RETI DI CALCOLATORI

Autunno 2018

docente: Laura Ricci

laura.ricci@unipi.it

Lezione 3:
INTRADOMAIN ROUTING: LINK STATE ROUTING

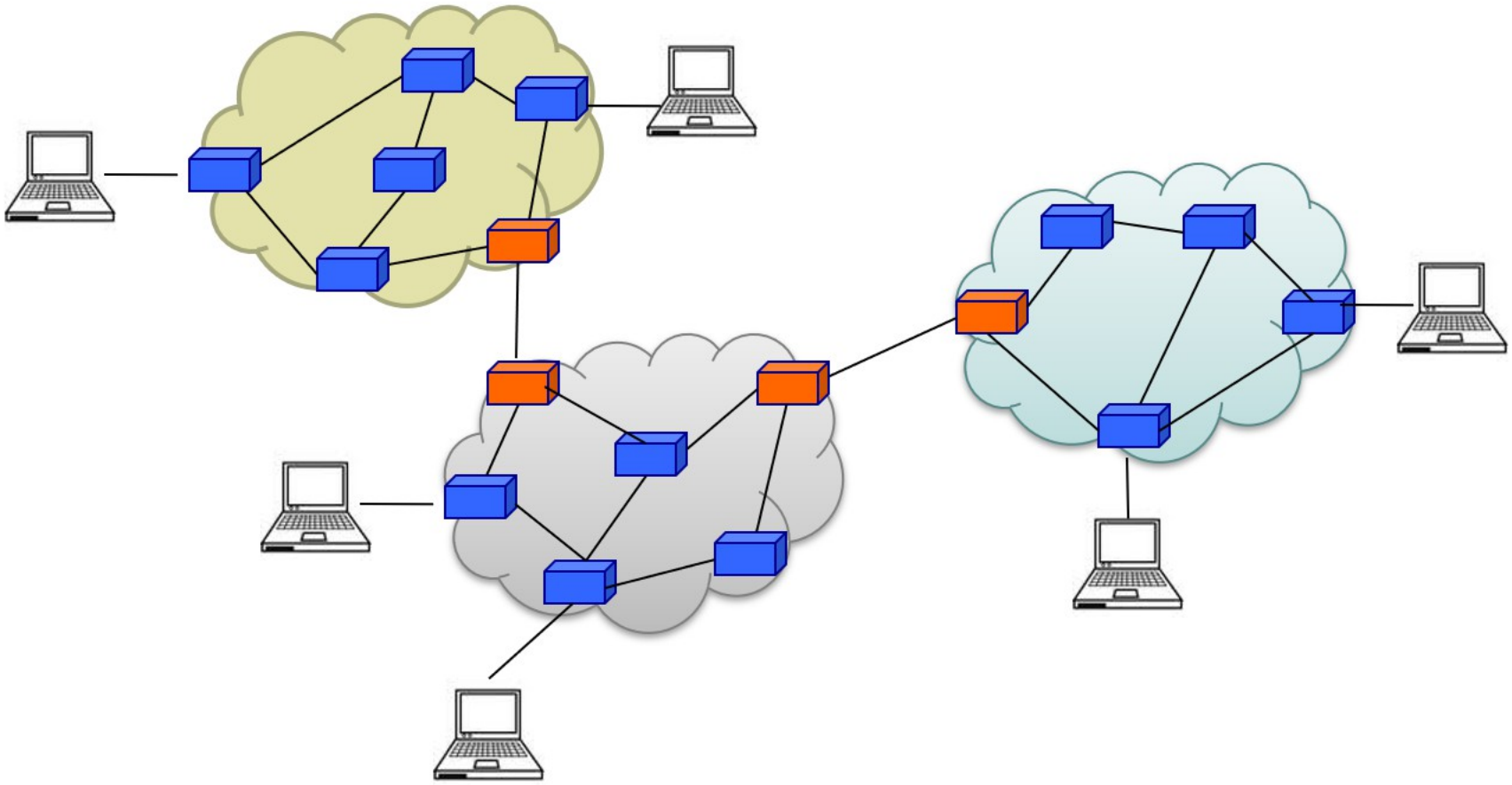
27/09/2018

parte di queste slides sono ricavati da slides pubblicate in corsi universitari tenuti da colleghi italiani e stranieri

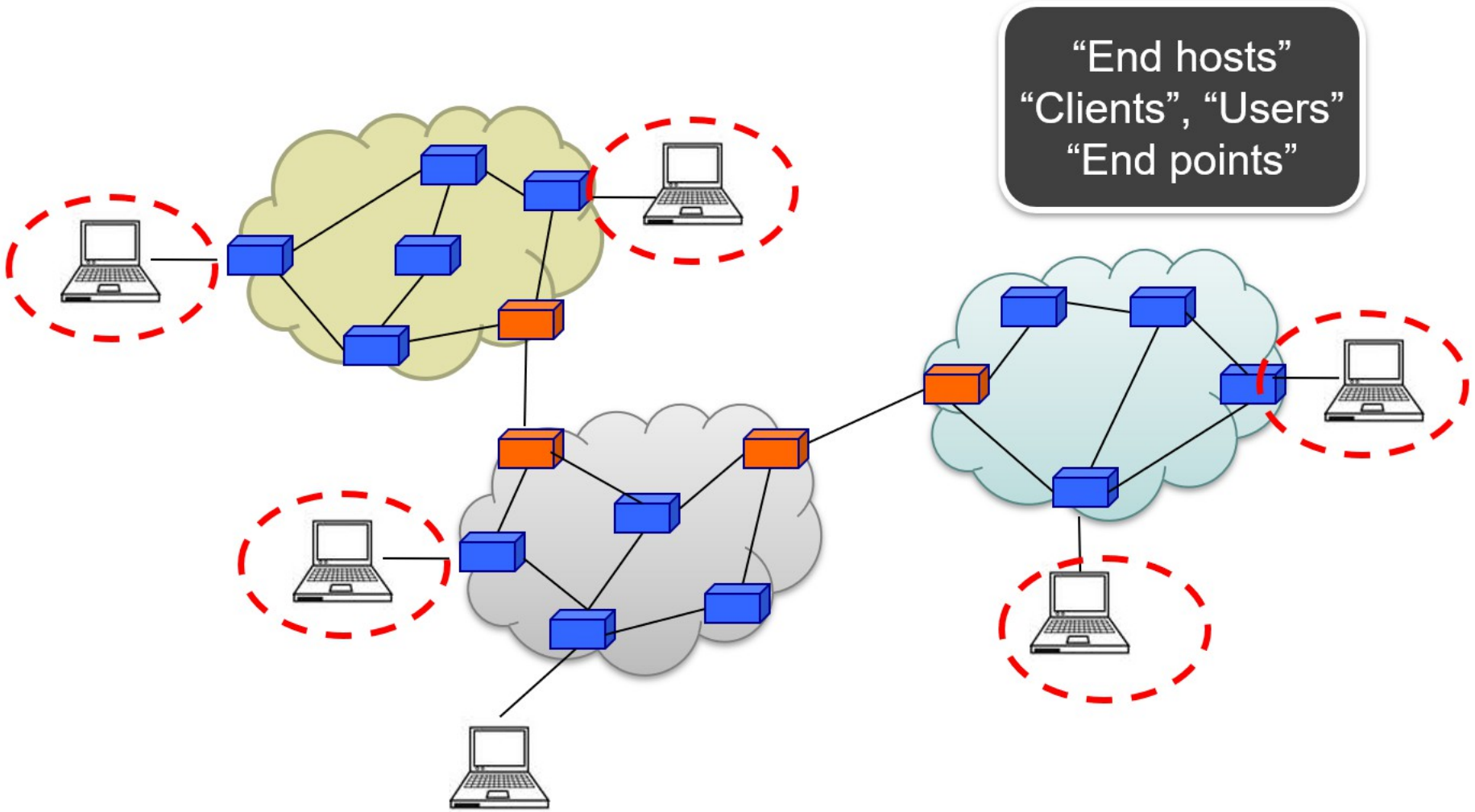
MATERIALE DIDATTICO

- Forouzan
  - Paragrafo 4.3.1
  - Paragrafo 4.3.2 (Routing a stato del collegamento)

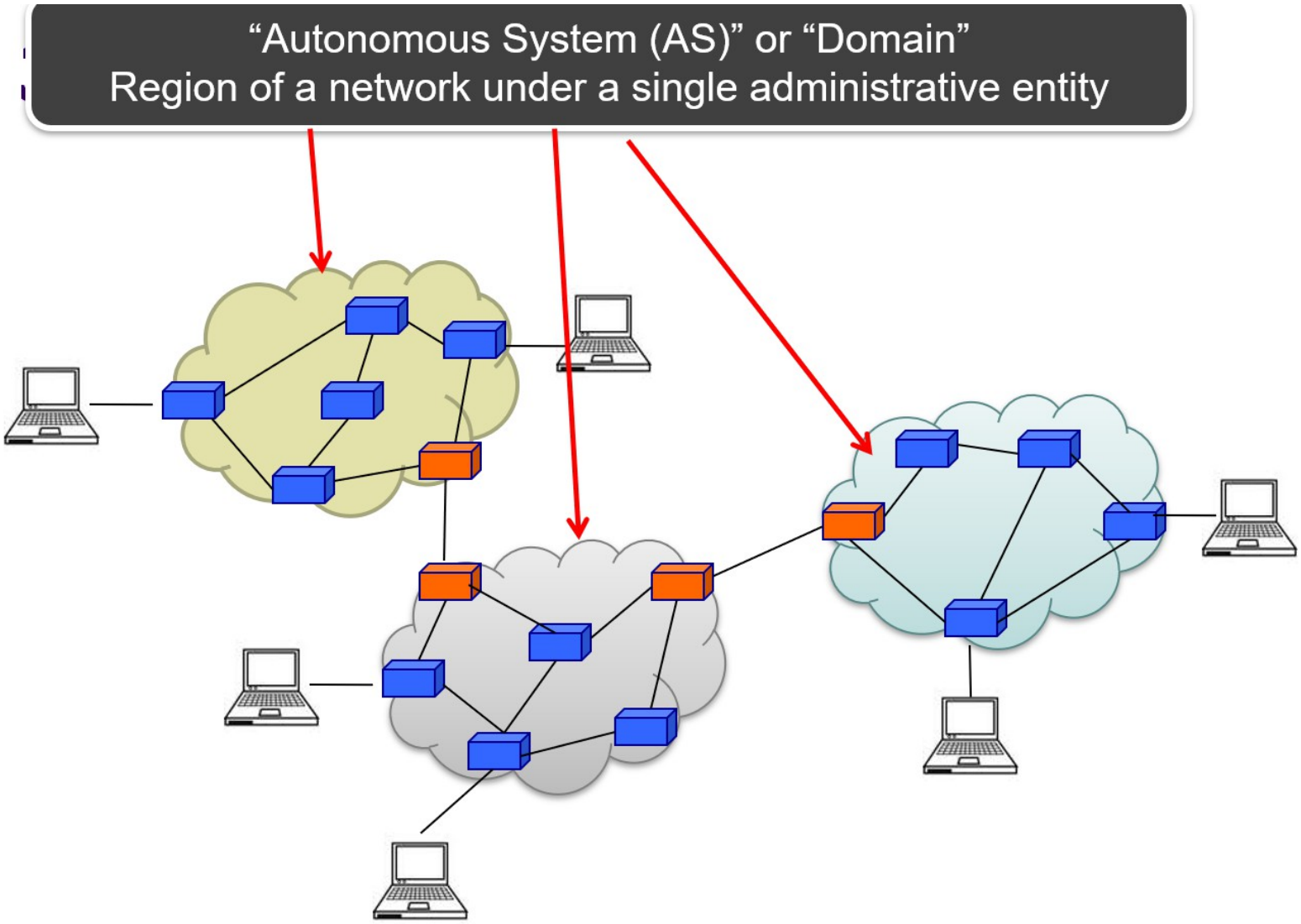
INTERNET: TOPOLOGIA SEMPLIFICATA



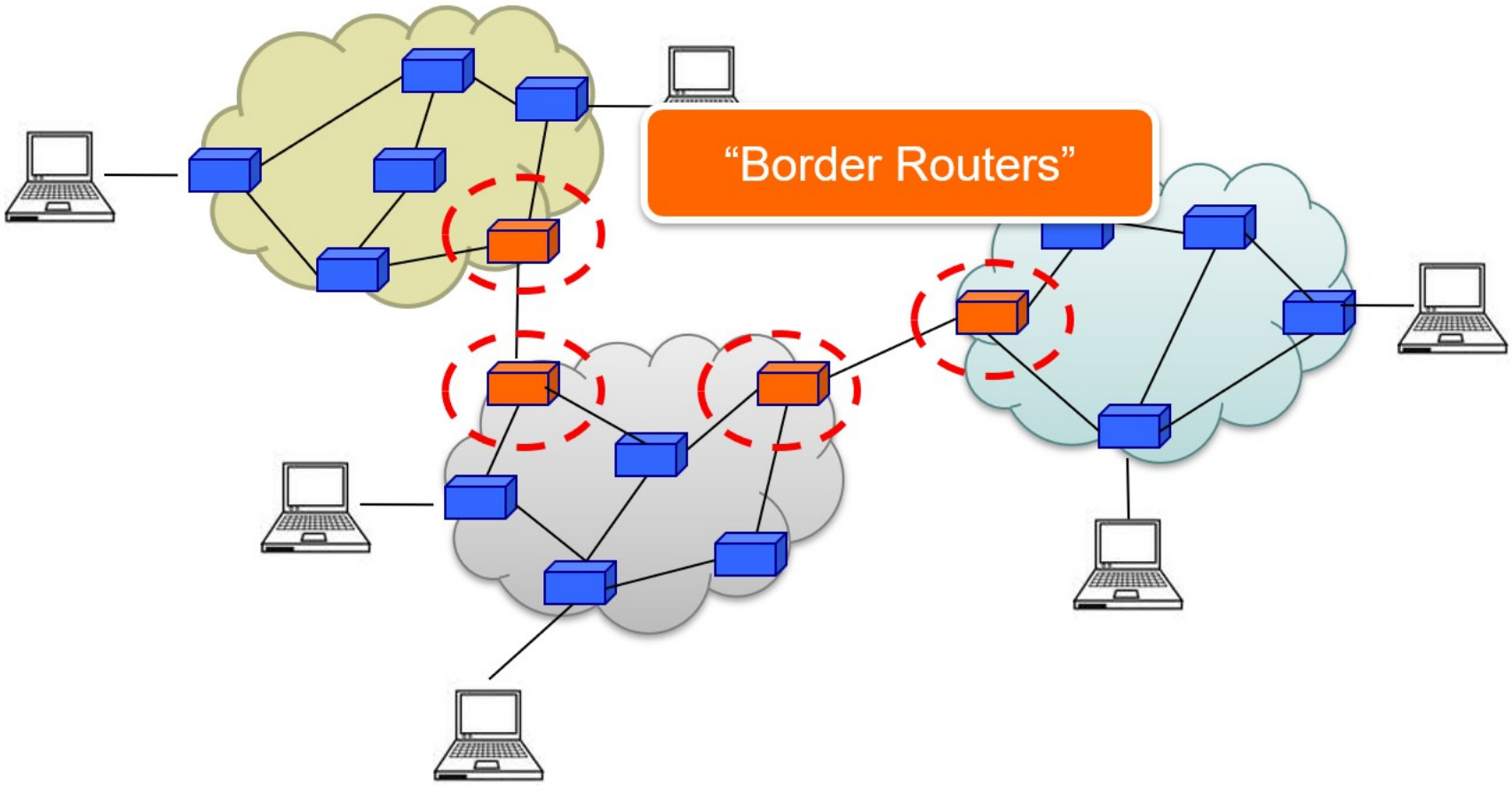
INTERNET: TOPOLOGIA SEMPLIFICATA



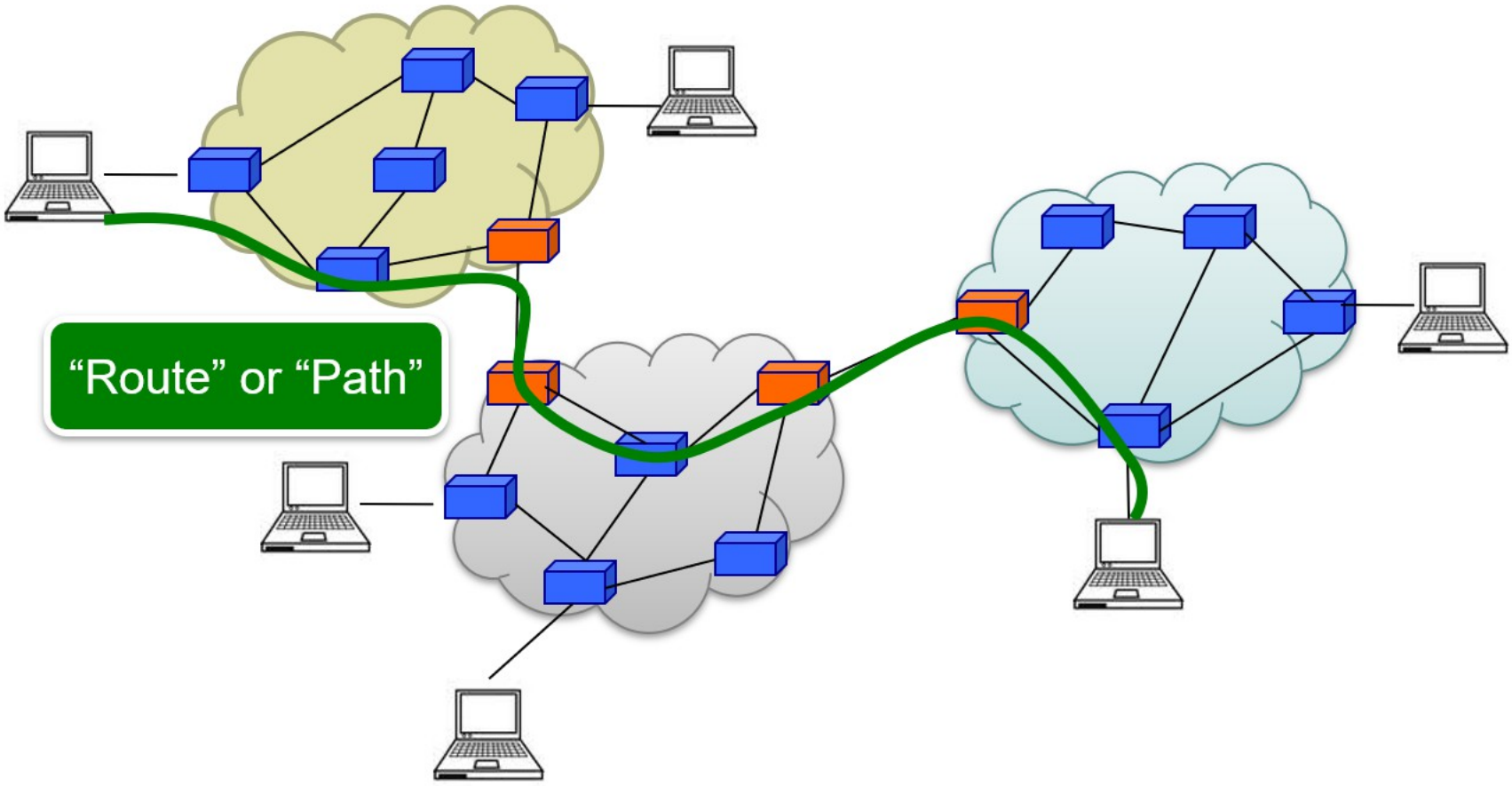
INTERNET: TOPOLOGIA SEMPLIFICATA



INTERNET: TOPOLOGIA SEMPLIFICATA



INTERNET: TOPOLOGIA SEMPLIFICATA



IL ROUTING

Il problema:

- definire un insieme di procedure che devono essere eseguite dagli switch/router di una rete packed-switched affinchè che un pacchetto inviato da un nodo sorgente raggiunga un nodo destinazione.
  - non garanzia assoluta della consegna, solo best effort
  - perdita di pacchetti è un evento previsto (overflow nelle code dei router, collisioni su linee condivise,...), possibile anche nel processo di routing
  - obiettivo: garantire un "ragionevole best effort"

- Concetti di base:
  - addressing
  - forwarding
  - Intradomanin distributed routing protocols
    - Link state protocols
    - Distance vectors
  - comportamento dei protocolli in presenza di churn/ fallimenti di elementi della rete

ROUTING: LE SFIDE

- Informazione distribuita:
  - ogni nodo conosce solo i propri vicini immediati.
  - la rete deve fornire connettività a livello globale, partendo da questa infomazione distribuita
- Efficienza:
  - I cammini individuati sevono essere "ragionevolmente buoni" altrimenti la latenza dei pacchetti può aumentare
  - sssociazione ad ogni link di costi, individuazione di un cammino che minimizzi il costo totale sui sui link
  - altro parametro da considerare: banda consumata per l'individuazione dei cammini migliori
- Fallimenti:
  - Link e nodi possono guastarsi arbitrariamente
  - soluzioni dinamiche e auto-adattabili

ADDRESSING

- per spedire pacchetti a un end point, occorre che sia identificabile rispetto agli altri

- ipotesi semplificativa
  - ogni end host o router è identificato in modo univoco
  - Semplificazione che consente una descrizione ad alto livello degli algoritmi ed dei protocolli di routing

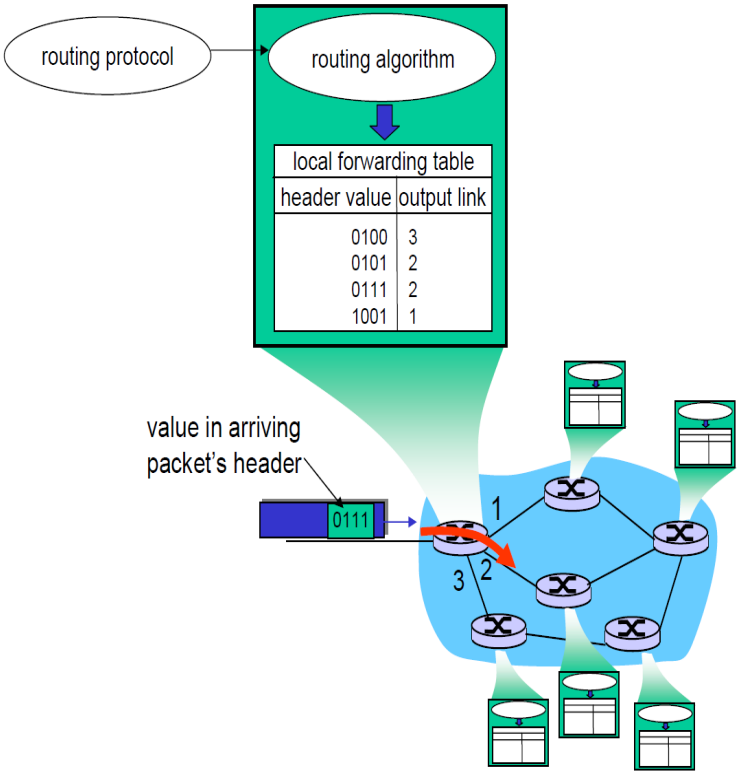
- in seguito:
  - struttura gerarchica degli indirizzi
  - distinzione tra l'identificatore (nome) di un nodo ed i suoi indirizzi.
    - più indirizzi per ogni nodo
    - indirizzo associato alla interfaccia di rete di un nodo,
  - nome unico del nodo diverso dagli indirizzi associate alle sue interfacce

FORWARDING

- In una packed-switched network, ogni pacchetto inviato contiene
  - l'indirizzo della destinazione
  - In generale anche l'indirizzo del sender, che permette alle applicazioni di rispedire eventualmente il pacchetto al mittente
- Forwarding:
  - quando un router riceve un pacchetto, consulta una tabella (tabella di routing) utilizzando con chiave l'indirizzo destinazione
  - determinare su quale link spedire il pacchetto per arrivare a destinazione.
- Route:
  - indirizzo destinazione+ outgoing link usato dal router
- Cammino: una sequenza di routes

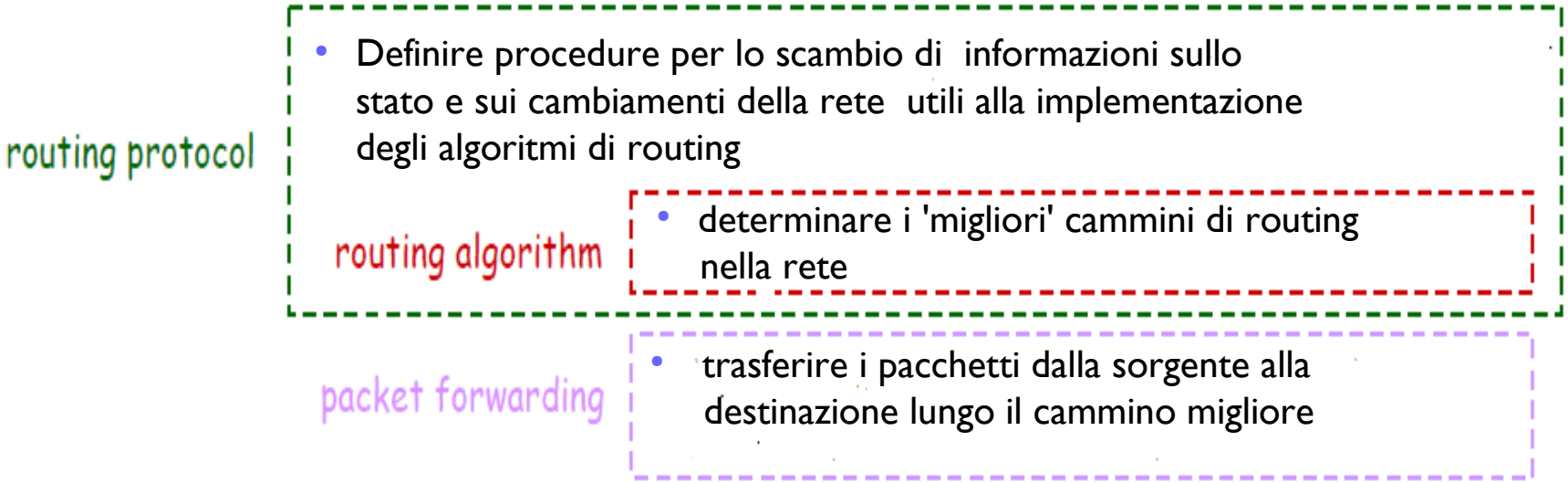
FORWARDING (ROUTING) TABLE

- Forwarding o routing table
  - interfaccia tra il processo di routing e quello di forwarding
- Hashmap:
  - chiave:
    - indirizzo destinazione di un host nella rete
  - valore:
    - porta di uscita del router
    - su quella porta il pacchetto individuato dalla chiave viene spedito verso la destinazione



RELAZIONE TRA COMPONENTI DEL ROUTING

- Routing: insieme di algoritmi e regole che permettono ai routers di:



obiettivi di un buon routing:

- accuratezza, rapidità, basso costo
- evitare link congestionati o falliti
- evitare routing loops
- adattabilità a traffico variabile: trovare cammini alternativi in base al traffico
- basso overhead: miniizzare l'overhead causato dai messaggi di controllo

ALGORITMI DI ROUTING INTRA DOMINIO

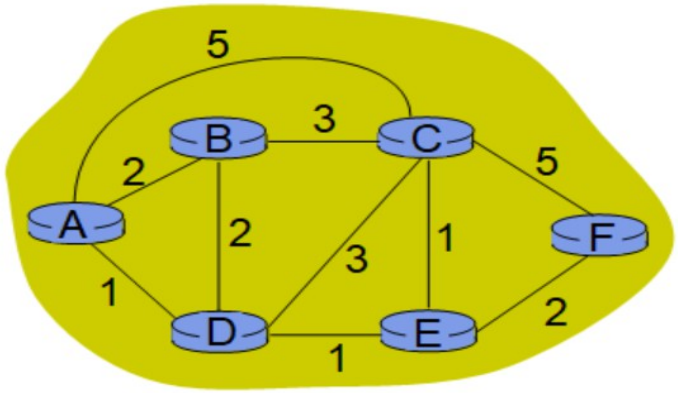
- algoritmi intra-dominio
  - eseguiti su una rete tipicamente gestita da un unico operatore o da una singola entità autonoma (autonomous system)
  - l'entità ha il controllo su tutti i dispositivi della rete

- poichè una singola entità controlla la rete e la gestisce, essa
  - può scegliere liberamente tra diversi cammini alternativi tra un nodo sorgente ed un nodo destinazione
  - tutti questi algoritmi cercano di minimizzare qualche metrica relativa al cammino tra due nodi

- due approcci principali
  - Link state
  - Distance Vectors

ALGORITMI DI ROUTING INTRA DOMINIO

- La rete viene modellata come un grafo
  - vertici: routers
  - archi: link di comunicazione
  - ogni arco ha associato un peso



- Un algoritmo di routing intra-dominio cerca di minimizzare qualche forma di "metrica di cammino", che a sua volta è la somma dei valori di alcune proprietà dei link che compongono il cammino

- Possibili scelte
  - considerare i propagation delay sui link del cammino e, di conseguenza, la latenza minima lungo un cammino
  - ma sono possibili anche altre scelte...
    - perdita di pacchetti
    - numero di hops di un cammino,.
    - ….

PROTOCOLLI DI ROUTING

- per la escuzione di un algoritmo di routing, ogni nodo ha bisogno di informazioni sulla topologia e lo stato della rete
- necessario scambio di informazione con i nodi vicini
- i vicini possono, a loro volta, veicolare informazione sui loro vicini

Routing protocol:

- supporta lo scambio di informazioni per l'esecuzione degli algoritmi di routing.
- le informazioni scambiate sono diverse a seconda dell'algoritmo di routing.

CLASSIFICAZIONE DEGLI ALGORITMI DI ROUTING

- routing area:
  - interno ad un autonomous system o esterno (intra domain o inter domain)
- statico/dinamico
  - a seconda della frequenza di aggiornamento delle informazioni relative al routing
- centralizzato/ globale/ distribuito
  - quali informazioni sono disponibili in un nodo per il calcolo del cammino migliore?
- source-based/hop-by-hop
  - hop-by-hop: il nodo memorizza solo il puntatore al prossimo nodo (hop) verso una certa destinazione
  - source based: il nodo memorizza l'intero cammino verso la destinazione

CLASSIFICAZIONE: ROUTING AREA

- Intra-domain routing
  - stabilisce i cammini all'interno di un singolo dominio amministrativo
  - due approcci (algoritmo/protocollo)
    - link State/Open Shortest Path First (OSPF)
    - distance Vector/Routing Information Protocol(RIP)

- Inter-domain routing
  - definisce i cammini tra domini diversi
    - path Vector/Border Gateway Protocol (BGP)

CLASSIFICAZIONE DEGLI ALGORITMI DI ROUTING

- Static Routing:
  - scambiano informazioni relative a prorietà della rete statiche o relativamente 'poco dinamiche'
    - topologia (numero di hops)
    - link propagation delay o capacity

- Dynamic Routing
  - Basati sulllo scambio del valore di proprietà dinamiche della rete
    - queueing delay
    - più complessi da gestire

CENTRALIZZATO/GLOBALE/DISTRIBUITO

- Cetralizzato:
  - esiste una unica entità nella rete che ha una visione globale della topologia della rete
  - costruisce i cammini tra host e li restituisce on demand
  - soluzione poco adottata
- Globale:
  - ogni router si costruisce una propria visione globale della rete
  - sulla base della sua visione globale della rete calcola i cammini minimi tra se stesso ed ogni altro host ed, in base a questi, riempe la routing table
  - le diverse visioni globali vanno mantenute consistenti
- Distribuito:
  - ogni router possiede sol una visione locale dei nodi vicini
  - i router eseguono un algoritmo distribuito per la individuazione dei cammini migliori

LINK STATE ROUTING

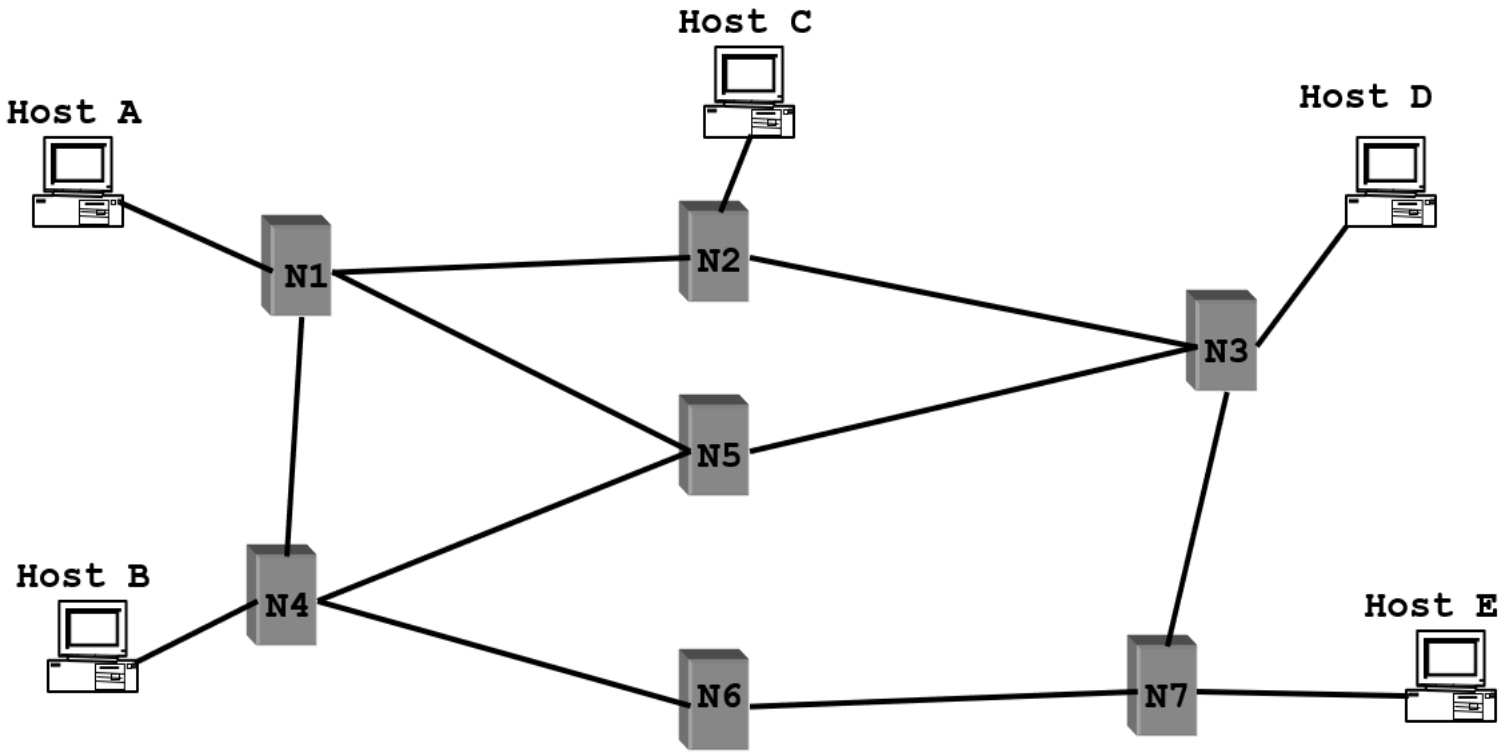
- condizione iniziale:
  - ogni router monitora e mantiene aggiornato dei suoi link incidenti
- passo 1:
  - ogni router invia in broadcast lo stato dei propri link
  - Scopo: fornire ad ogni router della rete una vista dell'intero grafo che rappresenta la topologia della rete
  - necessario meccanismo di flooding affidabile
- passo 2
  - ogni router calcola i cammini minimi tra esso stesso e tutti gli altri nodi della rete
  - tutti utilizzano lo stesso algoritmo per calcolare i cammini minimi
- Problema principale: scalabilità

LINK STATE ROUTING

- condizione iniziale:
  - ogni router monitora e mantiene aggiornato dei suoi link incidenti
- implementata mediante un semplice HELLO PROTOCOL
  - ogni nodo invia un HELLO PACKET lungo tutti I suoi link, periodicamente
  - se si riceve il pacchetto dal vicino si presume che sia vivo e che il link non sia guasto
  - possibili perdite di pacchetti: la assenza di uno (o di un piccolo numero) di pacchetti può non essere significativa
- frequenza di invio dei pacchetti
  - Invio di pacchetti con regolarità in modo che il numero atteso dei pacchetti in un intervallo di tempo sia approssimativamente sempre uguale
  - HELLO INTERVAL: tempo che intercorre tra l'invio di due pacchetti consecutivi
    - scelto nell'intervallo [HELLO INTERVAL - δ, HELLO INTERVAL +δ], con $\delta <$ HELLO INTERVAL.

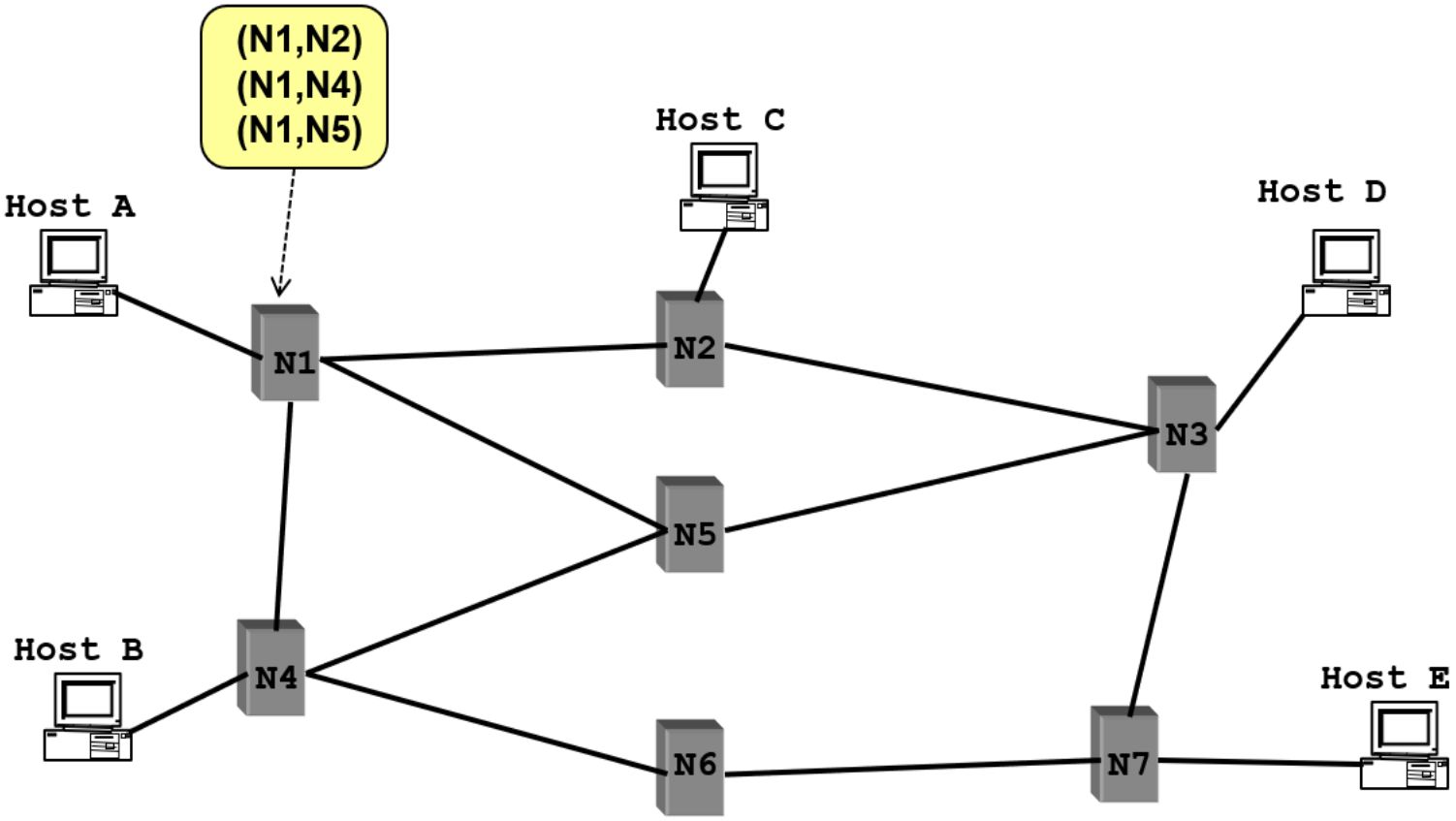
PASSO 1: FLOODING

ogni router invia in broadcast lo stato dei propri link, con lo scopo di fornire ad ogni router della rete una vista dell'intero grafo che rappresenta la topologia della rete



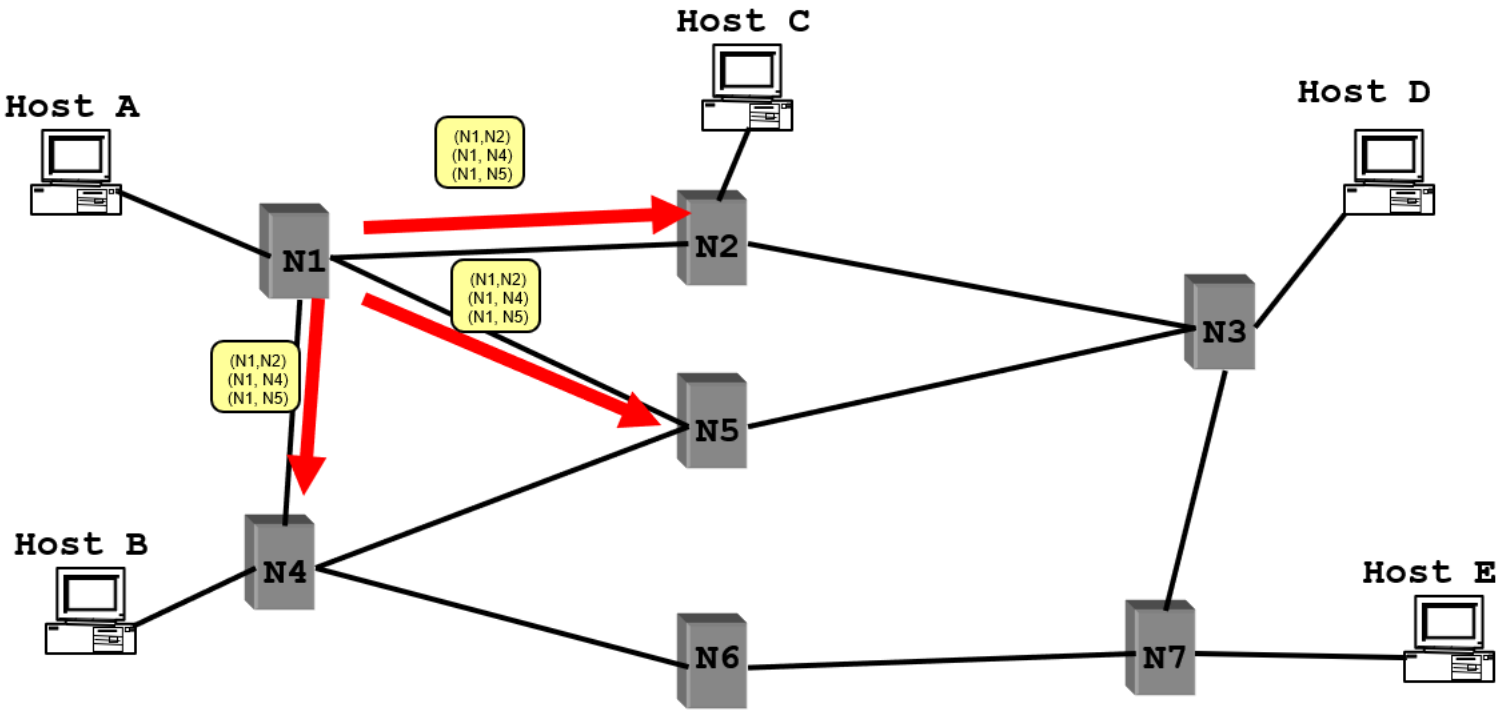
PASSO 1: FLOODING

- ogni router mantien il suo "link state" (LS)
  - una lista di tutti i link diretti verso i vicini e del loro costo



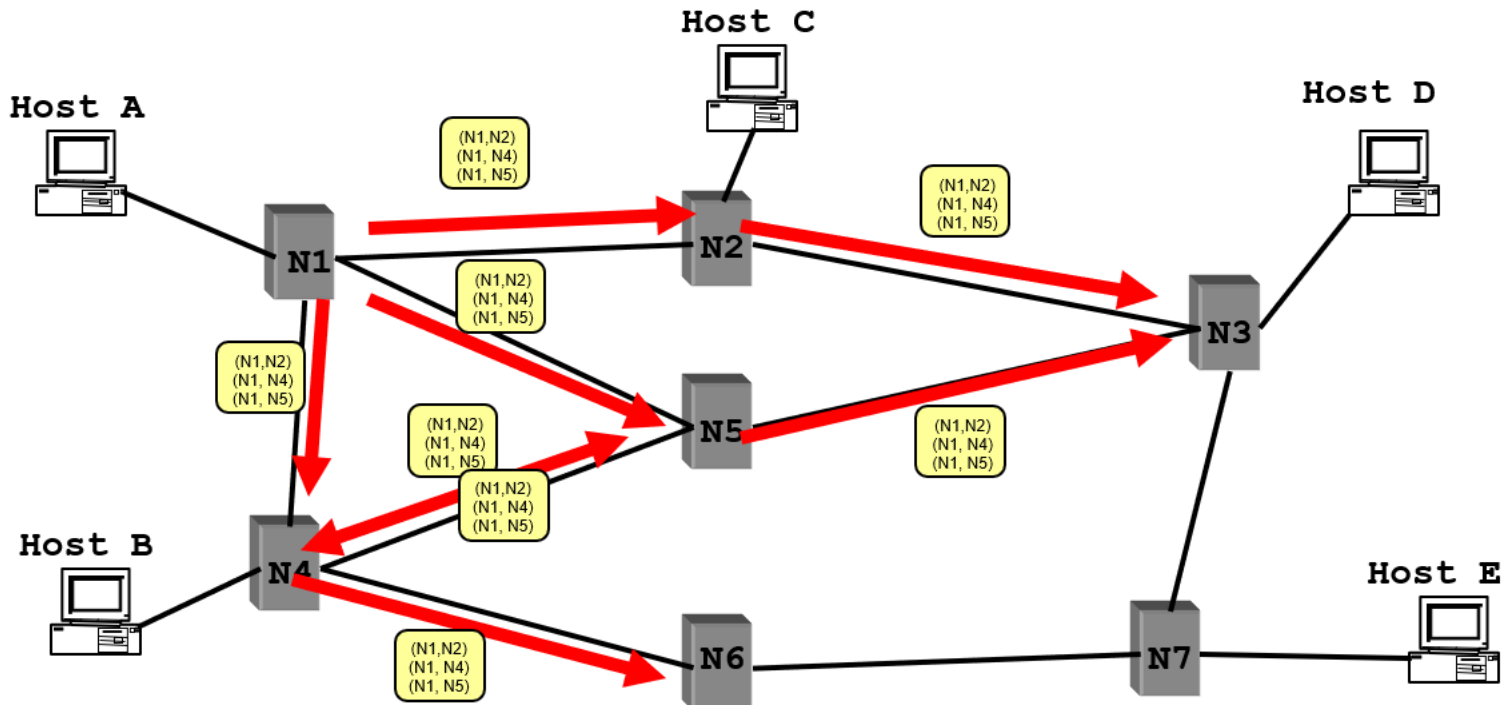
PASSO 1: FLOODING

- ogni router mantiene il suo "link state" (LS)
  - una lista di tutti i link diretti verso i vicini e del loro costo
  - invia in broadcast lo stato dei propri link, con lo scopo di fornire ad ogni router della rete una vista dell'intero grafo che rappresenta la topologia della rete



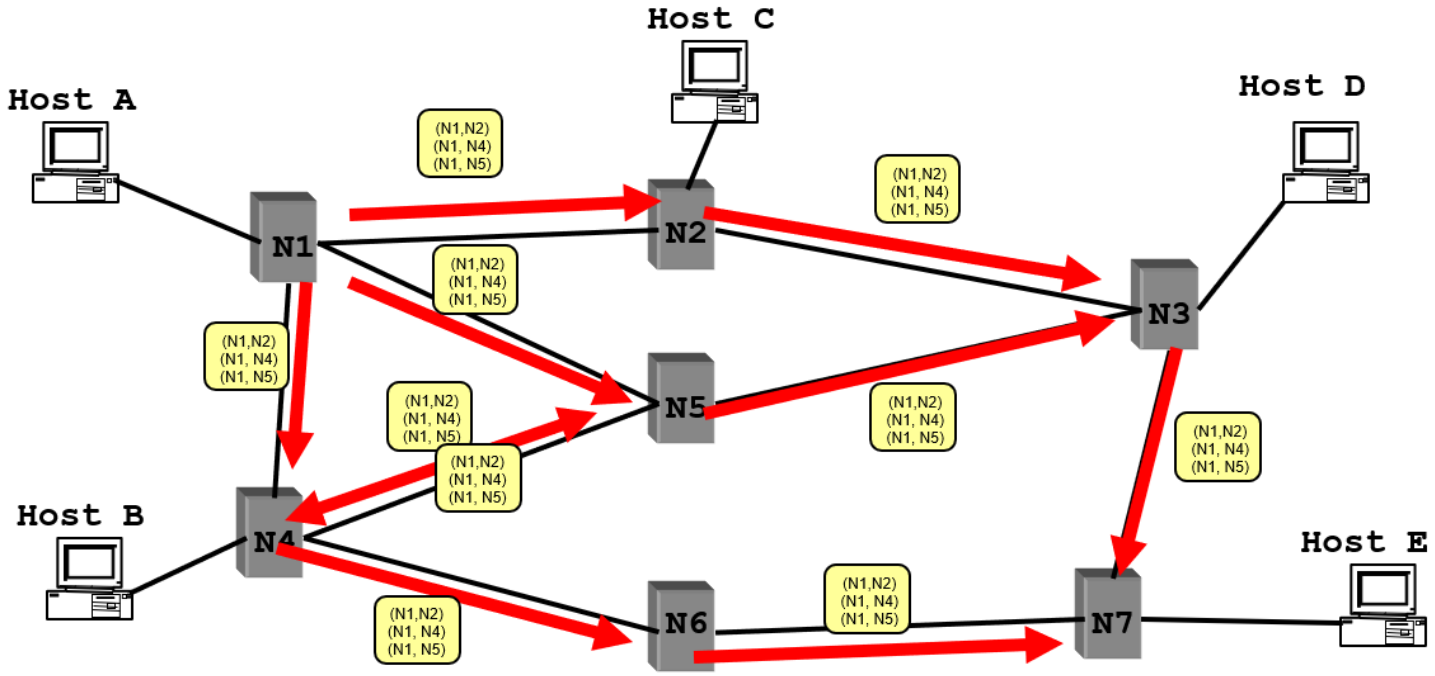
PASSO 1: FLOODING

- ogni router mantiene il suo "link state" (LS)
  - una lista di tutti i link diretti verso i vicini e del loro costo
  - invia in broadcast lo stato dei propri link, con lo scopo di fornire ad ogni router della rete una vista dell'intero grafo che rappresenta la topologia della rete
    - alla ricezione di un nuovo messaggio LS, un router lo propaga a sua volta a tutti i suoi vicini, a parte quello da cui ha ricevuto il messaggio.



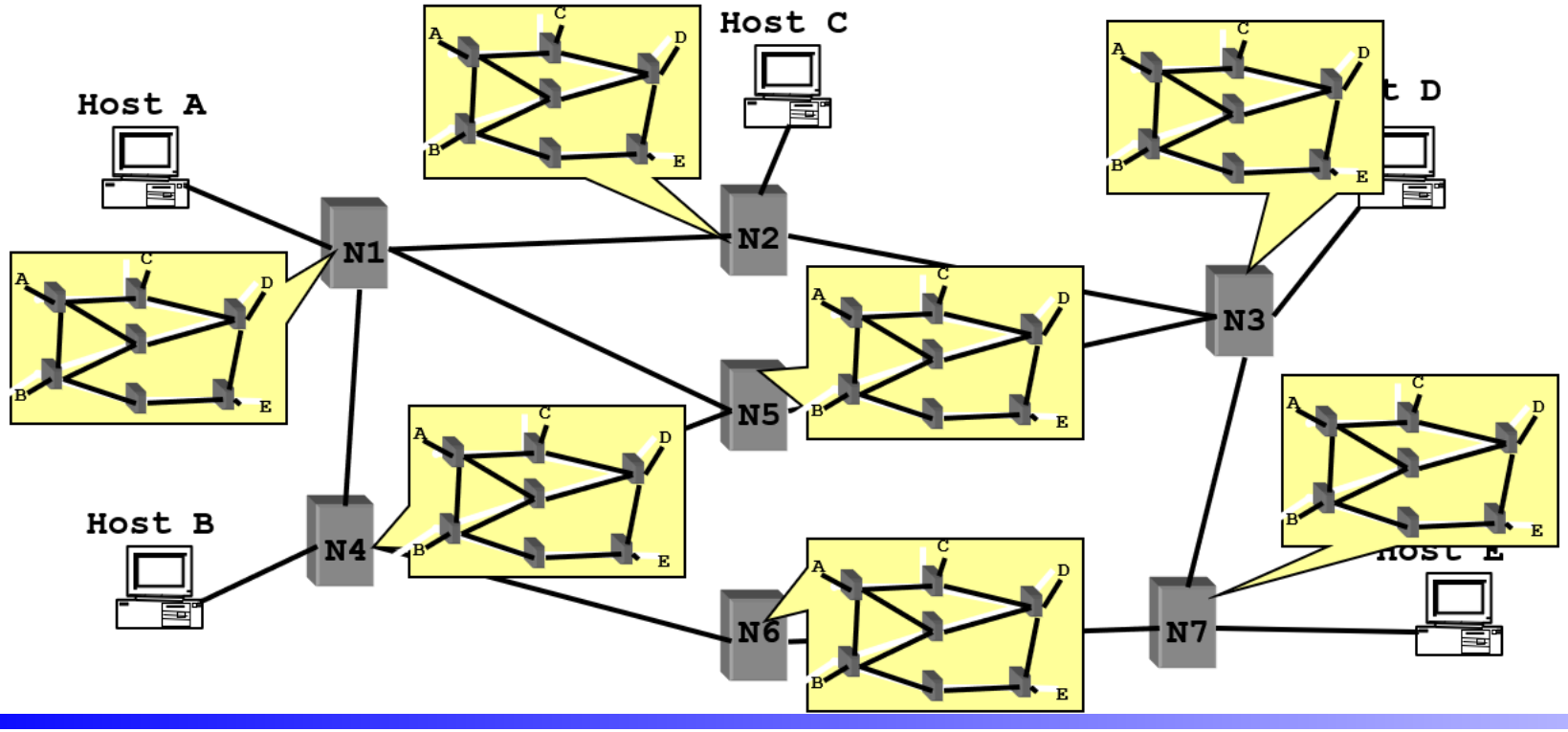
PASSO 1: FLOODING

- ogni router mantiene il suo "link state" (LS)
  - una lista di tutti i link diretti verso i vicini e del loro costo
  - invia in broadcast lo stato dei propri link, con lo scopo di fornire ad ogni router della rete una vista dell'intero grafo che rappresenta la topologia della rete
    - alla ricezione di un nuovo messaggio LS, un router lo propaga a sua volta a tutti i suoi vicini, a parte quello da cui ha ricevuto il messaggio.



PASSO 1: FLOODING

- ogni router mantiene il suo "link state" (LS)
  - una lista di tutti i link diretti verso i vicini e del loro costo
  - invia in broadcast lo stato dei propri link, con lo scopo di fornire ad ogni router della rete una vista dell'intero grafo che rappresenta la topologia
- della rete
  - alla fine, tutti i router possono costruire un grafo della intera rete



CALCOLO DEI CAMMINI MINIMI

- dopo che i router hanno ricostruito lo stato globale dell'intera rete,
  - possono applicare al grafo risultante un qualsiasi algoritmo per il calcolo dei cammini minimi
  - SSSP: single source shortest path, interessa trovare i cammini minimi dal router a tutti gli altri router della rete.

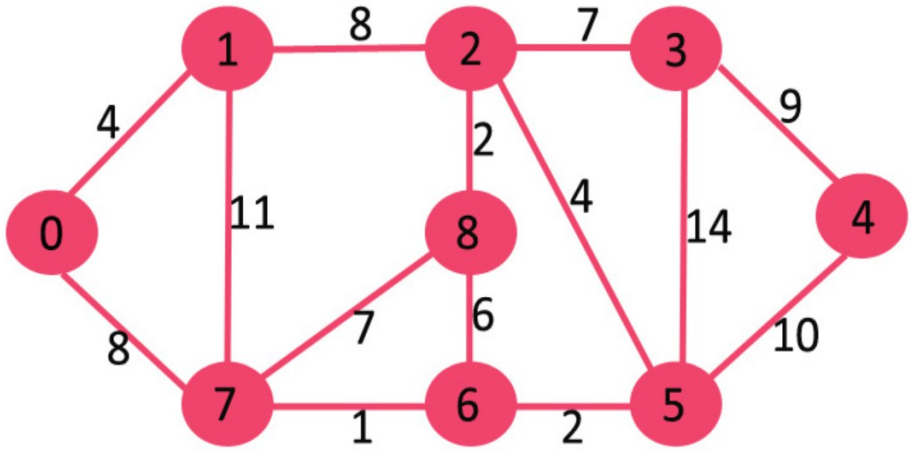
- Algoritmi SSSP, Shortest Path algorithms
  - Dijkstra: pesi sugli archi strettamente positivi
    - solo versione centralizzata

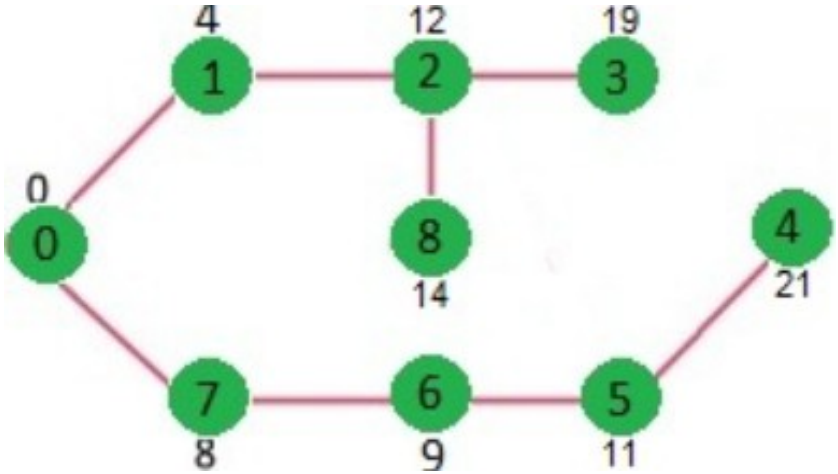
  - Bellman-Ford: può gestire anche pesi negativi.
    - versione centralizzata e distribuita

- Link state utilizza l'algoritmo di Dijkstra
  - Dijkstra's shortest path tree (SPT)

L'ALGORITMO DI DIJKSTA: PROPRIETA'

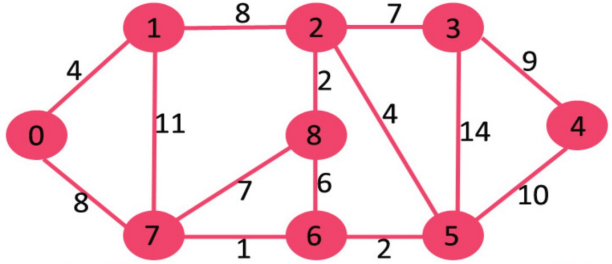
- calcola uno shortest path first tree
- i cammini minimi verso nodi distanti dalla radice sono branch di cammini minimi verso nodi più vicini all radice
- soluzione finale appare come un albero di cammini minimi dalla sorgente a tutti gli altri nodi

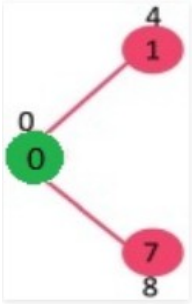


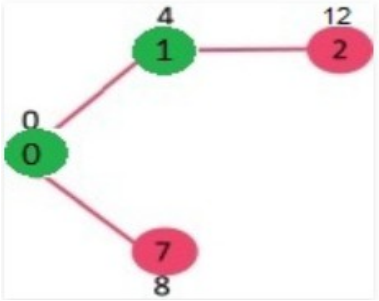


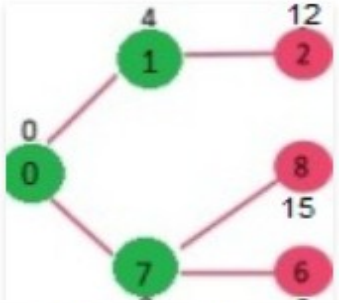
L'ALGORITMO DI DIJKSTA: PROPRIETA'

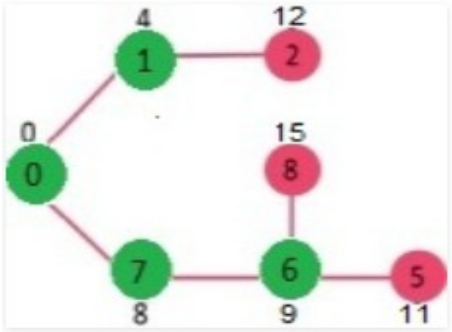
- L'algoritmo lavora espandendo incrementalmente l'albero
- Espansione basata sulla segunte proprietà dei cammini minimi:
  - se il cammino minimo dal nodo X al nodo Y passa dal nodo Z, allora anche il sottocammino da X a Z deve essere un cammino minimo

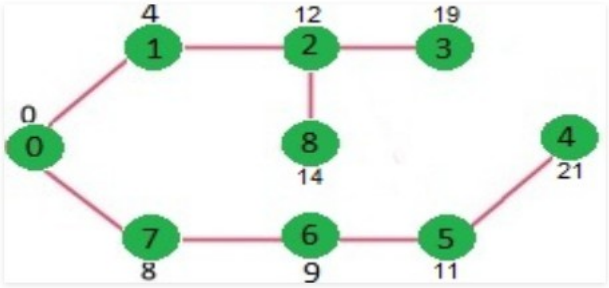












L'ALGORITMO DI DIJKSTA

Un algoritmo di routing a conoscenza globale:

- INPUT:
  - il grafo che rappresenta la topologia della rete, con i costi dei link
  - la sorgente deve avere una conoscenza globale della rete per determinare la sua routing table
- OUTPUT
  - cammini di costo minimo da un nodo N a tutti gli altri
  - produce un "albero" di cammini
  - l'albero ha radice in N
- Basato su una processo induttivo:
  - supponiamo che alla iterazione k, abbia trovato i k nodi più vicini
  - con un altro passo, posso individuare il k+1-esimo vicino, a partire dai k del passo precedente

L'ALGORITMO DI DIJKSTA

```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞;
7
```

c(i,j): costo del link tra i e j

D(v): costo attuale del cammino dal nodo sorgente a v

p(v): il predecessore di v lungo il cammino tra la sorgente e v

S: insieme di nodi per cui è stato calcolato definitivamente il cammino minimo

L'ALGORITMO DI DIJKSTA

```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞;
7
8  Loop
9    find w not in S such that D(w) is a minimum;
10   add w to S;
```

c(i,j): costo del link tra i e j

D(v): costo attuale del cammino dal nodo sorgente a v

p(v): il predecessore di v lungo il cammino tra la sorgente e v

S: insieme di nodi per cui è stato calcolato definitivamente il cammino minimo

L'ALGORITMO DI DIJKSTA

```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞;
7
8  Loop
9    find w not in S such that D(w) is a minimum;
10   add w to S;
11   update D(v) for all v adjacent to w and not in S:
12     if  D(w) + c(w,v) < D(v) then
         // w gives us a shorter path to v than we've found so far
13       D(v) = D(w) + c(w,v); p(v) = w;
```

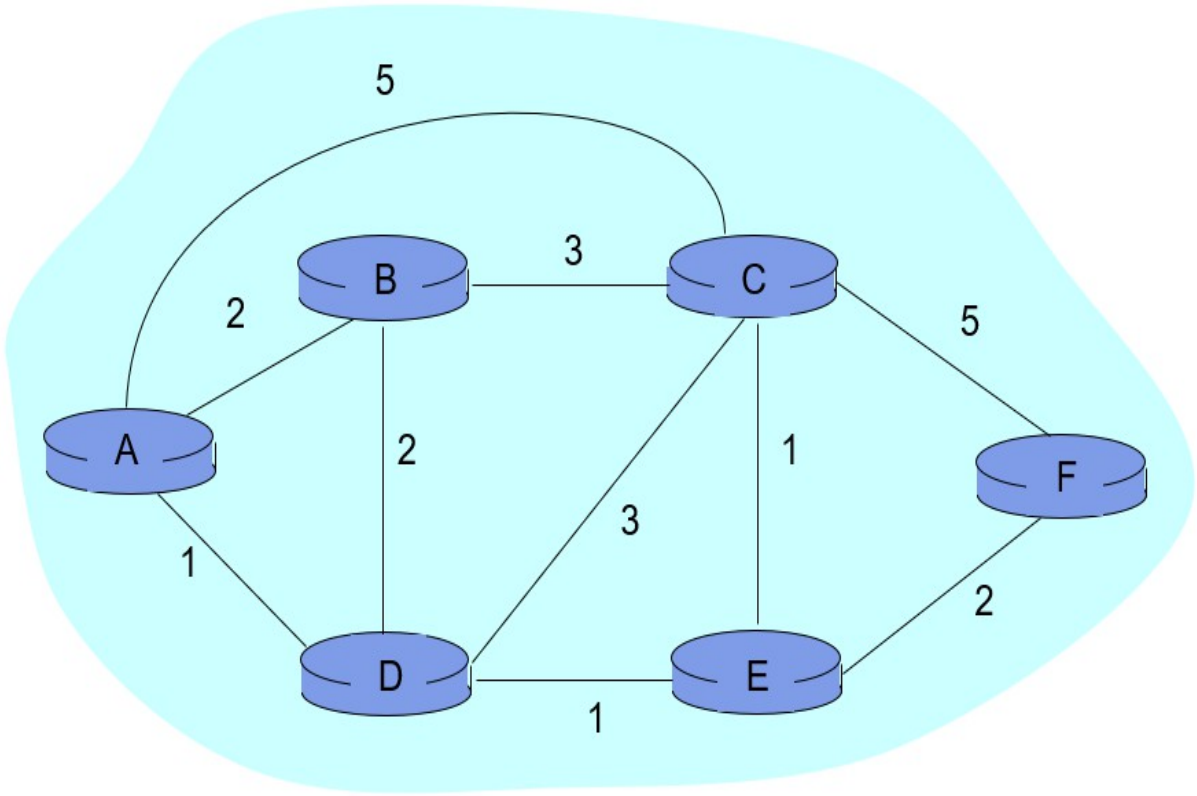
c(i,j): costo del link tra i e j

D(v): costo attuale del cammino dal nodo sorgente a v

p(v): il predecessore di v lungo il cammino tra la sorgente e v

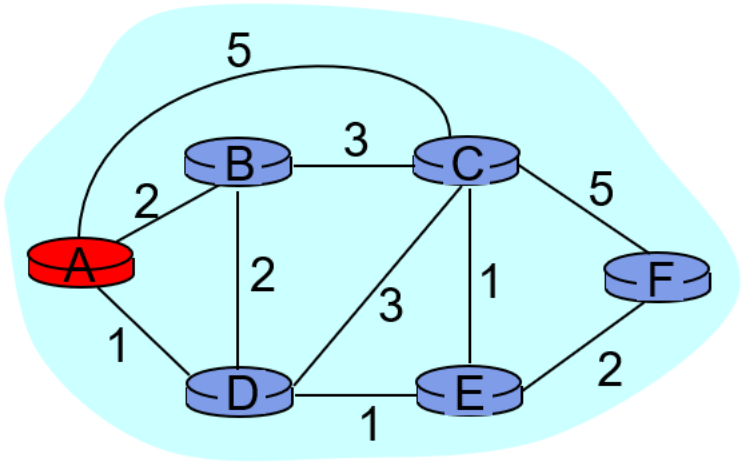
S: insieme di nodi per cui è stato calcolato definitivamente il cammino minimo

ALGORITMO DI DIJSTRA: UN ESEMPIO



ALGORITMO DI DIJSTRA: UN ESEMPIO

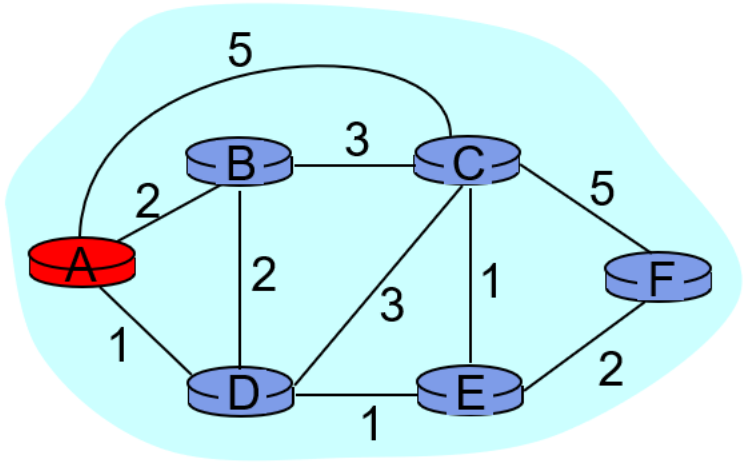
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞;
…
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

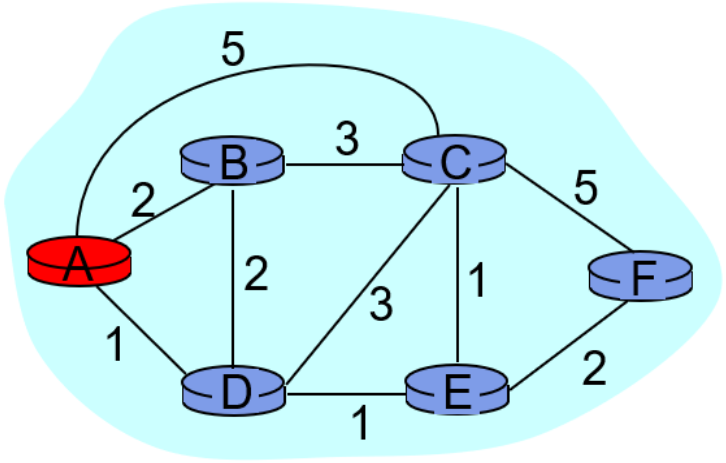
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

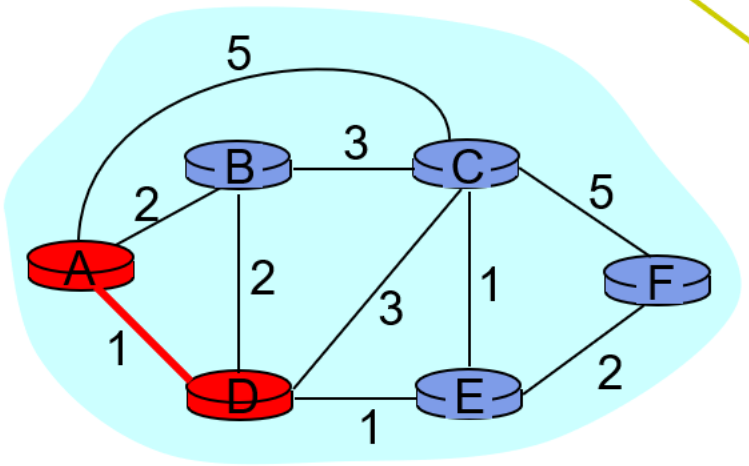
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

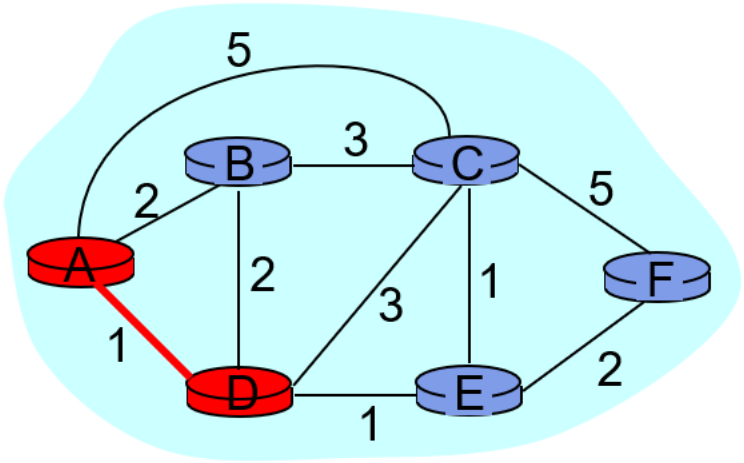
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

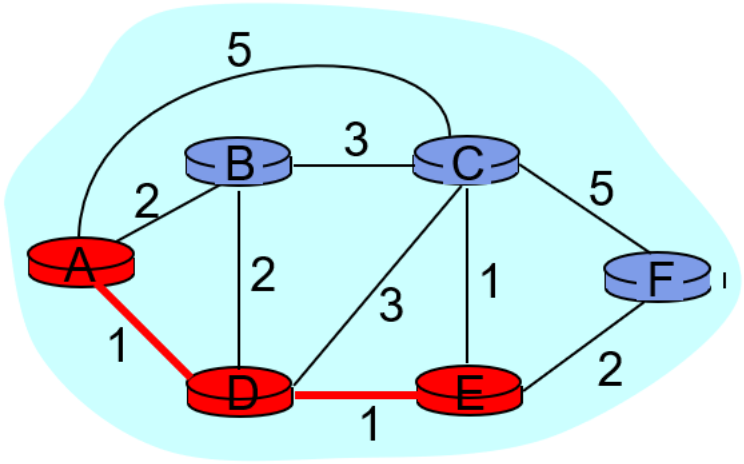
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2						
3						
4						
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14  until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

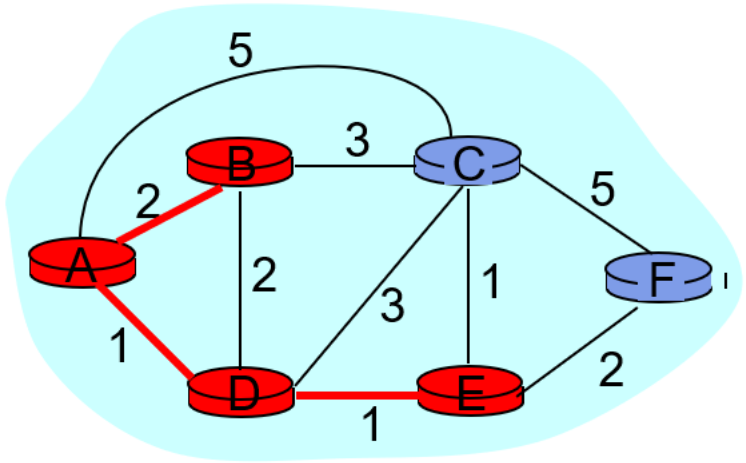
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3						
4						
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimur
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

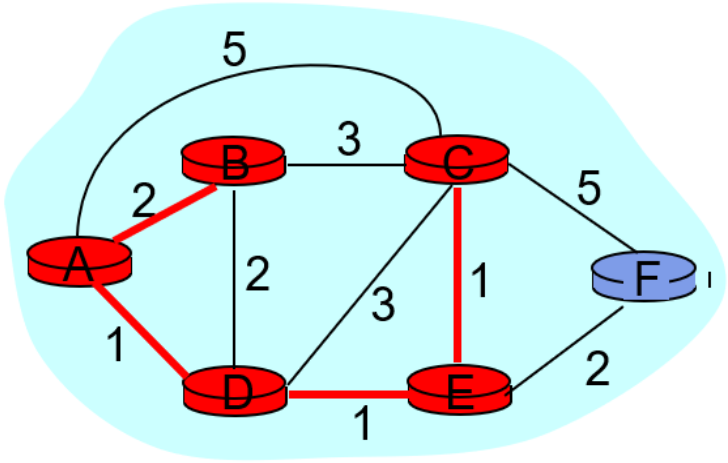
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4						
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

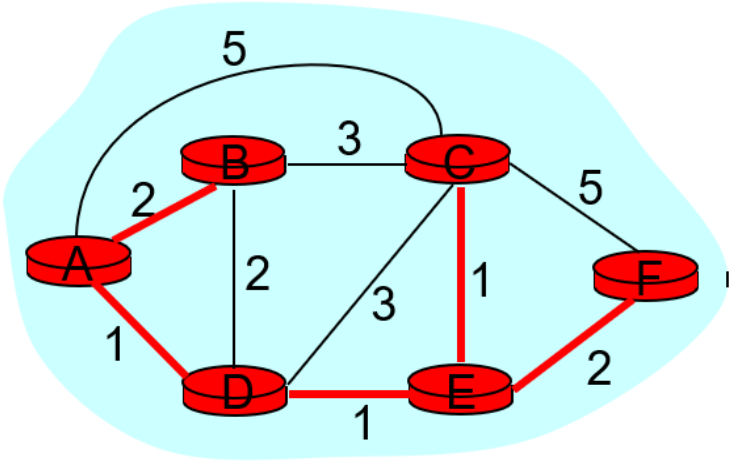
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

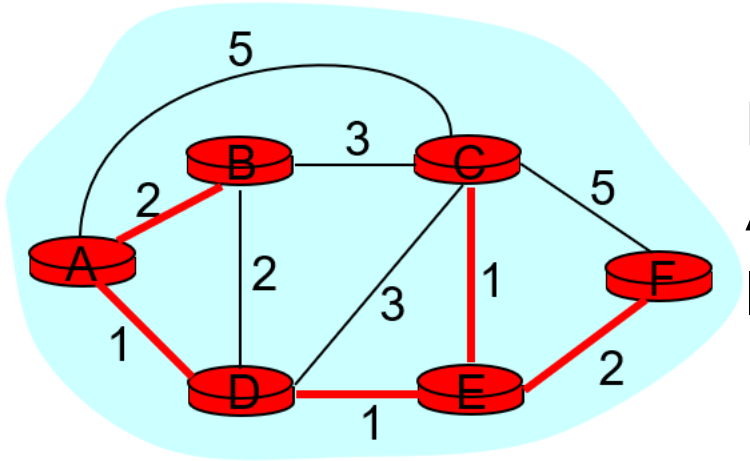
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in S;
```

ALGORITMO DI DIJSTRA: UN ESEMPIO

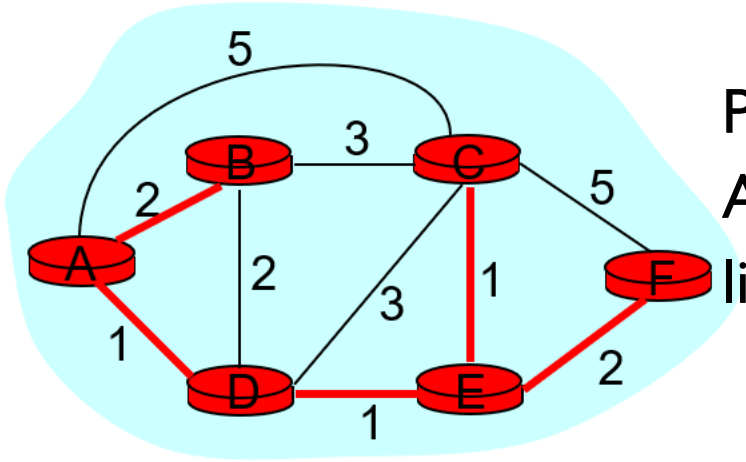
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



Per determinare il cammino minimo da A a C (ad esempio) percorrere a ritroso link, partendo da C, mediante p(v)

ALGORITMO DI DIJSTRA: UN ESEMPIO

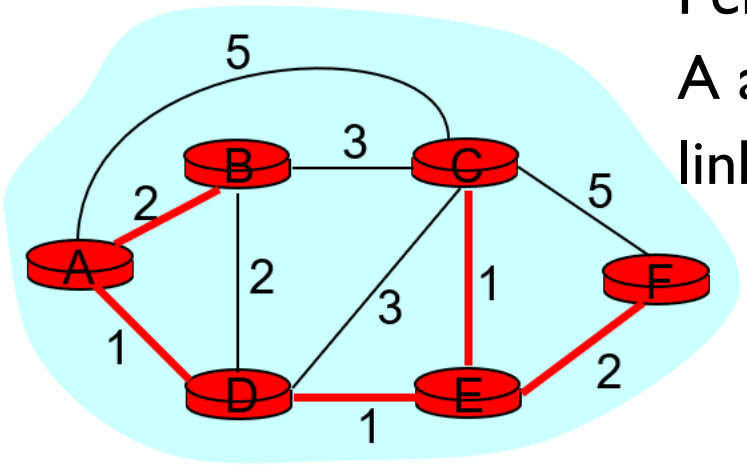
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



Per determinare il cammino minimo da A a C (ad esempio) percorrere a ritroso i link, partendo da C, mediante p(v)

ALGORITMO DI DIJSTRA: UN ESEMPIO

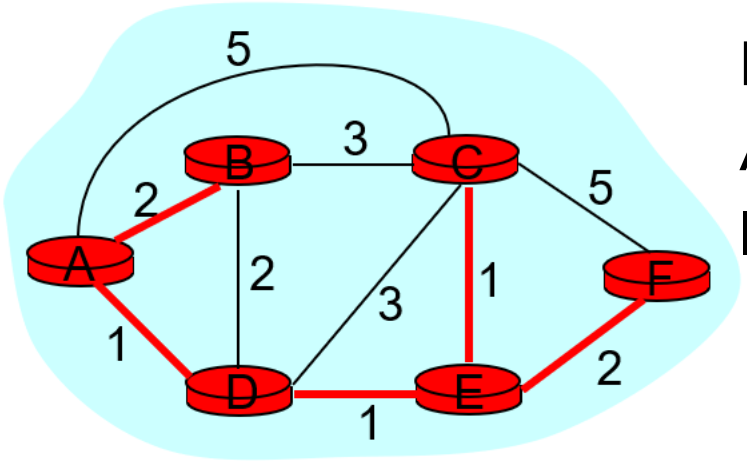
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



Per determinare il cammino minimo da A a C (ad esempio) percorrere a ritroso i link, partendo da C, mediante p(v)

ALGORITMO DI DIJSTRA: UN ESEMPIO

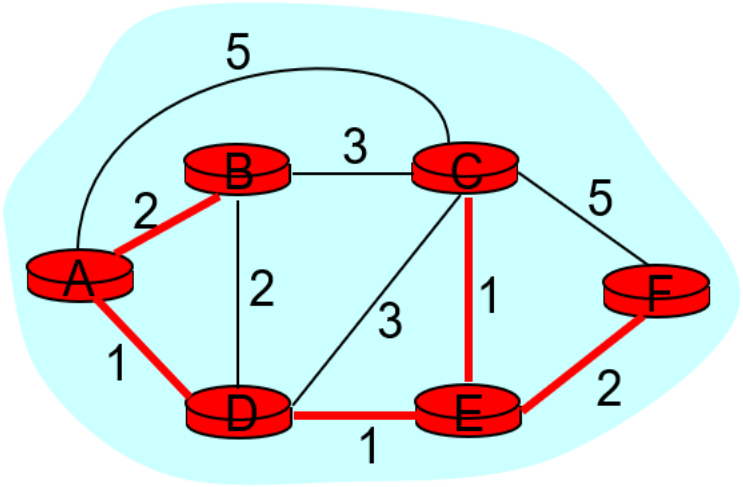
Step	set S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					



Per determinare il cammino minimo da A a C (ad esempio) percorrere a ritroso link, partendo da C, mediante p(v)

LINK STATE ROUTING: TABELLE DI ROUTING

- l'esecuzione dell'algoritmo di Dijkstra sul nodo A restituisce lo shortest path da A a tutte le destinazioni
- in seguito alla esecuzione dell'algoritmo, A costruisce quindi la seguente forwarding table, che indica, per ogni destinazione, il link di uscita su cui deve essere inoltrato un messaggio diretto verso quella destinazione.



Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

PROBLEMI: SCALABILITA'

- Quanti messaggi sono necessari per la propagazione dei messaggi "link state"?

  O(N x E), dove N è #nodi; E è #archi nel grafo

- Complessità di elaborazione per l'algoritmo di Dijkstra?

  $O(N^2)$,perchè si devono controllare tutti I nodi w non appartenenti ad S, ad ogni iterazione, e si hanno O(N) iterazioni

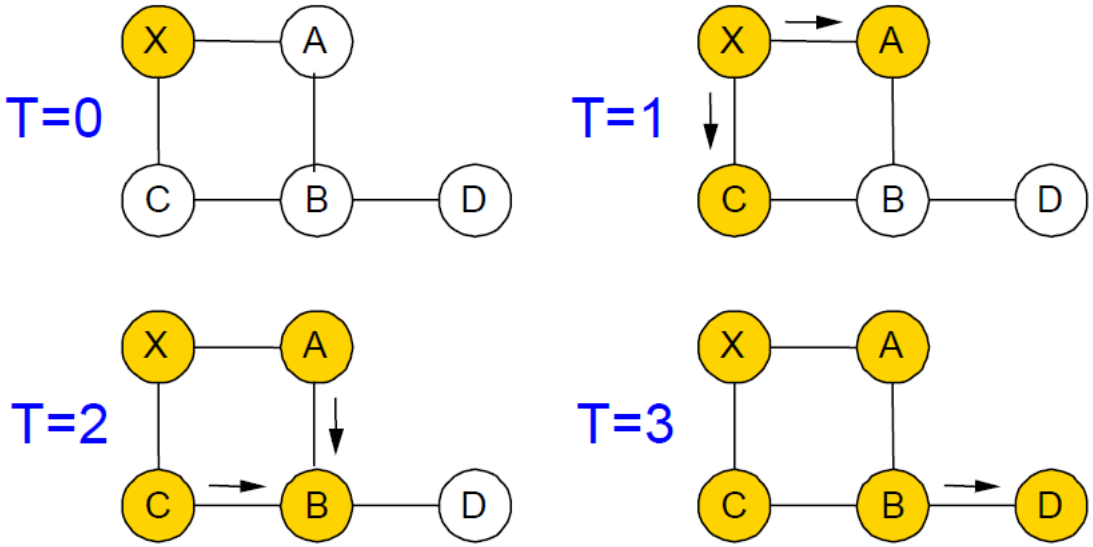
  Implementazioni più efficienti: **O(N log(N))**

- Ordine di complessità per il database della topologia? **O(E)**

- Ordine di complessità per la forwarding table? **O(N)**

PROBLEMI: IMPLEMENTAZIONE DEL FLOODING

- ogni router invia informazione su tutti i suoi link di uscita
- anche il prossimo router invia informazioni su tutte le porte di uscita
  - eccetto che al router da cui ha ricevuto l'informazione
  - necessità di ricordare i messaggi precedenti, e sopprimere i duplicati!



PROBLEMI: IMPLEMENTAZIONE DEL FLOODING

- Flooding affidabile:
  - assicurare che tutti i nodi ricevano le notifiche di link-state
  - assicurare che tutti i nodi usino gli aggiornamenti più recenti
- Difficoltà
  - perdita di pacchetti
  - arrivi fuori ordine
- Soluzioni
  - acknowledgments e ritrasmisssioni
  - numeri di sequenza

QUANDO INIZIARE IL FLOODING?

- Cambio nella topologia
  - fallimento di link oppure di nodi
  - recovery di link oppure di nodi
- Cambi di configurazione
  - cambi nel costo dei link
  - costi possono essere dinamici: frequenti i cambi
- Periodicamente
  - refresh delle notifiche link-state
  - ogni (ad esempio) 30 minuti, correzioni di possibili dati corrotti

CONVERGENZA

- Convergenza si ha quando l'informazione è consistente su tutti i nodi
  - tutti i nodi hanno lo stesso link-state database
- Forwarding consistente dopo la convergenza
  - tutti i nodi hanno lo stesso link state database
  - tutti i nodi inoltrano i pacchetti lungo gli stessi cammini

ROUTING E LINK FAILURE

- Stato della rete cambia dinamicamente a causa di churn e link failure
- È impossibile garantire che in ogni istante lo stato della rete sia consistente
  - tempo richiesto per la propagazione degli aggiornamenti
  - tempo richiesto per raggiungere un consenso tra I diversi nodi
- eventual convergence: modello ideale
  - dato uno stato inziale arbitrario della rete e lo stato delle tabelle di routing all'istante di tempo $t = 0$, si consideri una sequenza di fallimenti e recovey che avvengono in un intervallo di tempo τ.
  - si supponga inoltre che dopo $t = \tau$, nessun cambiamento avvenga nella rete
  - eventual convergence
    - se il protocollo di routing assicura con alta probabilità che ogni nodo della rete abbia tabelle di rotuing corrette dopo un intervallo di tempo seguente a $t = \tau$

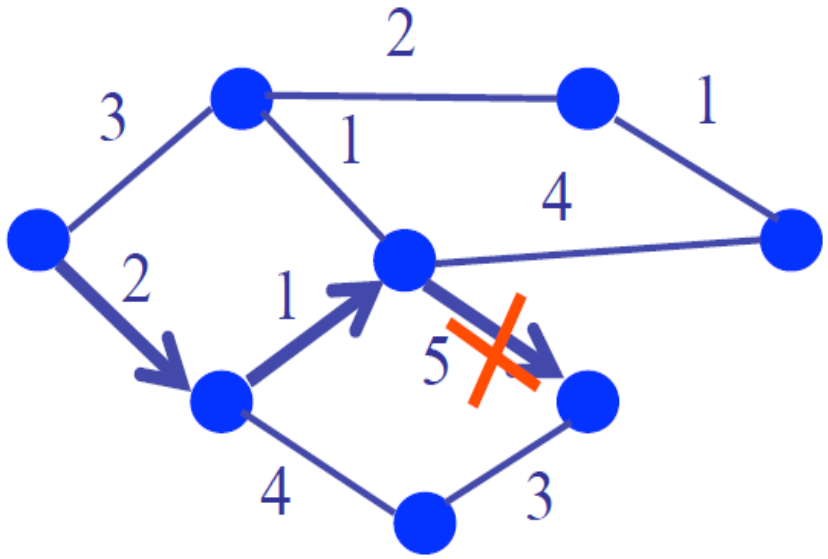
TEMPO DI CONVERGENZA

- tempo trascorso prima che ogni router abbia una visione consistente della rete
- possibili cause di ritardo
  - flooding delle notifiche link-state
  - ricalcolo delle tabelle di routing
  - memorizzazione delle tabelle di routing
- Prestazioni durante il periodo in cui la rete non ha raggiunto la convergenza:
  - parchetti persi dovuti a blackholes e TTL
  - pacchetti in loop che consumano risorse
  - pacchetti fuori ordine che raggiungono la destinazione
- Transitorio verso la convergenza: molto critico per applicazioni VoIP, online gaming, and video

BLACK HOLES

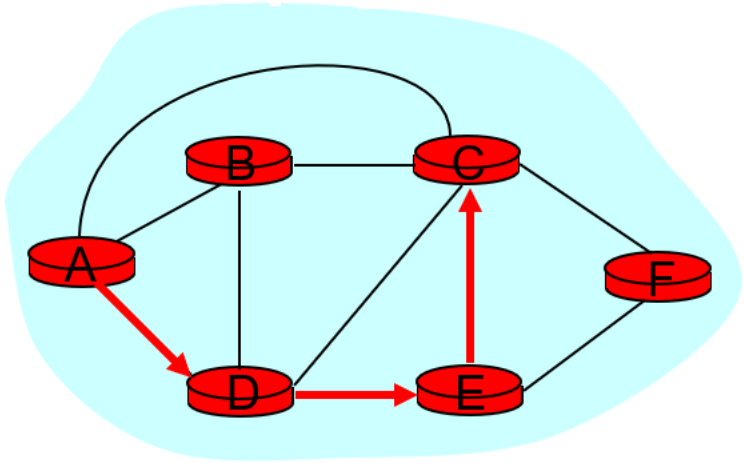
Ritrado nel riconoscimento del fallimento di un link

- en nodo non individua immediatamente il fallimento di un link
- ….ed inoltrai pacchetti in un black hole
- orobabilità dipende dal tempo richiesto per individuare il fallimento di un link
- perdita di pacchetti nel protocollo di routing

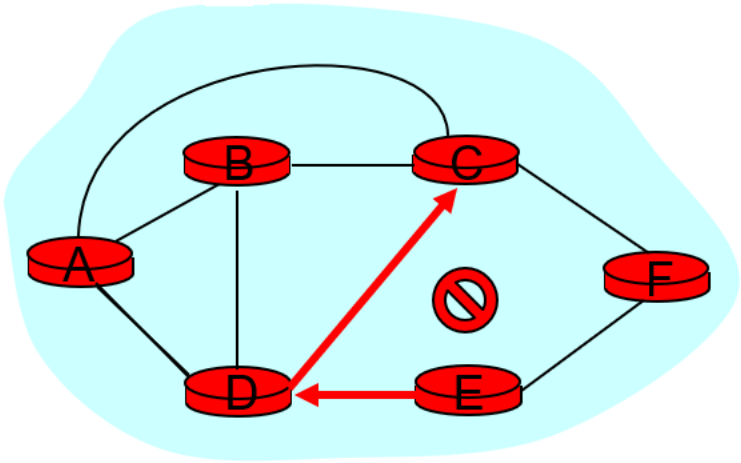


INCONSISTENZE DURANTE CONVERGENZA

- I database contenenti lo stato dei link possono essere inconsistenti
  - alcuni routers vengono a conoscenza di un guasto su un link, prima di altri
  - i cammini minimi non risultano più consistenti
  - questo può causare dei forwarding loops transitori



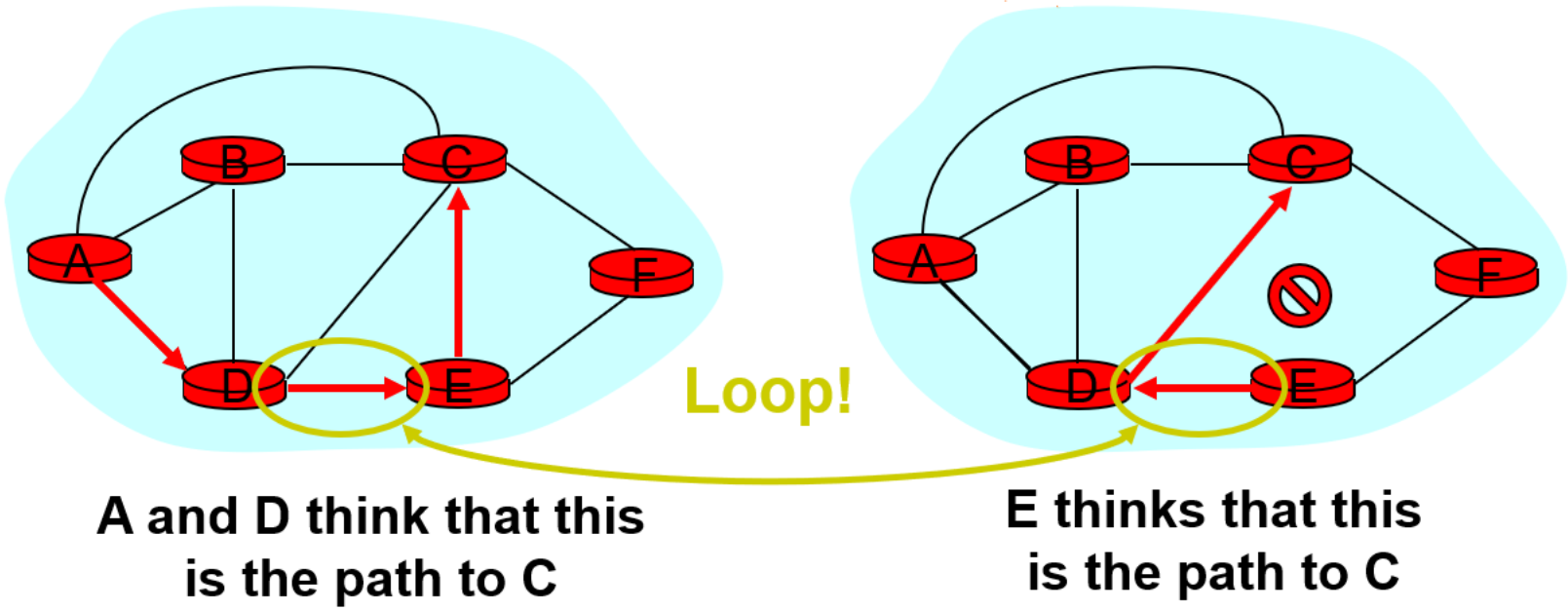
A and D think that this is the path to C



E thinks that this is the path to C

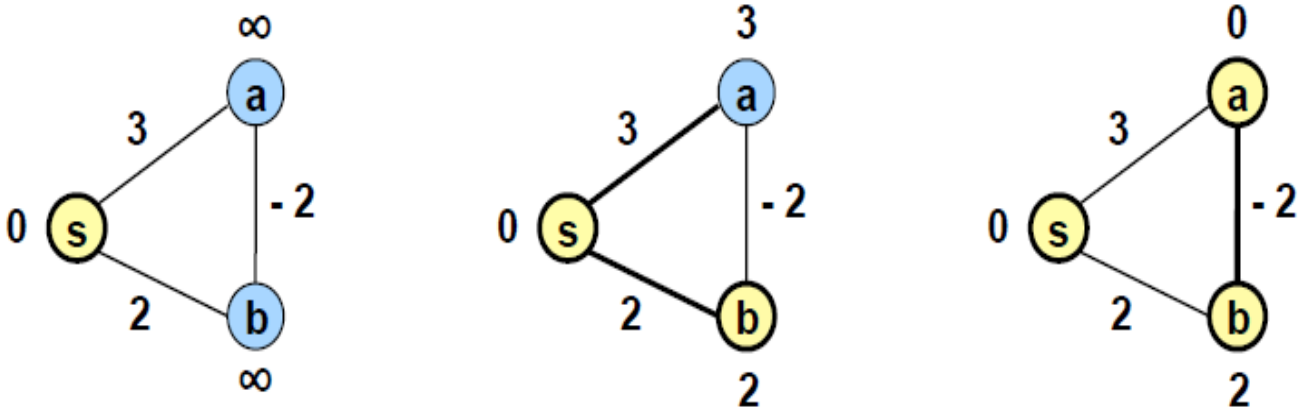
INCONSISTENZE DURANTE CONVERGENZA

- I database contenenti lo stato dei link possono essere inconsistenti
  - alcuni routers vengono a conoscenza di un guasto su un link, prima di altri
  - I cammini minimi non risultano più consistenti
  - questo può causare dei forwarding loops transitori



L'ALGORITMO DI DIJKSTRA: LIMITAZIONI

- funziona solamente con pesi degli archi maggiori di 0
- qual è il problema?



- l'algoritmo visita b, quindi a e termina associando a b una distanza di 2, invece che la distanza corretta di 1

RIDUZIONE DELLE INCONSISTENZE

- Individuazione veloce di fallimenti di link
  - HELLO MESSAGES più frequenti
  - Tecnologie a livello link capaci di individuare i guasti
- Flooding più efficiente
  - Immediata notifica di cambiamento
  - Alta priorità a notifiche link-state
- Computazione più efficiente
  - Processori ad alta performance sui routers
- Aggiornamento efficiente delle forwarding tables
  - Strutture dati che supportano aggiornamenti incrementali

DIJKSTRA SPF: IMPLEMENTAZIONE JAVA

```
// The program use an adjacency matrix to represent the graph
import java.util.*; import java.io.*;
class ShortestPath
{  // A utility function to find the vertex with minimum distance
value,
    // from the set of vertices not yet included in shortest path
tree
    final int V=8;
    int minDistance(int dist[], Boolean sptSet[])
       {  int min = Integer.MAX_VALUE, min_index=-1;
          for (int v = 0; v < V; v++)
            if (sptSet[v] == false && dist[v] <= min)
            {   min = dist[v];
                min_index = v;
            }
        return min_index; }
```

DIJKSTRA SPF: IMPLEMENTAZIONE JAVA

```
// A utility function to print the constructed distance array
    void printSolution(int dist[], int n)
    {
        System.out.println("Vertex   Distance from Source");
        for (int i = 0; i < V; i++)
            System.out.println(i+"              "+dist[i]);
    }
```

DIJKSTRA SPF: IMPLEMENTAZIONE JAVA

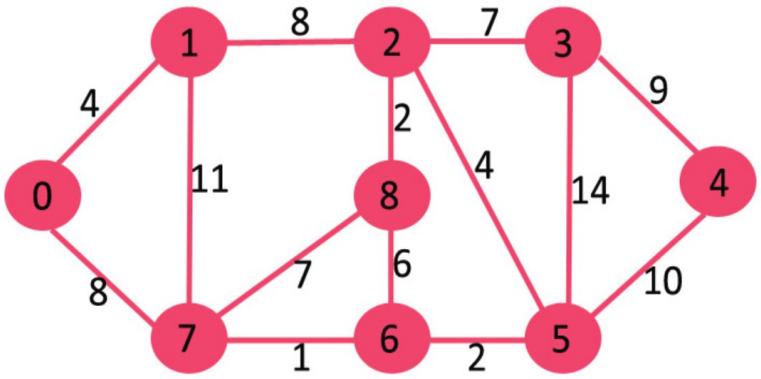
```
 // the graph is represented using an adjacency matrix
void dijkstra(int graph[][], int src)
    {   int dist[] = new int[V]; // The output array. dist[i] will hold
                                 // the shortest distance from src to i
        Boolean sptSet[] = new Boolean[V]; // sptSet[i] will true if vertex i
                                           is included in shortest
                                           // path tree or shortest distance
                                           from src to i is finalized
        // Initialize all distances as INFINITE and stpSet[] as false
        for (int i = 0; i < V; i++)
        {
            dist[i] = Integer.MAX_VALUE;
            sptSet[i] = false;
        }

        // Distance of source vertex from itself is always 0
        dist[src] = 0;
```

DIJKSTRA SPF: IMPLEMENTAZIONE JAVA

```
// Find shortest path for all vertices
for (int count = 0; count < V-1; count++)
      {  // Pick the minimum distance vertex from the set of vertices
          // not yet processed. u is always equal to src in first iteration
          int u = minDistance(dist, sptSet);
          // Mark the picked vertex as processed
          sptSet[u] = true;
          // Update dist value of the adjacent vertices of the picked vertex
          for (int v = 0; v < V; v++)
              // Update dist[v] only if is not in sptSet, there is an
              // edge from u to v, and total weight of path from src to
              // v through u is smaller than current value of dist[v]
              if (!sptSet[v] && graph[u][v]!=0 &&
                      dist[u] != Integer.MAX_VALUE &&
                      dist[u]+graph[u][v] < dist[v])
                  dist[v] = dist[u] + graph[u][v];}
   // print the constructed distance array
      printSolution(dist, V);}
```

DIJKSTRA SPF: IMPLEMENTAZIONE JAVA

```
public static void main (String[] args)
    {
        /* Ccreate the example graph discussed above */
       int graph[][] = new int[][]{{0, 4, 0, 0, 0, 0, 0, 8, 0},
                                   {4, 0, 8, 0, 0, 0, 0, 11, 0},
                                   {0, 8, 0, 7, 0, 4, 0, 0, 2},
                                   {0, 0, 7, 0, 9, 14, 0, 0, 0},
                                   {0, 0, 0, 9, 0, 10, 0, 0, 0},
                                   {0, 0, 4, 14, 10, 0, 2, 0, 0},
                                   {0, 0, 0, 0, 0, 2, 0, 1, 6},
                                   {8, 11, 0, 0, 0, 0, 1, 0, 7},
                                   {0, 0, 2, 0, 0, 0, 6, 7, 0}
                                  };
        ShortestPath t = new ShortestPath();
        t.dijkstra(graph, 0);
    }}
```



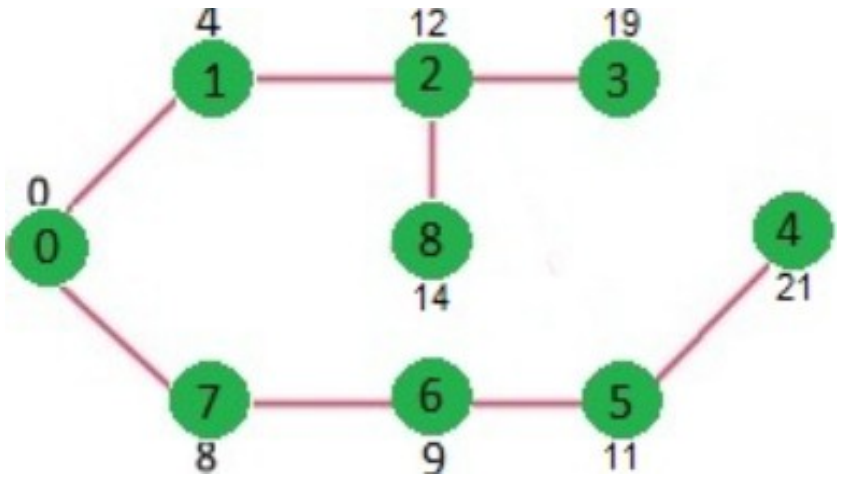
Ovviamente il programma dovrebbe prendere in input il numero di nodi del grafo

E la matrice di adiacenza: modificarlo

DIJKSTRA SPF: IMPLEMENTAZIONE JAVA

Risultato ottenuto:

```
Vertex   Distance from Source
0                0
1                4
2                12
3                19
4                21
5                11
6                9
7                8
```



OSPF OPEN SHORTEST PATH FIRST

- un protocollo che utilizza il link state routing, utilizzato per il routing intra-dominio
- oltre l'algoritmo di routing supporto per:
  - broadcasting
  - flooding di link state routing updates
    - periodico
    - event triggered
- numeri di sequenza per distinguere updates più recenti da quelli più datati
- supporto di diverse metriche per i pesi dei link

CONCLUSIONI

- Maggior problema del link-state è la scalabilità
- L'overhead del protocollo dipende dalla topologia
  - Banda: flooding dei link state advertisement
  - Memoria: memorizzazione della topologia
  - Elaborazione: calcolo degli shortest paths
- Possibile ottimizzazione: introdurre "aree gerarchiche di routing"