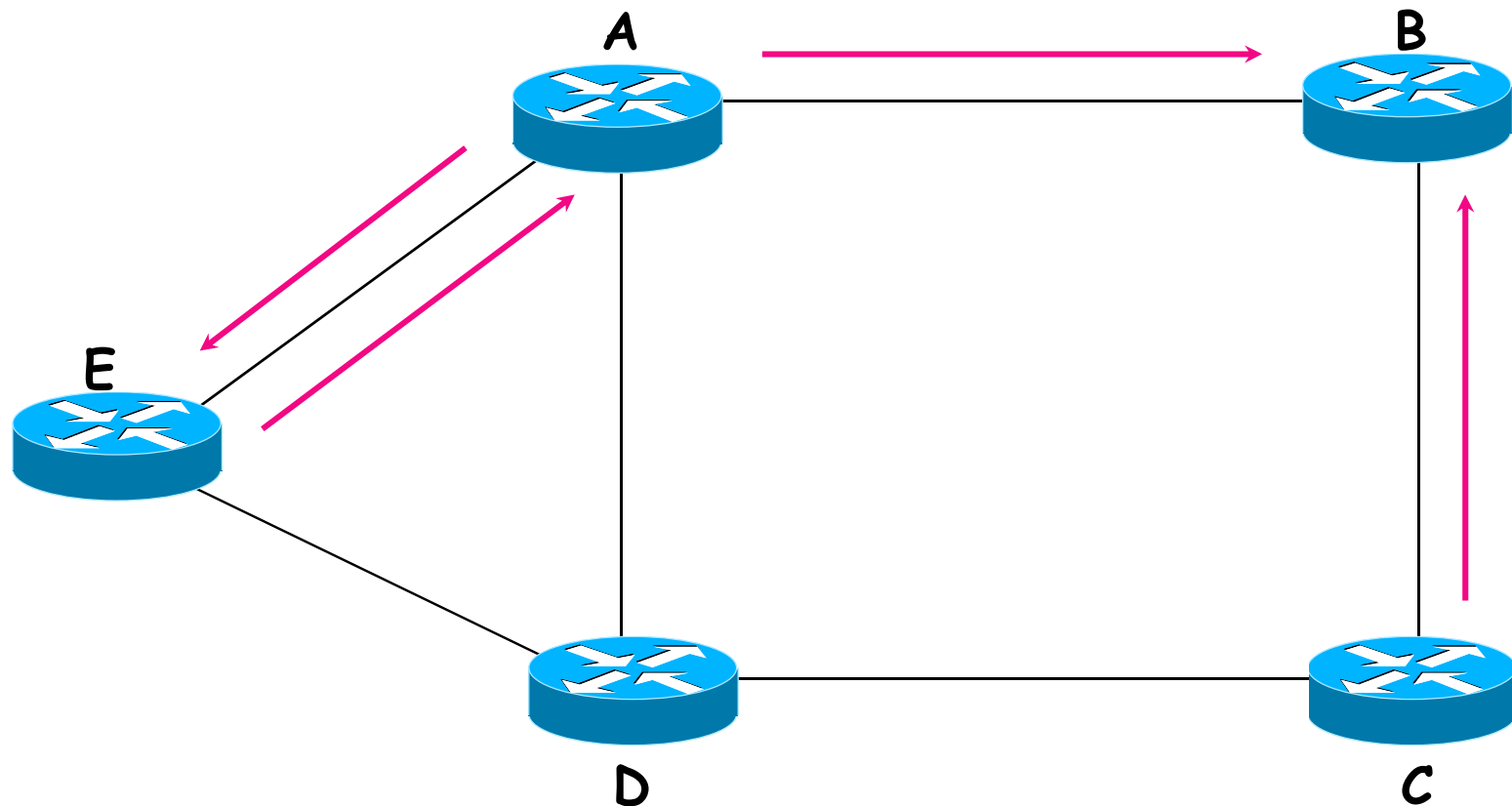


# Flooding



LSP trasmesso dal router D  
a tutti i router adiacenti

# Flooding



I router A, C, E ritrasmettono il LSP di D su tutte le interfacce tranne quella da cui lo hanno ricevuto

# Distance Vector vs. Link State

- **Distance-Vector**
  - ✗ ogni router invia le informazioni di routing ai router adiacenti
  - ✗ l'informazione trasmessa è una stima del costo verso tutte le reti
  - ✗ l'informazione è trasmessa su base periodica regolare
  - ✗ un router determina l'informazione sul next-hop usando l'algoritmo di Bellman-Ford distribuito sulle stime dei costi ricevute
- **Link-State**
  - ✗ ogni router invia le informazioni di routing a tutti gli altri router
  - ✗ l'informazione trasmessa è il valore esatto del costo dei link verso le reti adiacenti
  - ✗ l'informazione è trasmessa quando avviene un cambiamento
  - ✗ un router costruisce prima una descrizione della topologia della rete e poi usa un algoritmo di routing qualsiasi per calcolare le informazioni sul next-hop (tipicamente Dijkstra)