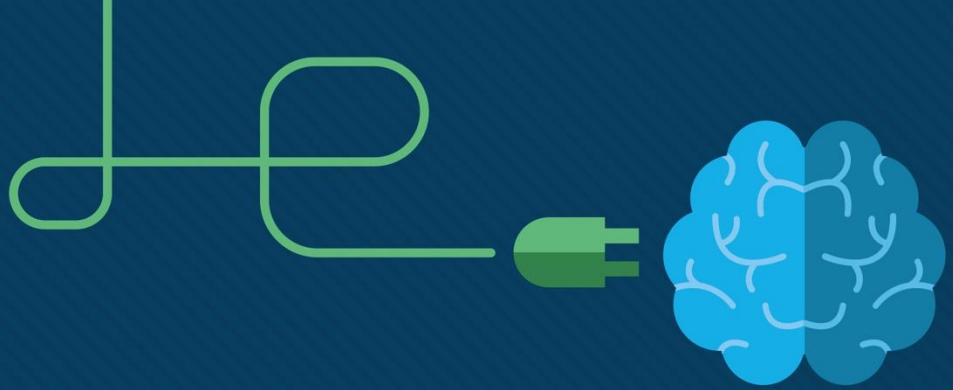
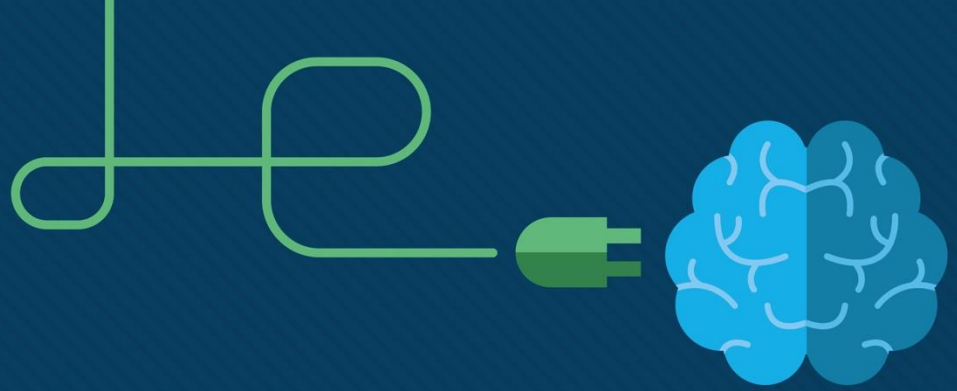


# Module 2: Dijkstra and Bellman-Ford

Enterprise Networking, Security, and Automation v7.0 (ENSA)



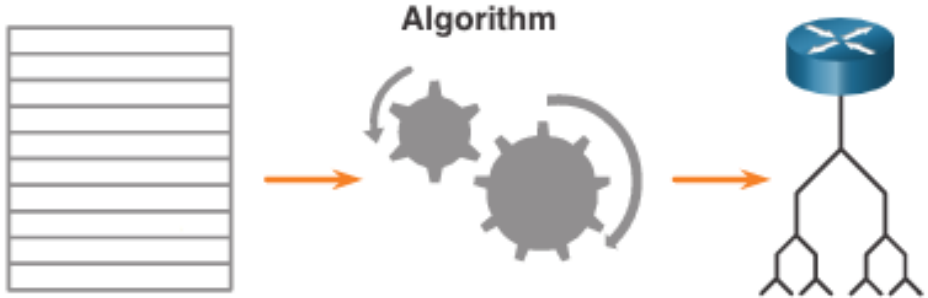
# Algoritmo Dijkstra

Enterprise Networking, Security, and Automation v7.0 (ENSA)

# Components of OSPF (Cont.)

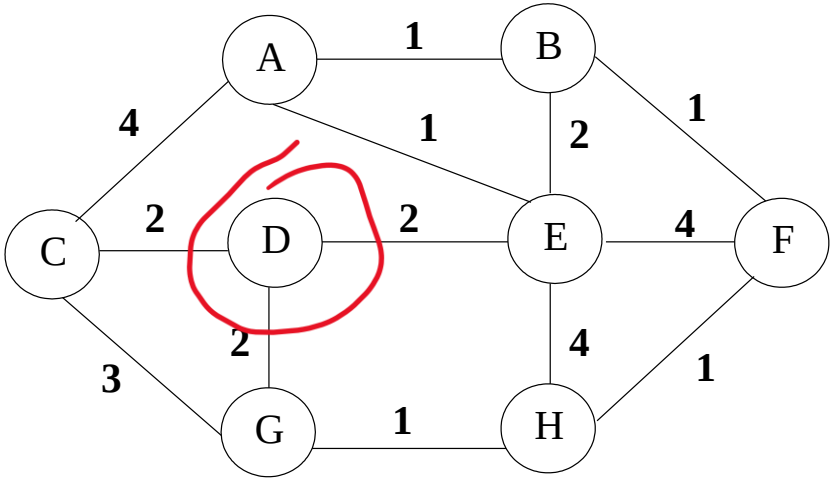
- The router builds the **routing table** from the **topology table** based on the Dijkstra **shortest-path first (SPF)** algorithm. The SPF algorithm is based on the cumulative cost to reach a destination.
- The SPF algorithm creates an SPF tree by placing each router at the root of the tree and calculating the shortest path to each node. The SPF tree is then used to calculate the best routes. OSPF places the best routes into the forwarding database, which is used to make the routing table.

| Database | Table |
|---|---|
| Adjacency Database | Neighbor Table |
| Link-state Database (LSDB) | Topology Table |
| Forwarding Database | Routing Table |



# Algoritmo Dijkstra

Albero di partenza

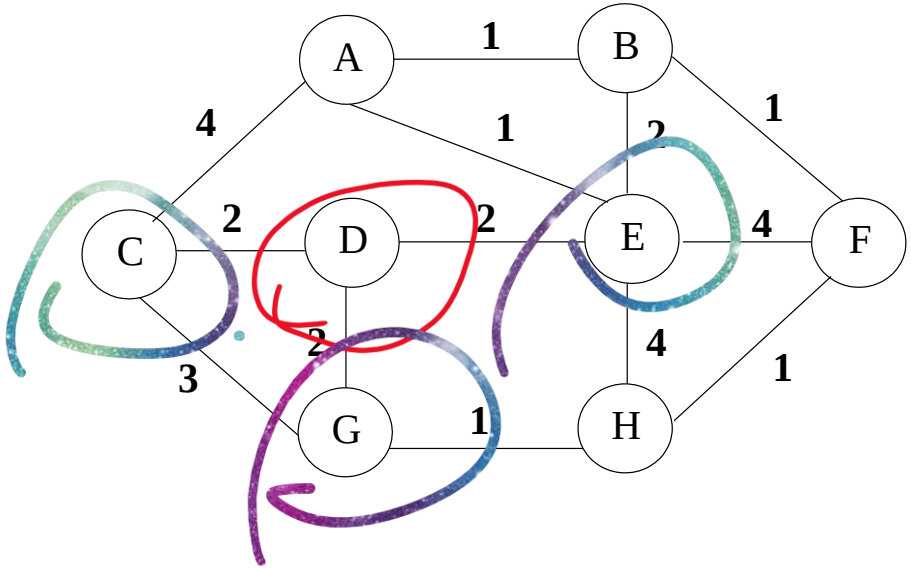



Dalla rete si crea un grafo pesato con pesi per link uguali ai costi di inoltro su ciascuno di questi

Consideriamo l'applicazione al nodo D che diventa radice (D è il nodo che sta implementando l'algoritmo per popolare la routing table). L'obiettivo è individuare l'albero con costi minimi da questo nodo radice costruito su questo grafo

**Step 1**: Al primo passo l'albero è costituito dal solo nodo radice

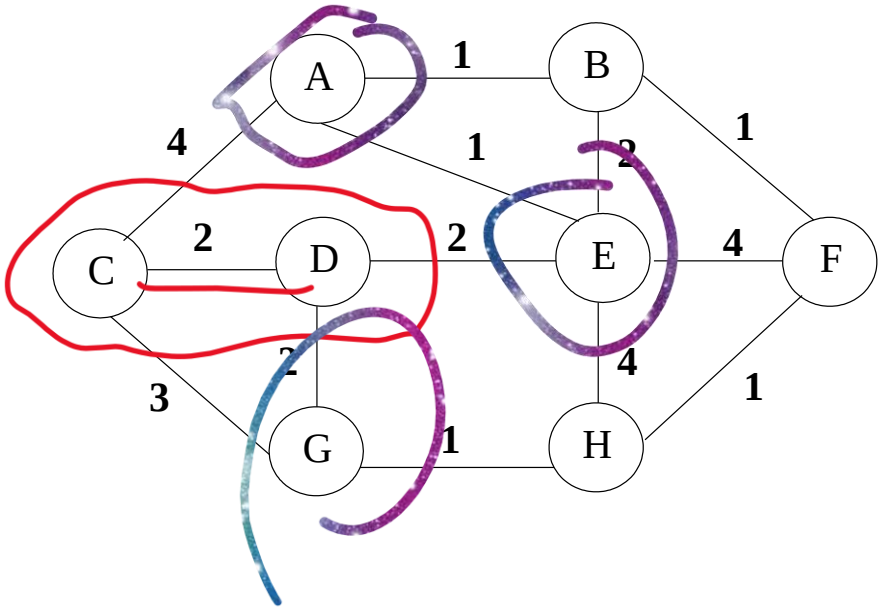
# Algoritmo Dijkstra




L'algoritmo procede per step e ad ogni step viene aggiunto all'albero attuale il nodo (tra quelli adiacenti all'albero attuale) con il costo più basso verso la radice passando solo su nodi dell'albero attuale

**Step 2:** In questo caso i candidati adiacenti sono C, G ed E, tutti a costo 2
A parità di costo si prende uno a caso. Il risultato non cambierà.
Supponiamo di scegliere C

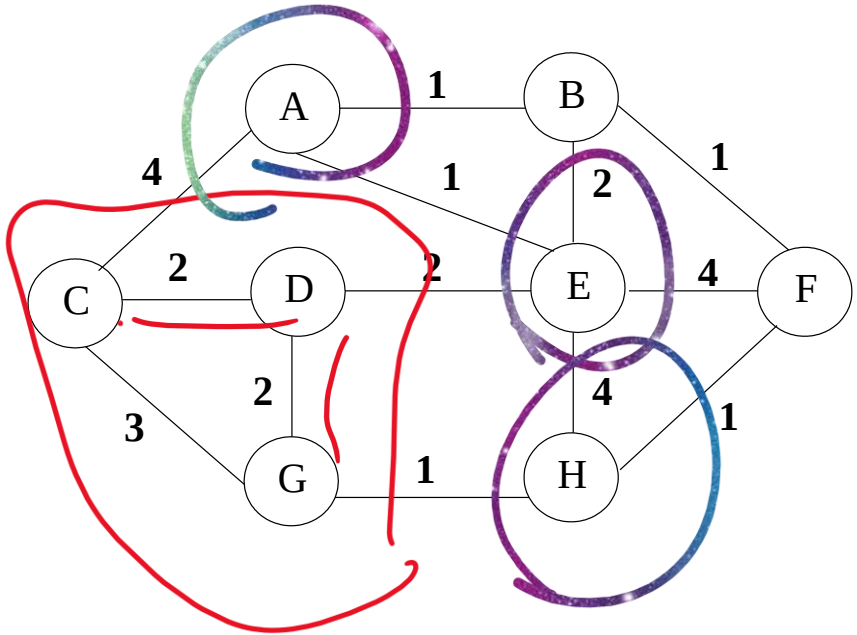
# Algoritmo Dijkstra




**Step 3:** Adesso i candidati sono: A con costo 6, E con costo 2 e G con costo 2

Scegliamo G

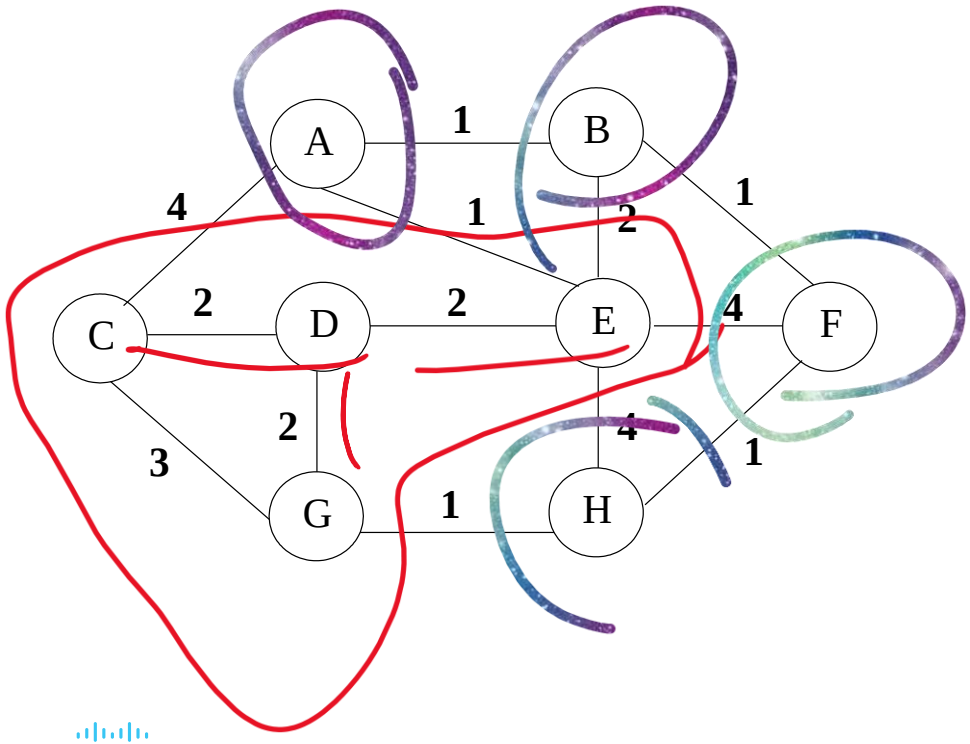
# Algoritmo Dijkstra




**Step 4:** Adesso i candidati sono: A con costo 6, E con costo 2 e H con costo 3

Scegliamo E

# Algoritmo Dijkstra

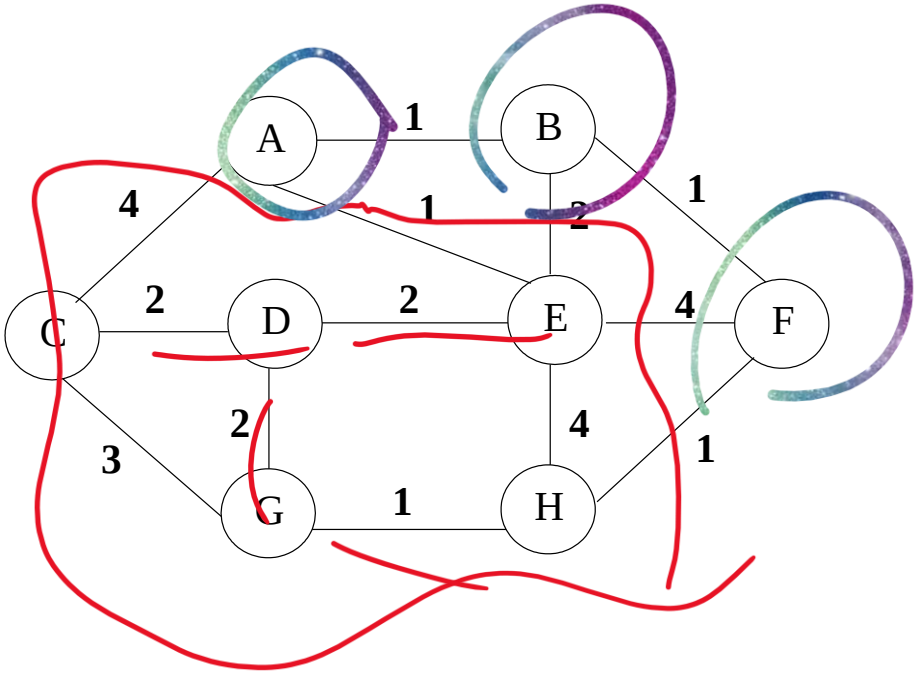



**Step 5:** Adesso i candidati sono: A con costo 3, B con costo 4 e F con costo 6 e H con costo 3

Notare che il costo dei nodi adiacenti cambia nei vari passi. Per esempio A è passato da 6 a 3

Scegliamo H

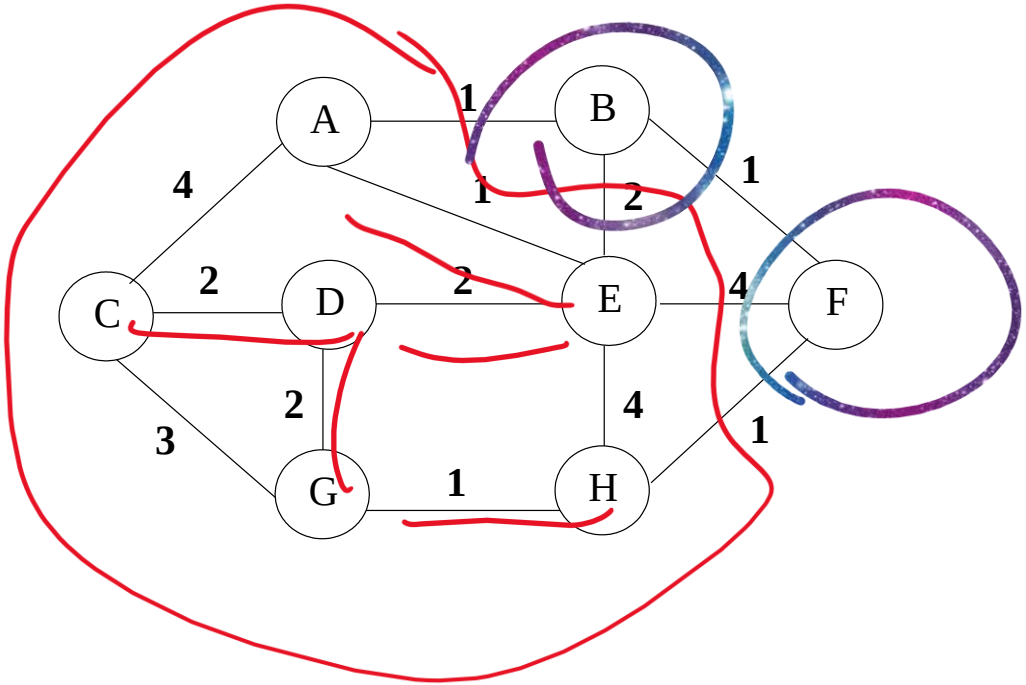
# Algoritmo Dijkstra




**Step 6:** Adesso i candidati sono: A con costo 3, B con costo 4 e F con costo 4

Scegliamo A

# Algoritmo Dijkstra

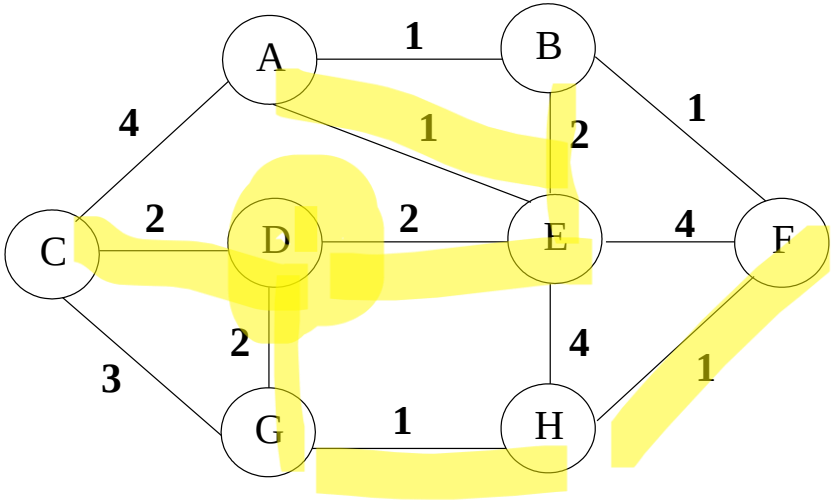



**Step 7:** Adesso i candidati sono: B con costo 4 e F con costo 4

Scegliamo F e poi B (**Step 8 finale**)

# Algoritmo Dijkstra

Albero di partenza

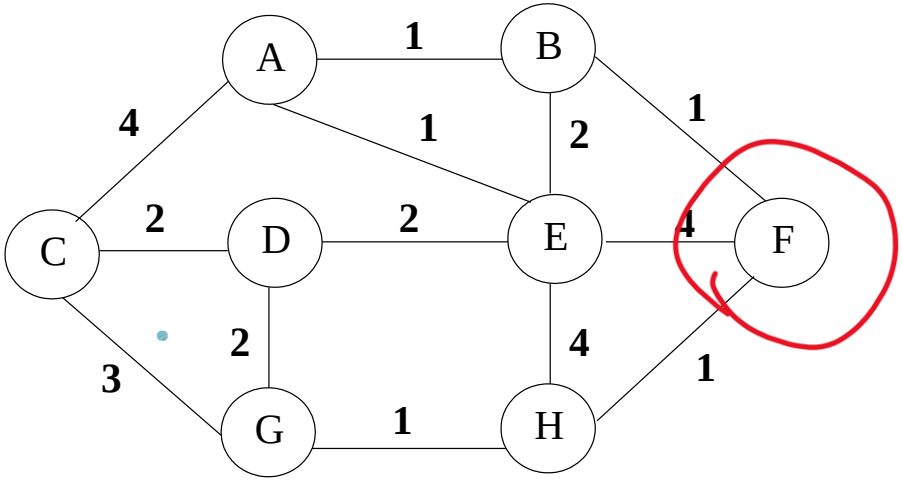



In giallo l'albero finale ottenuto con Dijikstra

Da questa verrà poi creata la tabella di routing per le varie router (sono i collegamenti tra i vari router)

# Algoritmo Dijkstra

Esercizio:
considerare F come nodo radice

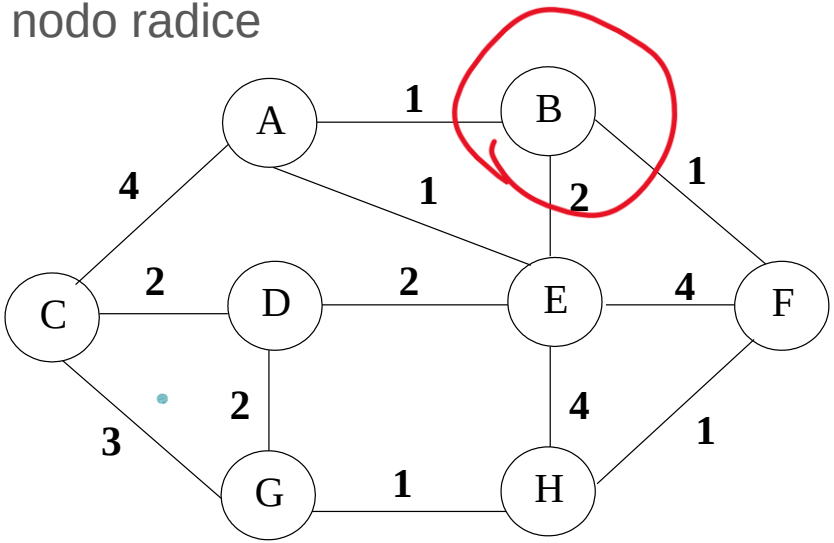



L'algoritmo procede per step e ad ogni step viene aggiunto all'albero il nodo (tra quelli adiacenti all'albero attuale) con il costo più basso verso la radice passando solo su nodi dell'albero attuale

Esercizio: costruite l'albero partendo dal nodo radice F

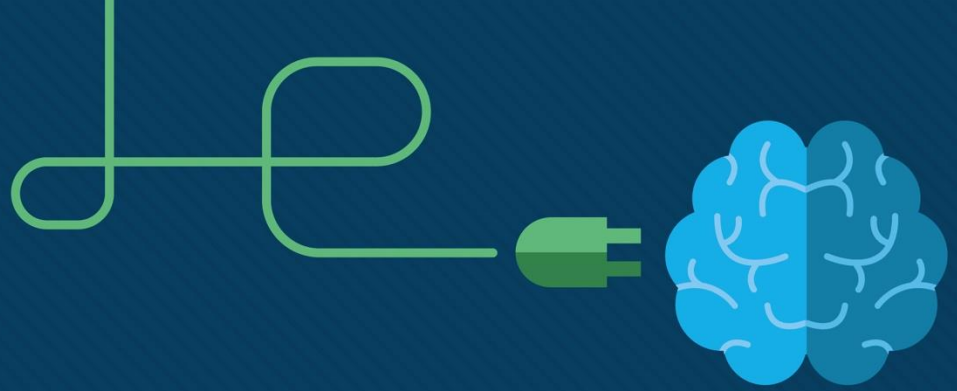
# Algoritmo Dijkstra

Esercizio:
considerare B come nodo radice




L'algoritmo procede per step e ad ogni step viene aggiunto all'albero il nodo (tra quelli adiacenti all'albero attuale) con il costo più basso verso la radice passando solo su nodi dell'albero attuale

Esercizio: costruite l'albero partendo dal nodo radice B

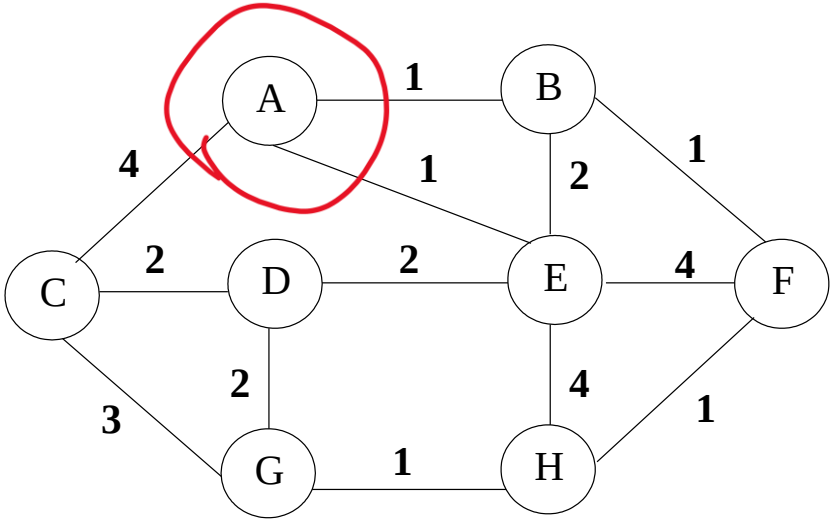


# Bellman-Ford

Enterprise Networking, Security, and Automation v7.0 (ENSA)

# Algoritmo Bellman-Ford

Albero di partenza



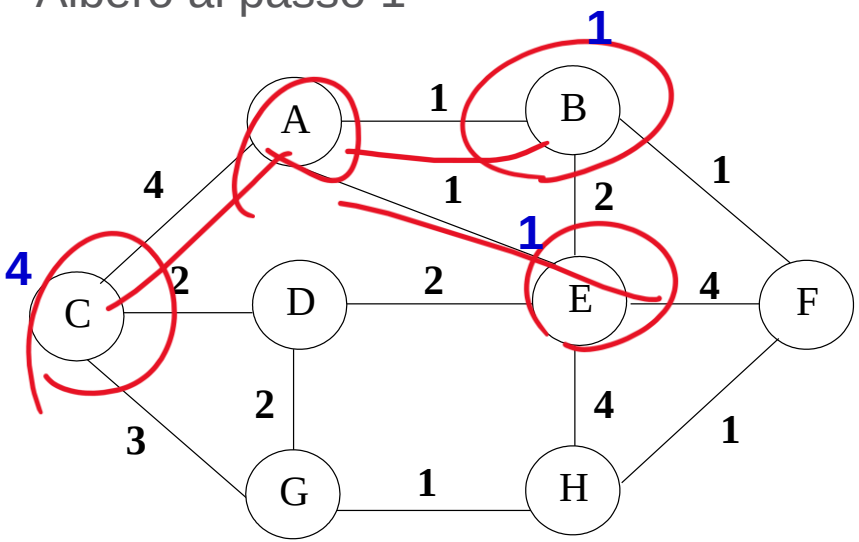

Anche in questo caso si deve considerare il grafo ed un nodo radice (quello rispetto al quale calcolare i percorsi). Supponiamo di considerare A

Ad ogni passo *i* stavolta si aggiorna un albero dove tutti nodi con un numero di salti al più uguale ad *i* dalla radice ne fanno parte. Si prende chiaramente il percorso a costo più basso

Al primo step *i=0*
Solo A fa parte dell'albero

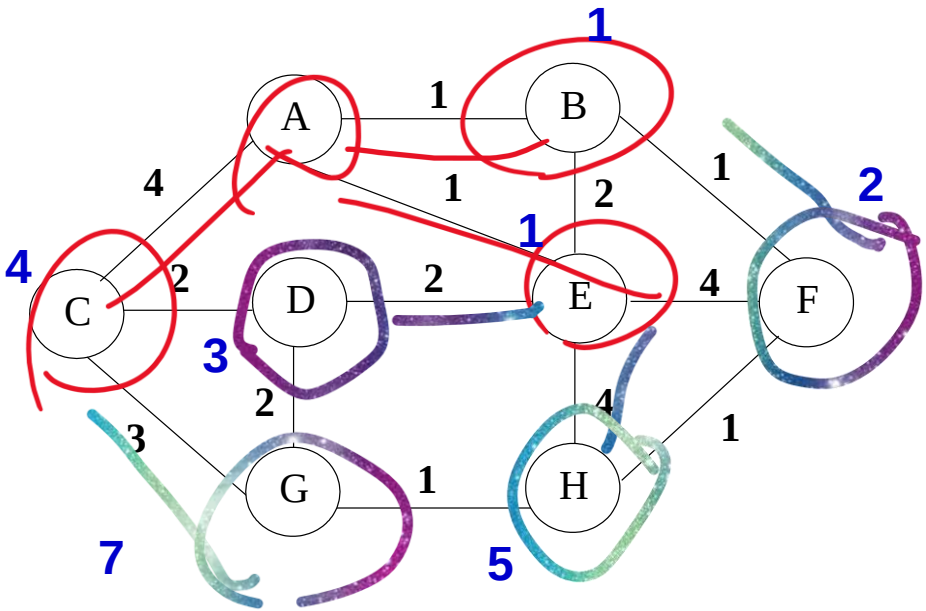
# Algoritmo Bellman-Ford

Albero al passo 1




Allo step successivo *i=1* vengono inclusi B, E e C (raggiungibili con un solo salto dalla radice) con costo per raggiungerli 1, 1, 4, rispettivamente

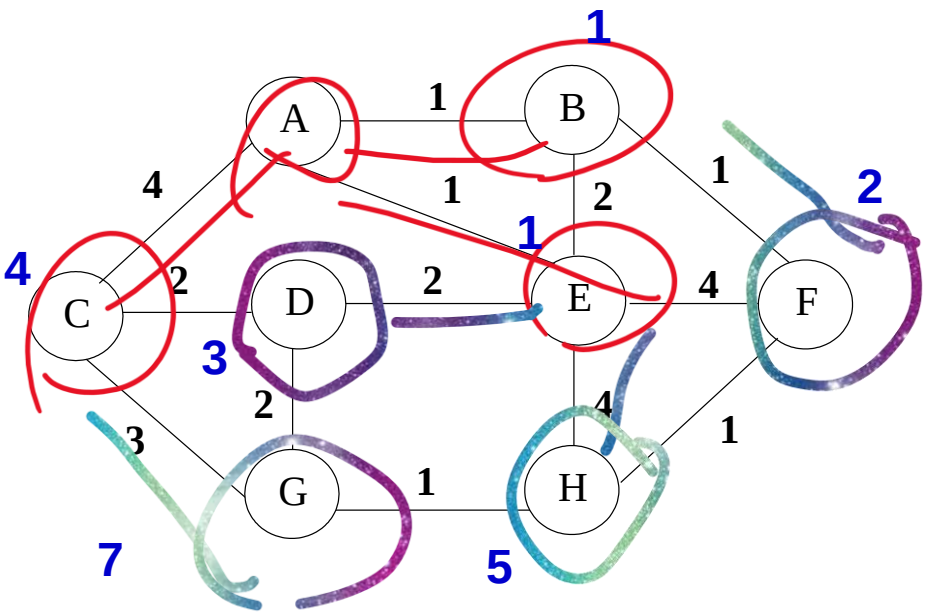
# Algoritmo Bellman-Ford




Al passo *i=2* aggiungiamo i nodi raggiungibili dalla radice con percorsi con al più due salti. Sono di fatto i nodi adiacenti l'albero attualmente costruito.

Vengono riconsiderati anche i nodi già presenti al passo *i* = 1 ma raggiungibili con due salti ed inclusi nel nuovo albero con due salti se più convenienti (p.e. E. tramite B e B tramite E).

# Algoritmo Bellman-Ford



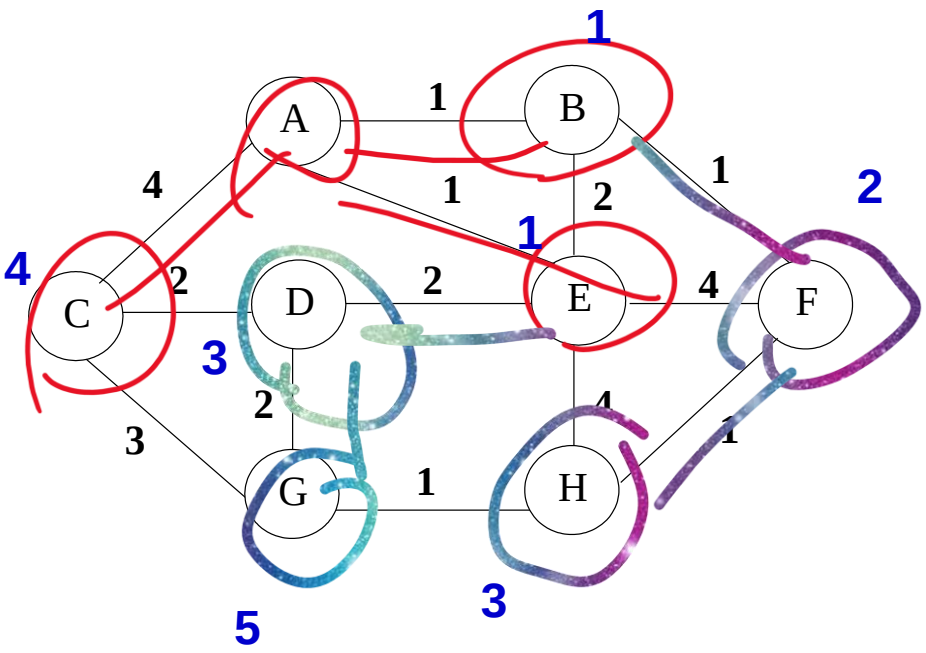

L'albero in questo caso ha raggiunto tutti i nodi ma non è detto che sia quello con i percorsi più brevi

Infatti possono esserci dei percorsi a tre salti (siamo al passo *i=3*) più convenienti

A tal fine vengono riconsiderati i costi per raggiungere i nodi già presenti tramite quelli aggiunti al passo 2. In questo caso quelli interessanti sono (ma devo verificarli tutti):

- G tramite D (con tre salti costo 5)
- H tramite F (con tre salti costo 3)

# Algoritmo Bellman-Ford

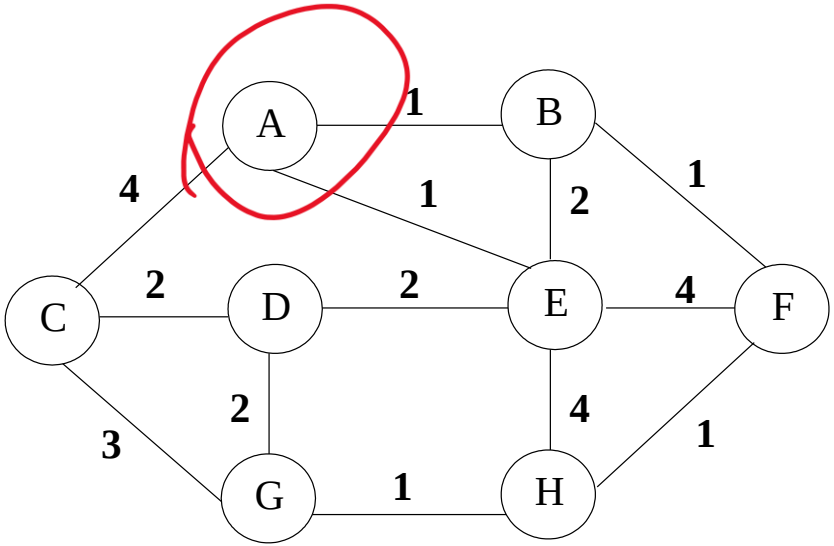



Passo *i=3*
- G tramite D
- H tramite F

Passiamo al passo *i=4*
Ci sono altre modifiche da fare?

L'algoritmo termina al passo nel quale non vengono introdotte delle modifiche rispetto a quello precedente in quanto non ci sono percorsi più lunghi a costo minore

# Algoritmo Bellman-Ford Implementazione distribuita (p.e., nel RIP)




Questo algoritmo può essere implementato in modo distribuito

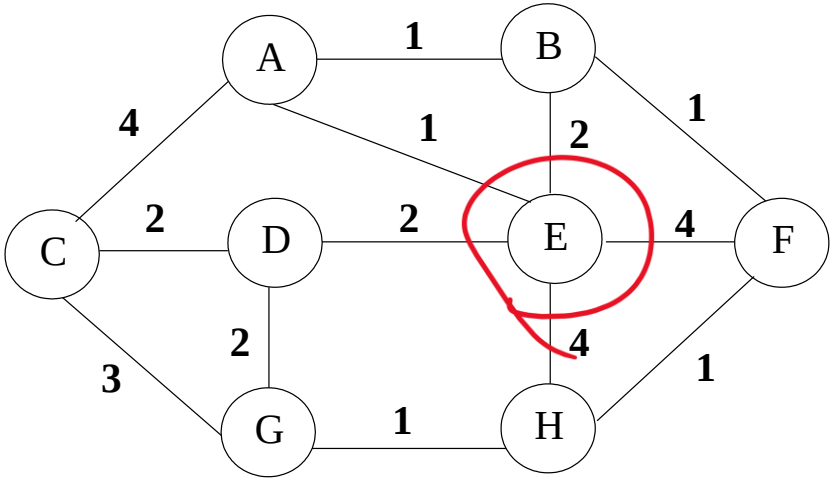
Ogni nodo riceve informazioni dai nodi adiacenti (p.e. A ricevere da C, B, E) sulle distanza che loro conoscono al momento per arrivare ad ogni altro nodo.

Il nodo che ha ricevuto questa informazione somma al costo ricevuto quello verso i nodi che hanno inviato queste distanze.

Con il passare delle iterazioni si scoprono anche percorsi più lunghi che eventualmente hanno costi più bassi e che quindi sostituiscono i precedenti.

# Algoritmo Bellman-Ford

Albero di partenza:
esercizio partendo da E



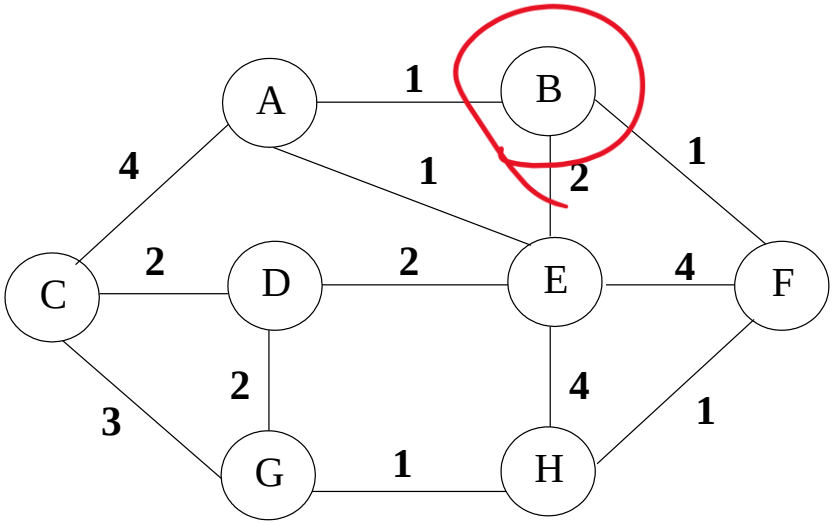

Anche in questo caso si deve considerare il grafo ed un nodo radice (quello rispetto al quale calcolare i percorsi)
Supponiamo di considerare E stavolta

Ad ogni passo *i* stavolta si aggiorna un albero dove tutti nodi con un numero di salti al più uguale ad *i* ne fanno parte. Si prende chiaramente il percorso a costo più basso

Esercizio: consideriamo E e costruiamo l'albero

# Algoritmo Bellman-Ford

Albero di partenza:
esercizio partendo da E




Anche in questo caso si deve considerare il grafo ed un nodo radice (quello rispetto al quale calcolare i percorsi)
Supponiamo di considerare E stavolta

Ad ogni passo *i* stavolta si aggiorna un albero dove tutti nodi con un numero di salti al più uguale ad *i* ne fanno parte. Si prende chiaramente il percorso a costo più basso

Esercizio: consideriamo B e costruiamo l'albero