ALGORITMO BELLMAN-FORD

È un algoritmo centralizzato proposto nel 1957.
Contrariamente a Dijkstra non fissa un nodo di partenza, bensì quello di destinazione

h indica l'iterazione
D_i^h distanza da nodo i a destinazione al passo h

i) inizializzazione

$$D_d^h = \emptyset \quad \forall h$$

d = destinazione

$$D_i^{\emptyset} = \infty \quad \forall i \neq d$$

ii) selezione nodo i nel percorso verso destinazione da k

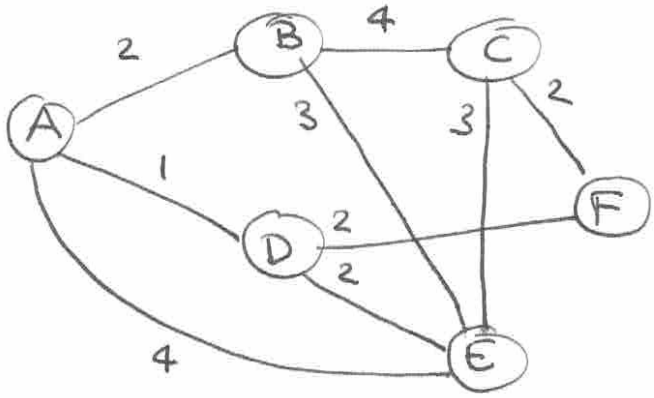
$$D_k^{h+1} = \min_j \{ \ell(j,k) + D_j^h , D_k^h \} \quad \forall k \neq d$$

$$i = \underset{k \neq d}{\arg\min}\, D_k^{h+1}$$

iii) stop se $D_i^{h+1} = D_i^h \quad \forall i$ altrimenti (ii)

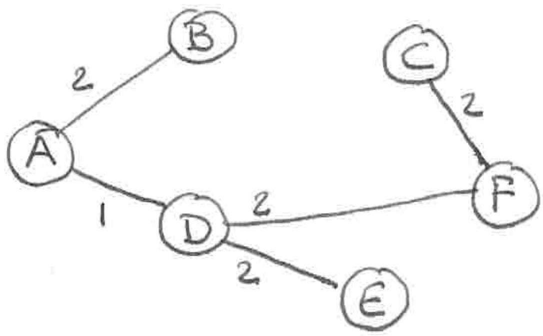
Esempio

Fisso destinazione $d = A$



h (passo)	D_A^h	D_B^h	D_C^h	D_D^h	D_E^h	D_F^h
$\emptyset$	$\emptyset$	∞	∞	∞	∞	∞
1	$\emptyset$	2 (B−A)	∞ (C−A)	1 (D−A)	4 (E−A)	∞ (F−A)
2	$\emptyset$	2	6 (C−B−A)	1	3 (E−D−A)	3 (F−D−A)
3	$\emptyset$	2	5 (C−F−D−A)	1	3	3
4	$\emptyset$	2	5	1	3	3

Nota: fino ad h hop al passo h.



ALGORITMO DISTANCE VECTOR

È uno degli algoritmi proattivi distribuiti più popolari. Rappresenta una versione distribuita di B-F.

Ogni nodo mantiene una tabella di instradamento che per ogni destinazione memorizza:

- indirizzo destinatario
- indirizzo del prossimo nodo (next hop) a cui inviare il pkt verso il destinatario
- costo dell'intero collegamento per raggiungere la destinazione

⇓

- vettore di distanze (costi) verso i possibili nodi di destinazione <u>DV</u>

i) I nodi connessi direttamente si scambiano i DV.

$DV^{(i)}$ distance vector del nodo i.

ii) Quando un nodo i riceve il $DV^{(j)}$ dal nodo j verifica se per ogni possibile destinazione k si ha

$$\ell(i,j) + DV_k^{(j)} \underset{?}{<} DV_k^{(i)}$$

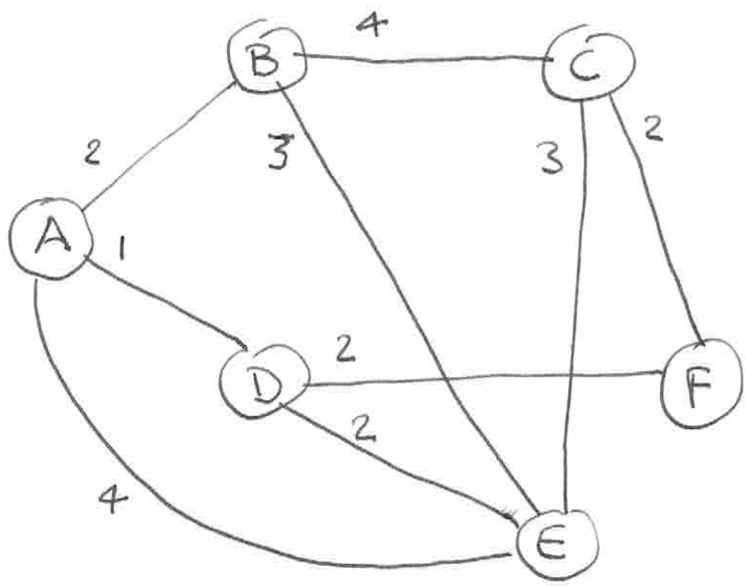
Se si allora i aggiorna la sua tabella di instradamento verso k con

$$DV_k^{(i)} = DV_k^{(j)} + \ell(i,j)$$

e come "next hop" passa da j per arrivare a k.

Altrimenti distanze e $\underbrace{\text{next hop}}_{NH}$ rimangono inalterati

Esempio



$i = A$, vicini $j = B, D, E$

k	$DV^{(B)}$	$DV^{(D)}$	$DV^{(E)}$
A	2	1	4
B	∅	∞	3
C	4	∞	3
D	∞	∅	2
E	3	2	∅
F	∞	2	∞

situazione iniziale

Al nodo $i = A$

k	$DV^{(A)}$	$NH^{(A)}$
A	$\emptyset$	A
B	2	B
C	∞	/
D	1	D
E	4	E
F	∞	/

situazione iniziale

entra $DV^{(B)}$

k	$DV^{(A)}$	$NH^{(A)}$
A	$\emptyset$	A
B	2	B
C	6	B
D	1	D
E	4	E
F	∞	/

entra $DV^{(D)}$

k	$DV^{(A)}$	$NH^{(A)}$
A	$\emptyset$	A
B	2	B
C	6	B
D	1	D
E	3	D
F	3	D

entra $DV^{(E)}$

k	$DV^{(A)}$	$NH^{(A)}$
A	$\emptyset$	A
B	2	B
C	6	B
D	1	D
E	3	D
F	3	D

- gli altri nodi aggiornano allo stesso modo (continuando questa riga cambierà in $\boxed{C \quad 5 \quad E}$)
- il nodo A sa che per andare verso F deve instradare il pkt verso D e la distanza complessiva sarà 3.

In maniera automatica e distribuita si costruiscono le tabelle di routing per ogni nodo verso gli altri nodi.

È necessario bilanciare la periodicità con cui si trasmette il DV: compromesso fra dinamicità dell'instradamento e l'overhead di segnalazione.

Un problema del DV è il <u>count-to-infinity</u>

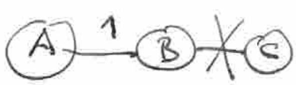
es. 3 nodi in cascata

iniziale: $DV_c^{(A)} = 2$, $NH_c^{(A)} = B$

$DV_c^{(B)} = 1$, $NH_c^{(B)} = C$

supponiamo si rompa/cessi il collegamento B-C

$DV_c^{(A)} = 2$, $NH_c^{(A)} = B$



$DV_c^{(B)} = \infty$, $NH_c^{(B)} = /$

I scambio DV

$DV_c^{(A)} = \infty$, $NH_c^{(A)} = /$

$\rightarrow DV_c^{(B)} = DV_c^{(A)} + \ell(B,A) = 3$, $NH_c^{(B)} = A$

II scambio DV

$DV_c^{(A)} = DV_c^{(B)} + \ell(A,B) = 4$, $NH_c^{(A)} = B$

$DV_c^{(B)} = \infty$, $NH_c^{(B)} = /$

⋮

ALGORITMO LINK-STATE

Migliora il DV eliminando il problema del count-to-infinity.

La topologia della rete è disseminata periodicamente mediante un pkt LSP (link-state packet)

Ogni LSP contiene l'indirizzo del nodo, quello dei nodi vicini e il costo delle linee verso i nodi vicini.

Controlled flooding di LSP per inviarlo a tutte le linee di uscita tranne la linea verso nodo da cui è stato ricevuto

Ogni LSP contiene un numero di sequenza progressivo per poter verificare quale è più aggiornato e dopo quanto considerare la informazione sullo stato dei link ~~troppo~~ vecchia (max age)

Se un nodo si rompe allora manca la propagazione dell'informazione. Per questo i vicini vengono scoperti tramite un pkt di "hello" che periodicamente un nodo invia ai propri vicini (1 hop) dalle proprie linee di uscita.

"hello" contiene la lista dei vicini.

Chi riceve "hello" verifica se il proprio ID è presente nella lista e in tal caso si appresta ad ottenere da lui lo stato del link tramite LSP.

Se non è presente nella lista memorizza il vicino per potergli trasmettere successivamente lo stato del link via LSP.

Esistono altri algoritmi di routing

- Ad-hoc on-demand distance vector (AODV) (reattivo)

- Hot potato