

# Algoritmi e Strutture Dati

## Capitolo 13

Cammini minimi:  
algoritmo di Dijkstra

**Cammini minimi in grafi:**  
cammini minimi a singola sorgente  
(senza pesi negativi)

# Cammini minimi in grafi pesati

Sia  $G=(V,E,w)$  un grafo orientato o non orientato con pesi  $w$  **reali** sugli archi. Il **costo** o **lunghezza** di un cammino

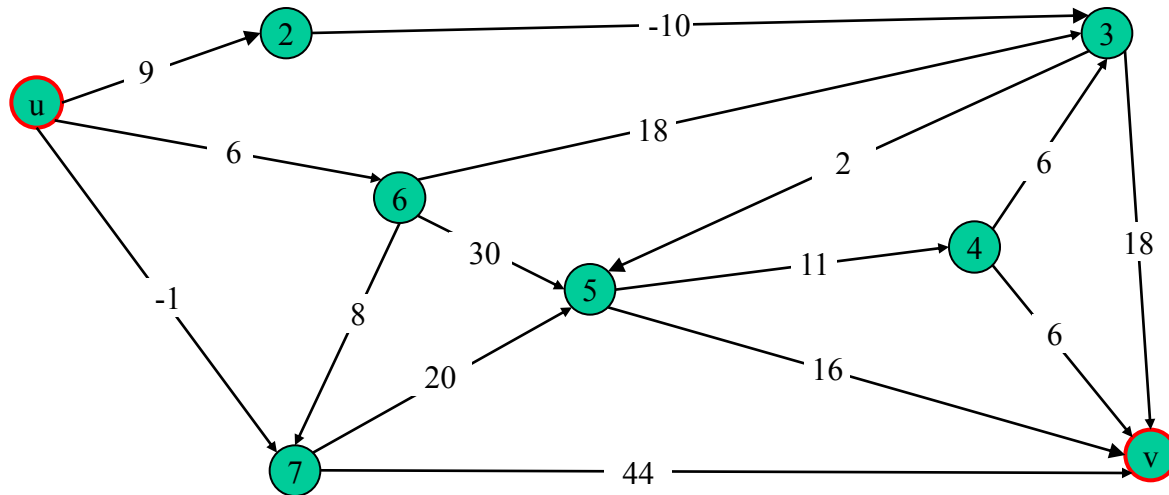
$\pi=\langle v_0, v_1, v_2, \dots, v_k \rangle$  è:

$$w(\pi) = \sum_{i=1}^k w(v_{i-1}, v_i)$$

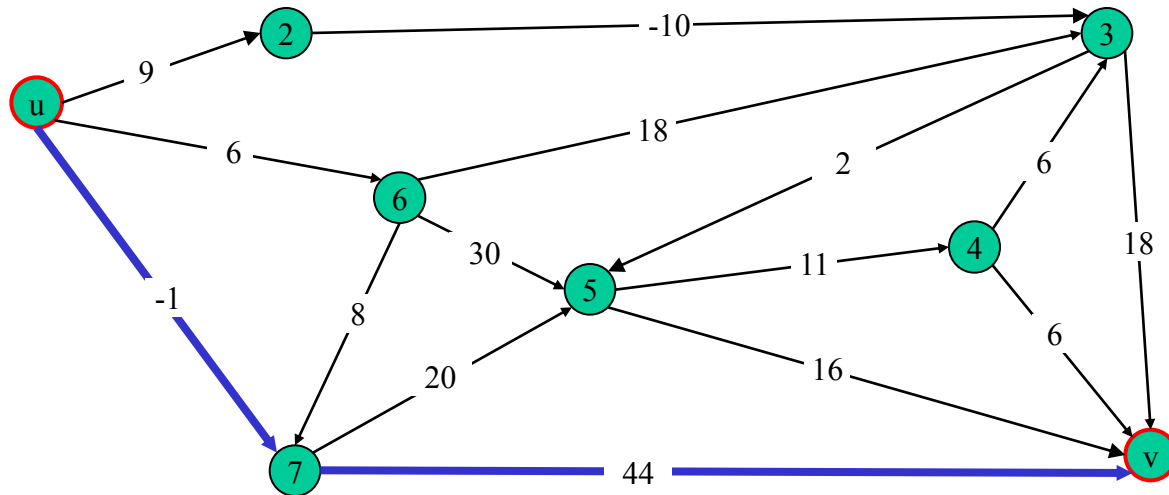
Un **cammino minimo** tra una coppia di vertici  $x$  e  $y$  è un cammino avente **costo minore o uguale** a quello di ogni altro cammino tra gli stessi vertici.

**NOTA:** Il cammino minimo non è necessariamente unico.

# Esempio: cammino minimo su un grafo pesato

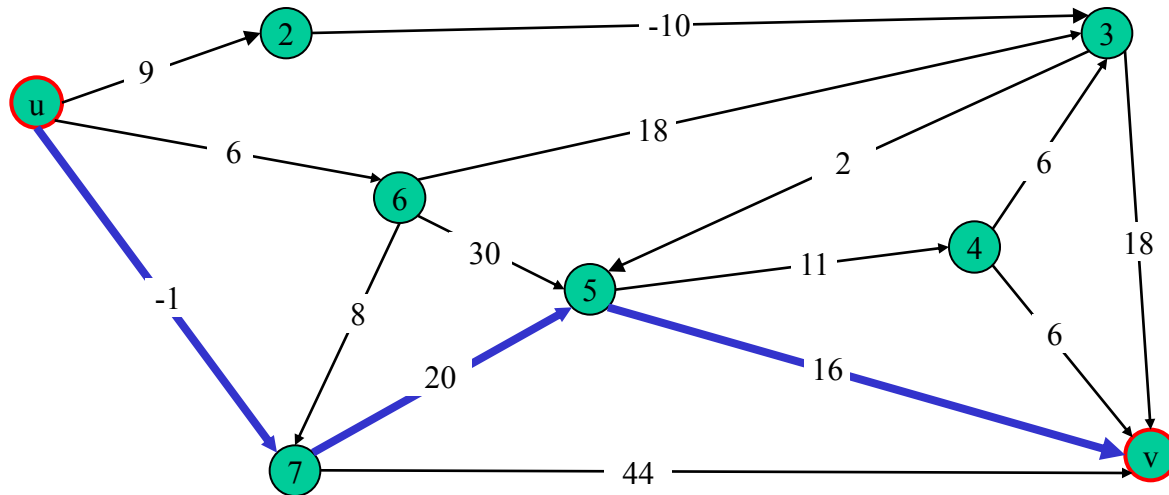


# Esempio: cammino minimo su un grafo pesato



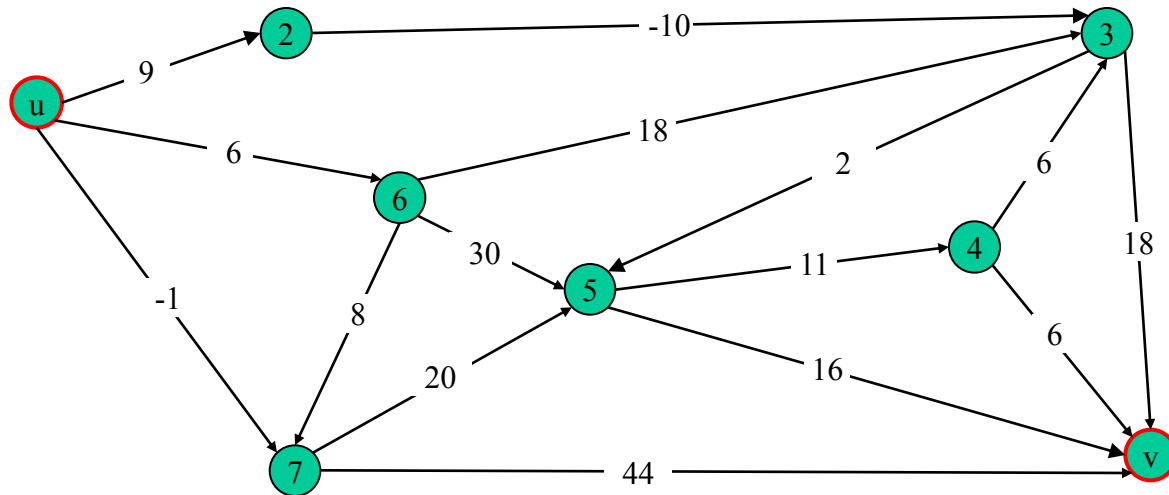
cammino di  
lunghezza  
43

# Esempio: cammino minimo su un grafo pesato



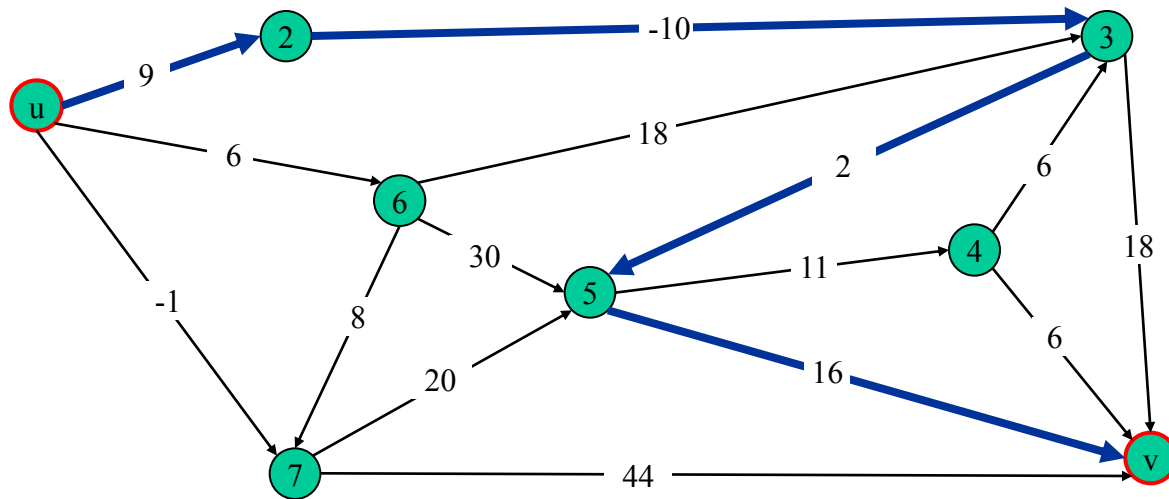
cammino di  
lunghezza  
35

# Esempio: cammino minimo su un grafo pesato



la distanza  $d_G(u,v)$  da  $u$  a  $v$  in  $G$  è il costo di un qualsiasi cammino minimo da  $u$  a  $v$ .

# Esempio: cammino minimo su un grafo pesato



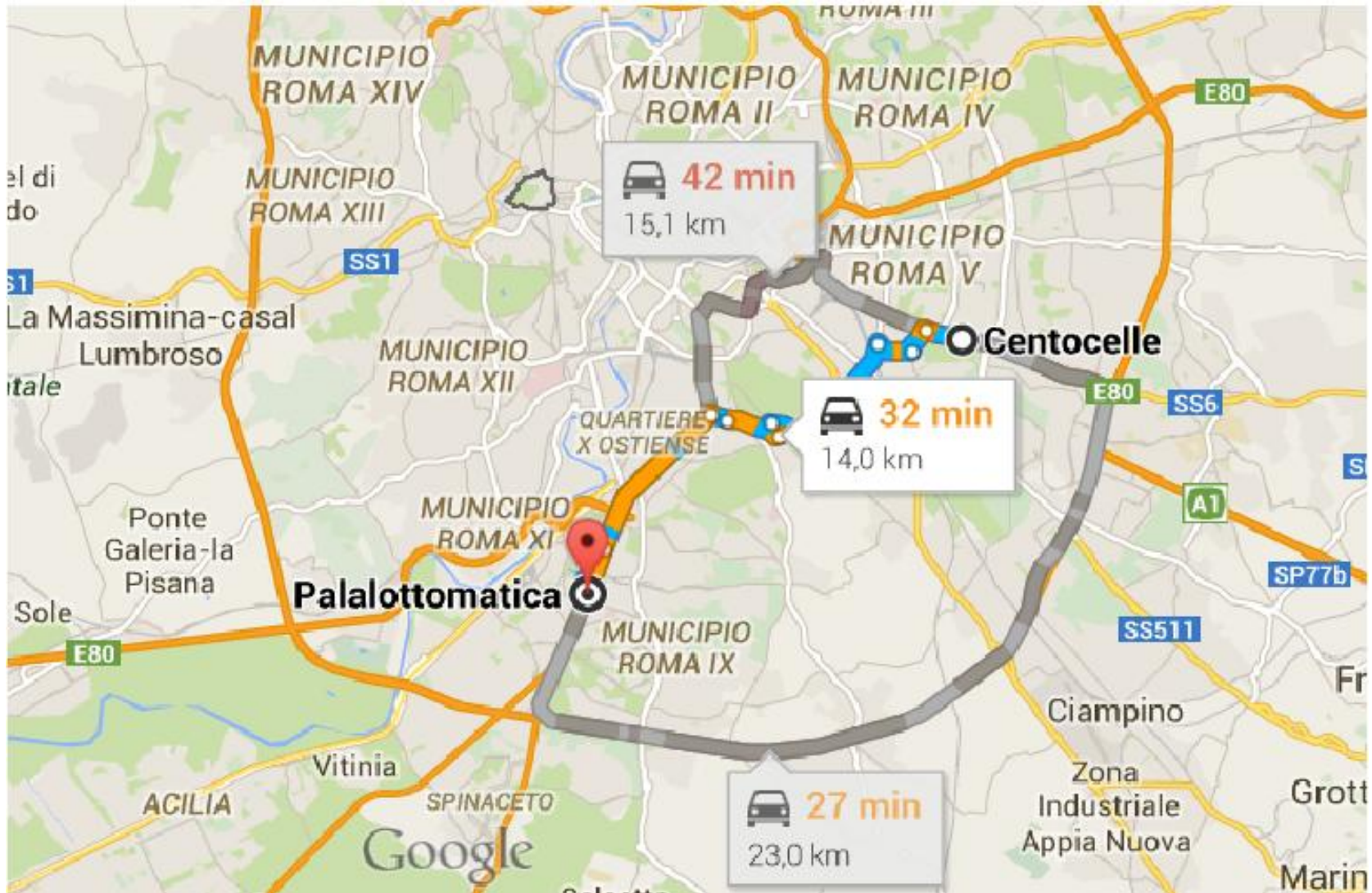
la distanza  $d_G(u,v)$  da  $u$  a  $v$  in  $G$  è il costo di un qualsiasi cammino minimo da  $u$  a  $v$ .

$$d_G(u,v)=17$$

**Problema:** dati  $u$  e  $v$ , trovare un cammino minimo (e/o distanza) da  $u$  a  $v$



problema:  
trovare il cammino minimo fra due nodi



problema:  
trovare il cammino minimo fra due nodi

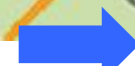


pesi archi:  
lunghezze



strada  
più breve

pesi archi:  
tempo  
percorrenza



strada  
più veloce

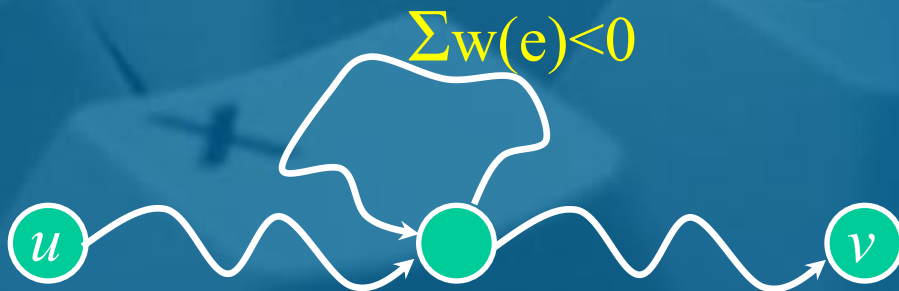


esiste sempre un cammino  
minimo fra due nodi?



...no!

- se non esiste nessun cammino da  $u$  a  $v$ 
  - $d(u,v)=+\infty$
- se c'è un cammino che contiene un ciclo il cui costo è **negativo**
  - $d(u,v)=-\infty$

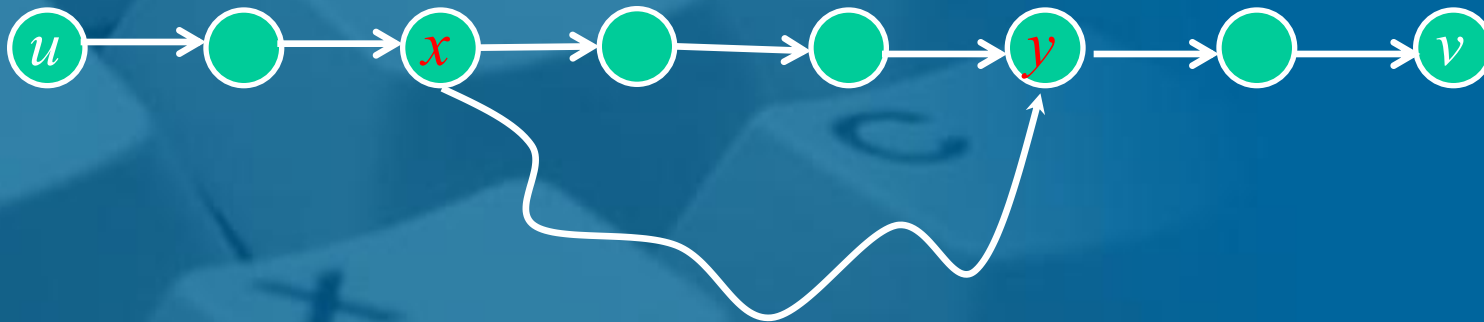


**Oss:** se  $G$  non contiene cicli negativi, esistono cammini minimi che sono cammini **semplici**

# sottostruttura ottima

Ogni sottocammino di un cammino minimo è un cammino minimo.

dim: tecnica cut&paste



ipotetico cammino  
più corto da  $x$  a  $y$

allora il cammino  
da  $u$  a  $v$  non era minimo!

# disuguaglianza triangolare

per ogni  $u, v, x \in V$ , vale:

$$d(u,v) \leq d(u,x) + d(x,v)$$



il cammino da  $u$  a  $v$  che passa per  $x$  è un cammino nel grafo e quindi il suo costo è almeno il costo del cammino minimo da  $u$  a  $v$

# Cammini minimi a singola sorgente



# Problema del calcolo dei cammini minimi a singola sorgente

Due varianti:

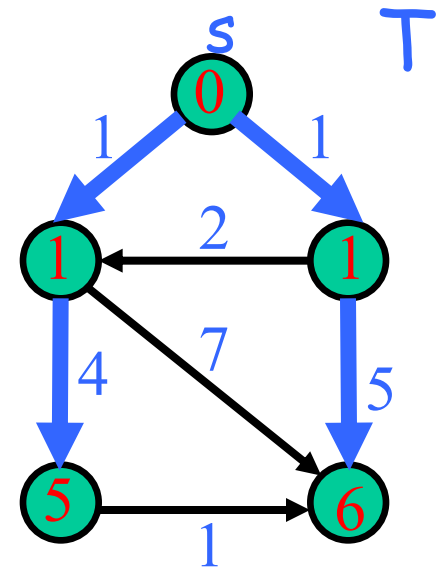
- Dato  $G=(V,E,w)$ ,  $s \in V$ , calcola le distanze di tutti i nodi da  $s$ , ovvero,  $d_G(s,v)$  per ogni  $v \in V$
- Dato  $G=(V,E,w)$ ,  $s \in V$ , calcola l'*albero dei cammini minimi* di  $G$  radicato in  $s$



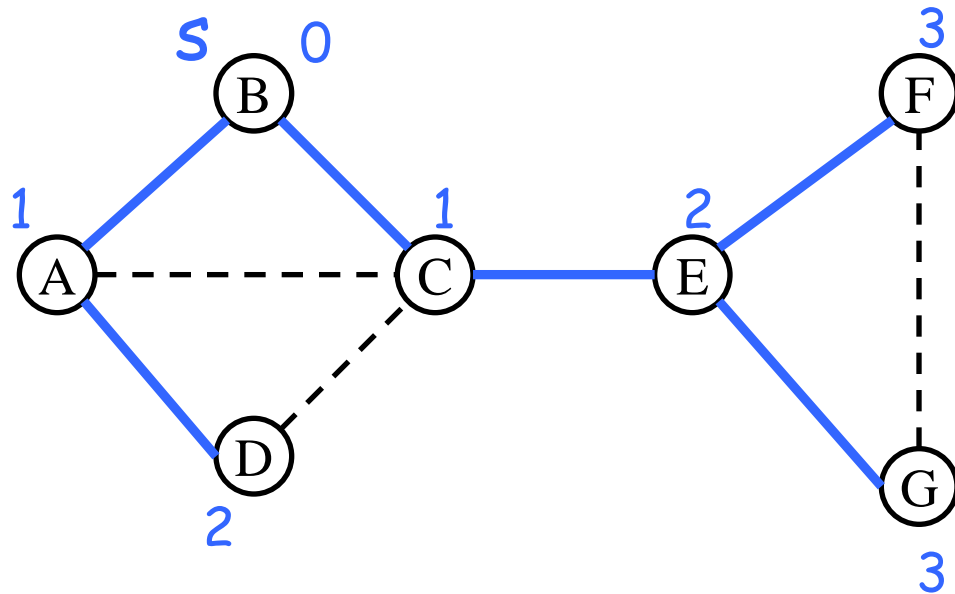
# Albero dei cammini minimi (o Shortest Path Tree - SPT)

$T$  è un **albero dei cammini minimi** con sorgente  $s$  di un grafo  $G=(V,E,w)$  se:

- $T$  è un albero radicato in  $s$
- per ogni  $v \in V$ , vale:  
 $d_T(s,v) = d_G(s,v)$



# Albero dei cammini minimi (o Shortest Path Tree - SPT)

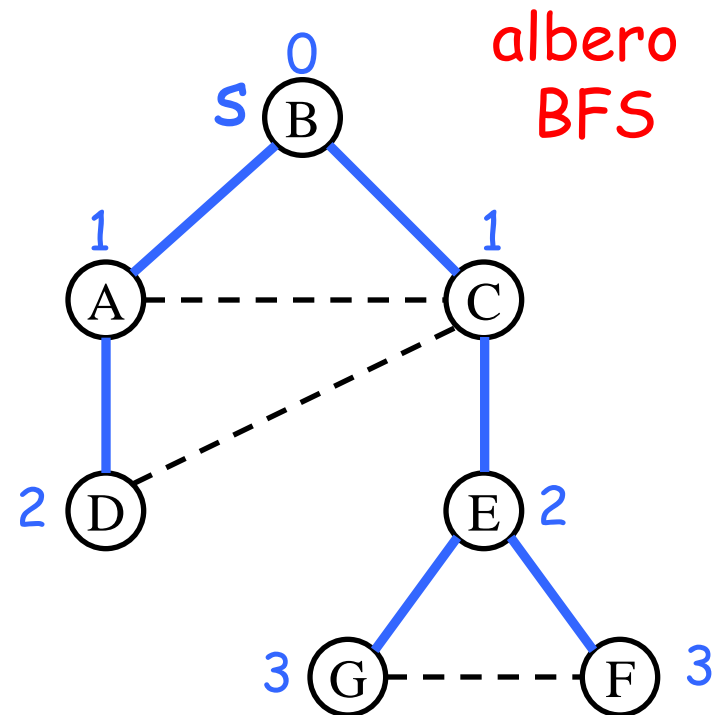


per grafi non pesati:

SPT radicato in s

=

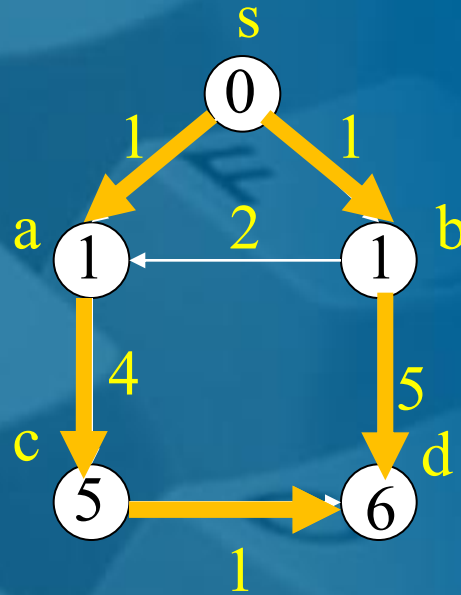
Albero BFS radicato in s



# Albero dei cammini minimi

- L'**unione di tutti i cammini minimi** da un vertice  $s$  a tutti i vertici da esso raggiungibili nel grafo  $G$  genera un sottografo di  $G$ , detto **sottografo dei cammini minimi con sorgente in  $s$**
- Se da tale sottografo rimuoviamo archi in modo tale da ridurre ad 1 il **grado entrante** di tutti i nodi (escluso  $s$  che ha grado entrante pari a 0) otterremo un **albero** orientato con tutti gli archi orientati in direzione delle foglie, detto **albero dei cammini minimi radicato in  $s$**

# Esempio di albero dei cammini minimi



oppure

# Esercizio

1. Progettare un algoritmo che, dato un grafo diretto e pesato  $G=(V,E,w)$  e un suo albero dei cammini minimi radicato in un nodo  $s$ , calcola in tempo lineare le distanze di ogni nodo da  $s$ .
2. Progettare un algoritmo che, dato un grafo diretto e pesato  $G=(V,E,w)$  e le distanze di ogni nodo da un nodo  $s$ , calcola in tempo lineare un albero dei cammini minimi di  $G$  radicato in  $s$ .

**Osservazione:** le due varianti del problema sono essenzialmente equivalenti

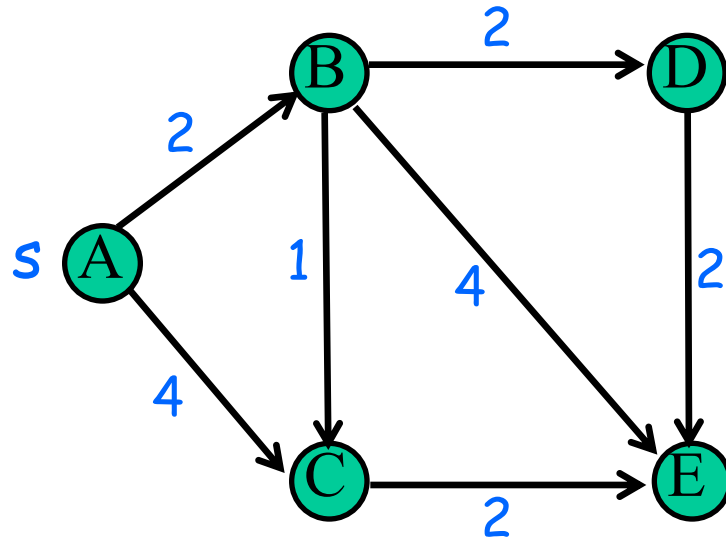
# Algoritmo di Dijkstra

**Assunzione:** tutti gli archi hanno peso non negativo, ovvero ogni arco  $(u,v)$  del grafo ha peso  $w(u,v) \geq 0$

→ cammini minimi esistono se esistono i cammini dalla sorgente

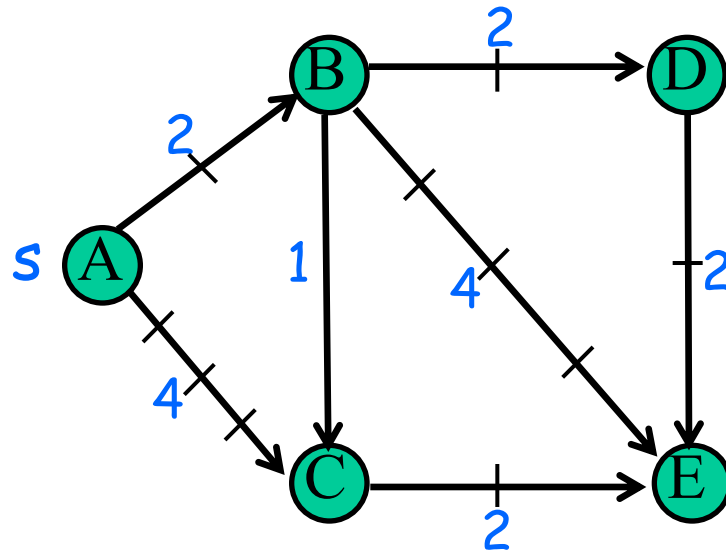
→  $d(s,v) \geq -\infty$

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



archi come **tubi**  
peso degli archi  
come **lunghezza**  
acqua **scorre** a  
**velocità costante**

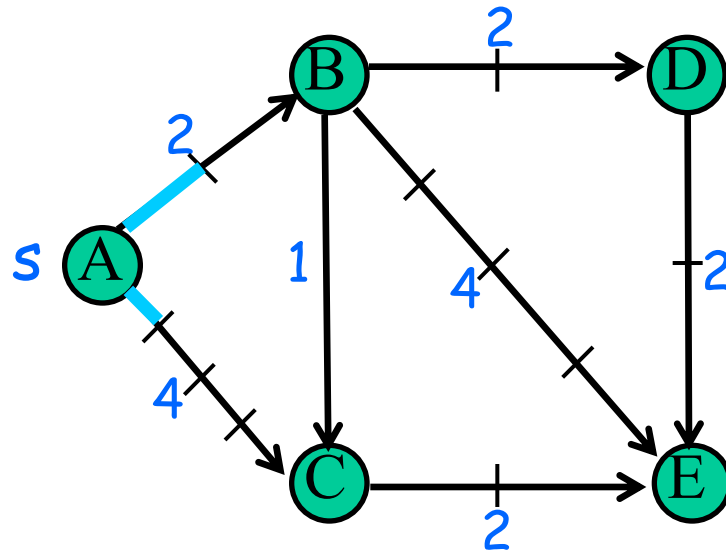
# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



archi come **tubi**  
peso degli archi  
come **lunghezza**  
acqua **scorre** a  
**velocità costante**

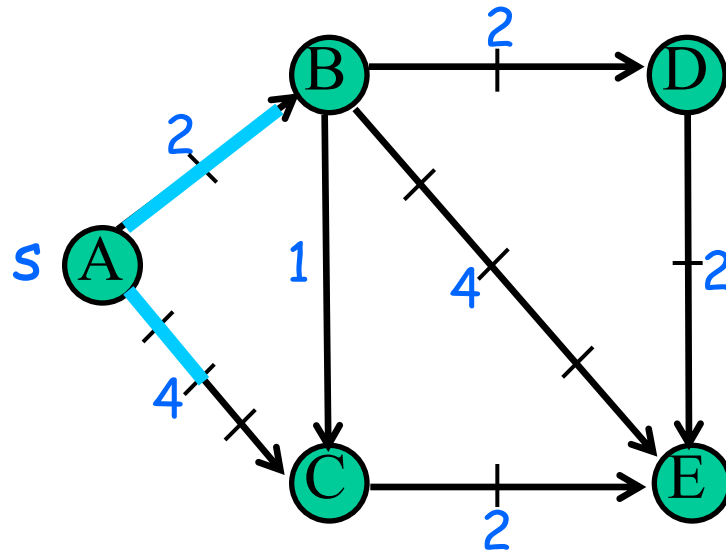


# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



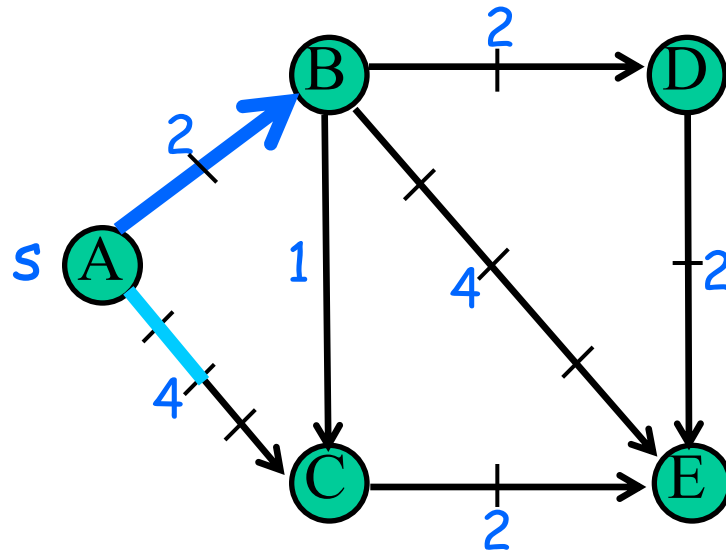
archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



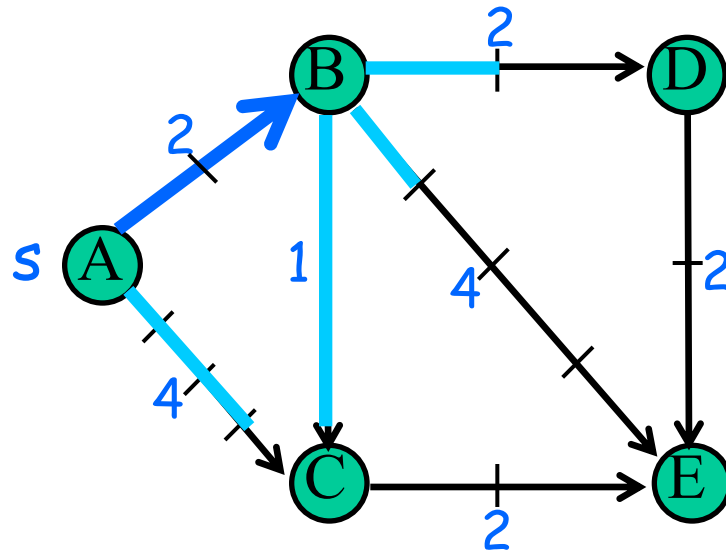
archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



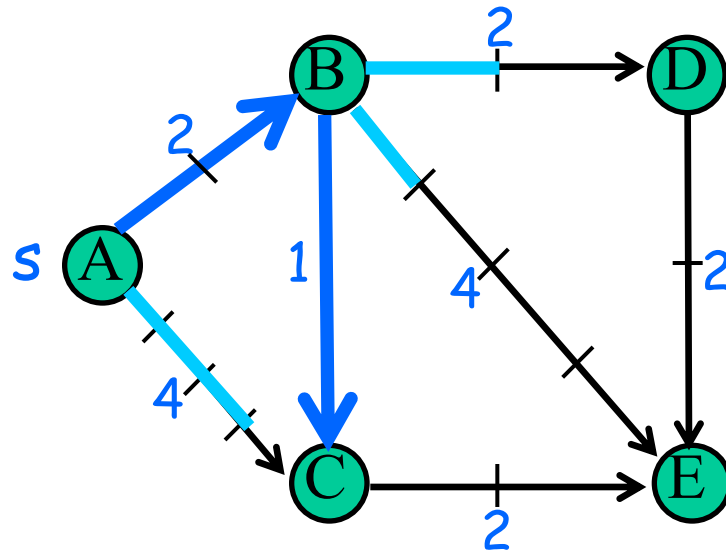
archi come **tubi**  
peso degli archi  
come **lunghezza**  
acqua **scorre** a  
**velocità costante**

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



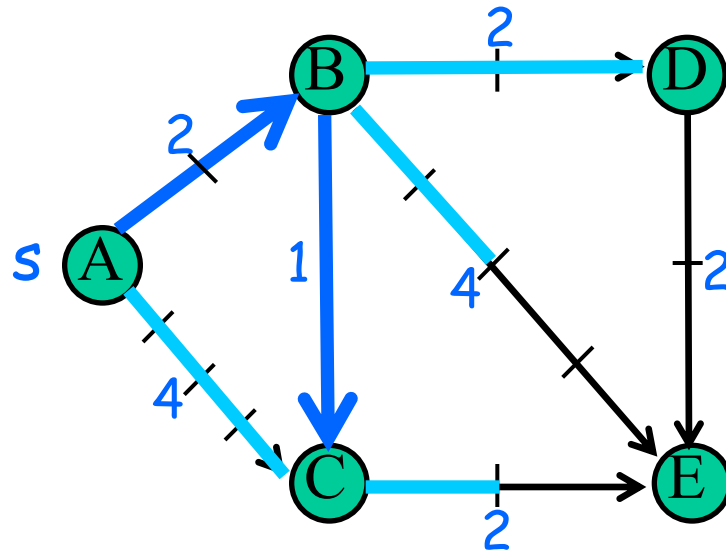
archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



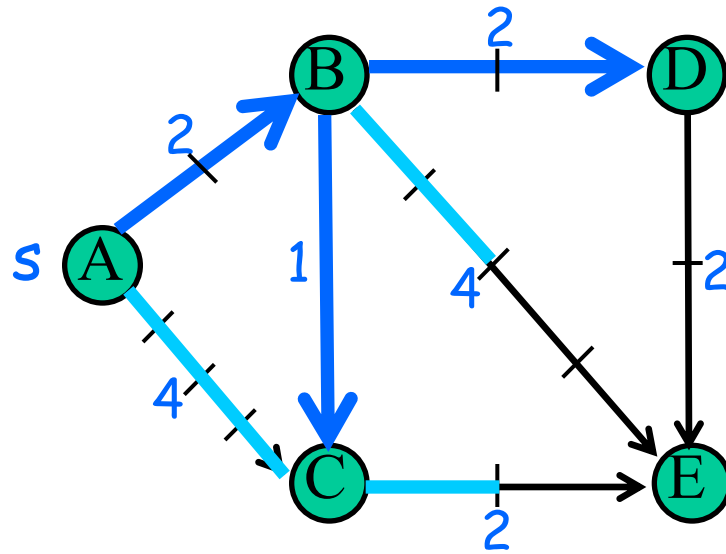
archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



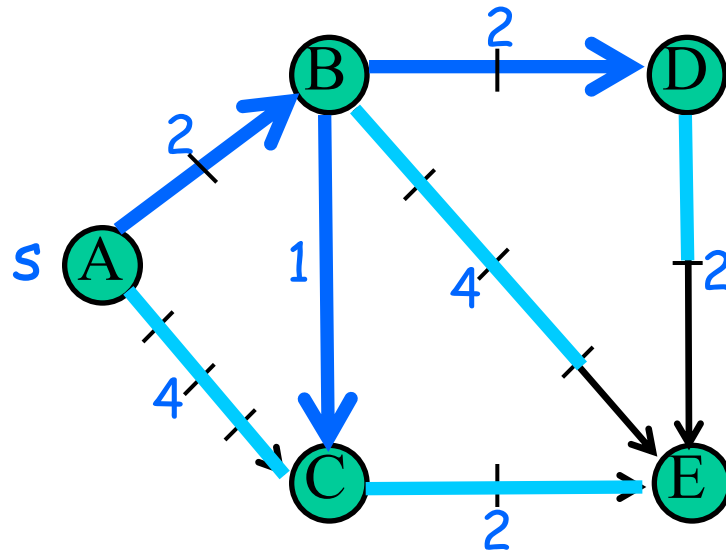
archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

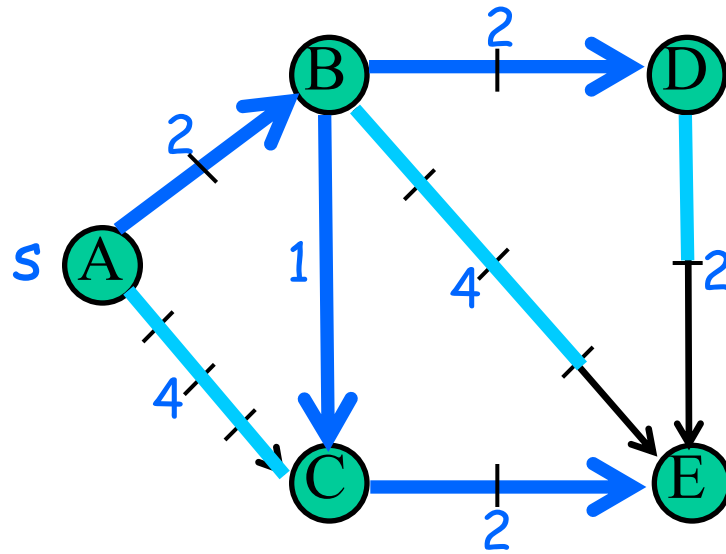
# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



archi come **tubi**  
peso degli archi  
come **lunghezza**  
acqua **scorre** a  
**velocità costante**

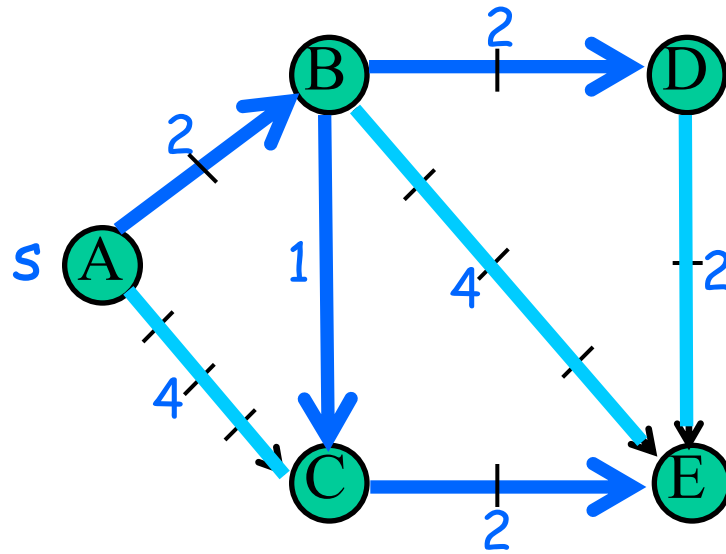


# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



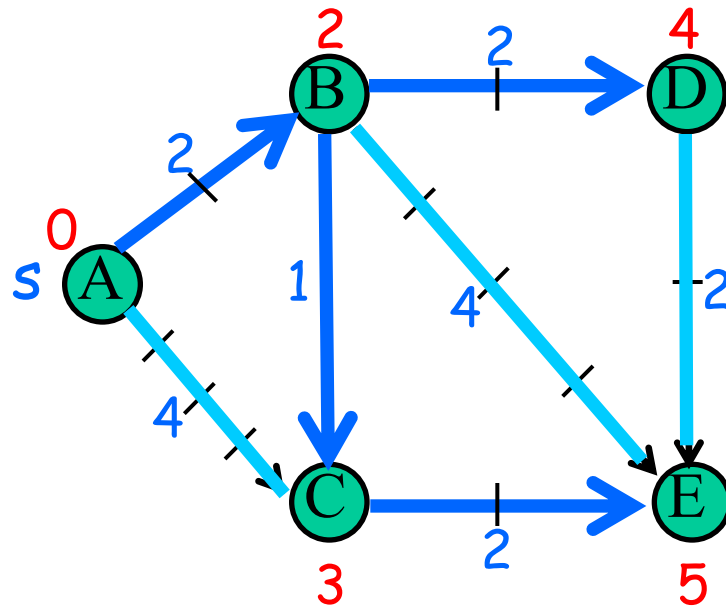
archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



archi come **tubi**  
peso degli archi  
come **lunghezza**  
acqua **scorre** a  
**velocità costante**

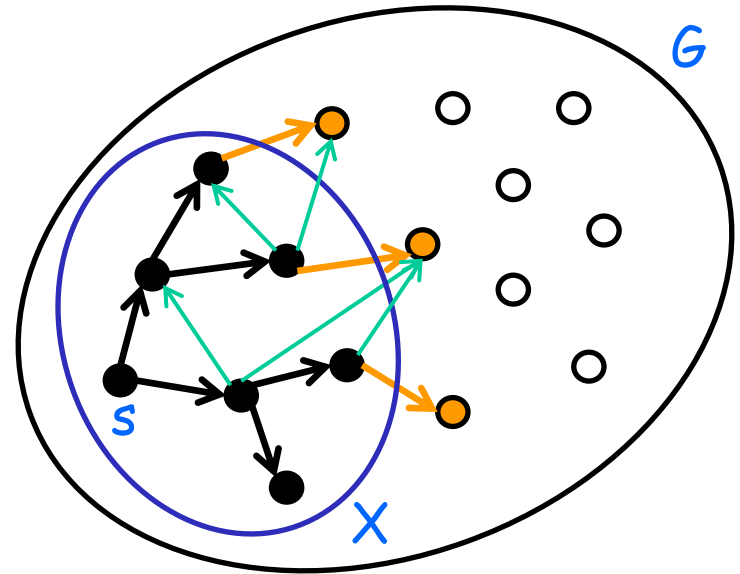
# Idea intuitiva dell'algoritmo: pompare acqua nella sorgente



archi come tubi  
peso degli archi  
come lunghezza  
acqua scorre a  
velocità costante

# Verso l'algoritmo: approccio greedy (goloso)

1. mantiene per ogni nodo  $v$  una stima (per eccesso)  $D_{sv}$  alla distanza  $d(s,v)$
2. mantiene un insieme  $X$  di nodi per cui le stime sono esatte; e anche un albero  $T$  dei cammini minimi verso nodi in  $X$  (albero nero). Inizialmente  $X=\{s\}$ ,  $T$  non ha archi.
3. ad ogni passo aggiunge a  $X$  il nodo  $u$  in  $V-X$  la cui stima è minima; aggiunge a  $T$  uno specifico arco (arancione) entrante in  $u$
4. aggiorna le stime guardando i nodi adiacenti a  $u$



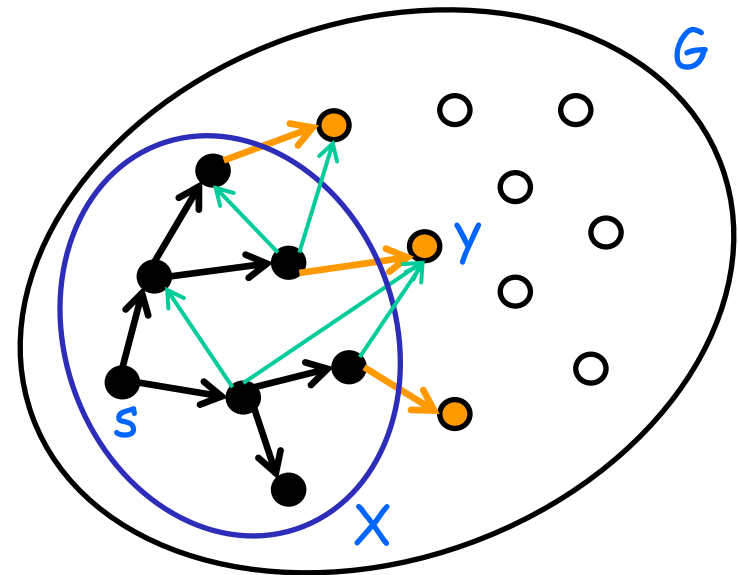
I nodi da aggiungere progressivamente a  $X$  (e quindi a  $T$ ) sono mantenuti in una **coda di priorità**, associati ad **un unico arco** (arco arancione) che li connette a  $T$ .

la stima per un nodo  $y \in V - X$  è:

$$D_{sy} = \min \{ D_{sx} + w(x, y) : (x, y) \in E, x \in X \}.$$

L'arco che fornisce il minimo è l'arco arancione.

Se  $y$  è in coda con arco  $(x, y)$  associato, e se dopo aver aggiunto  $u$  a  $T$  troviamo un arco  $(u, y)$  tale che  $D_{su} + w(u, y) < D_{sx} + w(x, y)$ , allora **rimpiazziamo**  $(x, y)$  con  $(u, y)$ , ed aggiorniamo  $D_{sy}$

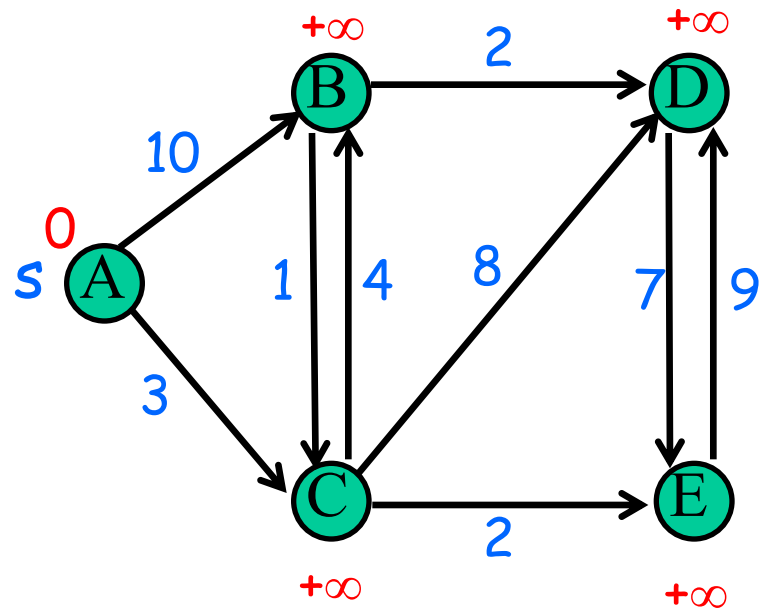


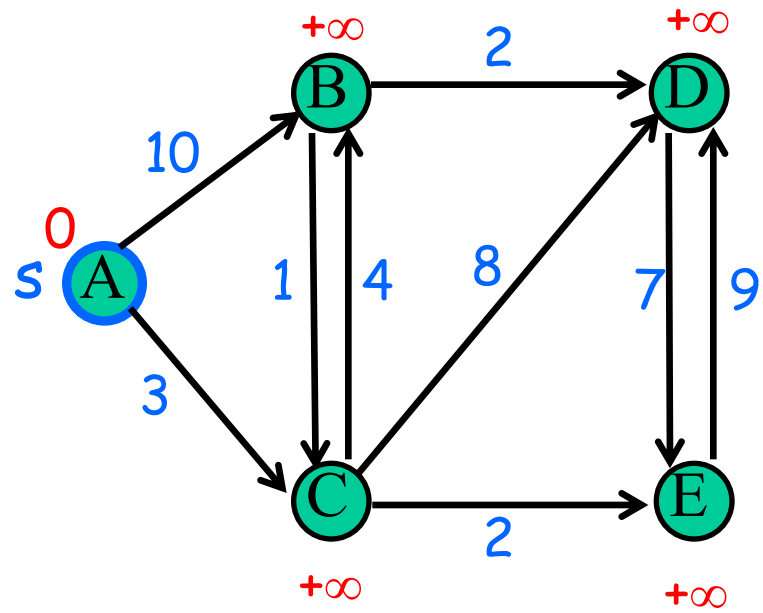
- nodi per i quali non è stato "scoperto" nessun cammino; stima =  $+\infty$
- nodi "scoperti"; hanno stima  $< +\infty$  sono mantenuti in una coda con priorità insieme al "miglior" arco entrante (arancione)

# Pseudocodice

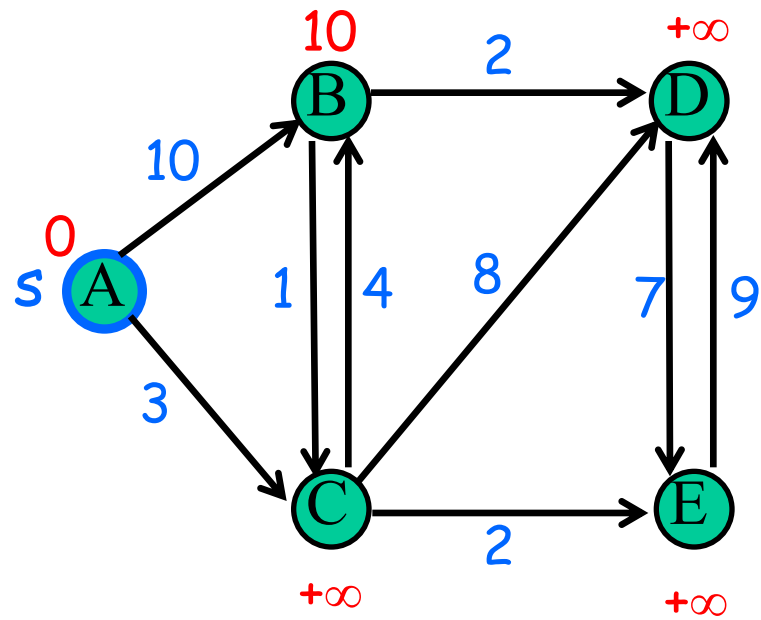
```
algoritmo Dijkstra(grafo  $G$ , vertice  $s$ )  $\rightarrow$  albero
  for each ( vertice  $u$  in  $G$  ) do  $D_{su} \leftarrow +\infty$ 
   $\hat{T} \leftarrow$  albero formato dal solo nodo  $s$ ;  $X \leftarrow \emptyset$ 
  CodaPriorita  $S$ 
   $D_{ss} \leftarrow 0$ 
   $S.insert(s, 0)$ 
  while ( not  $S.isEmpty()$  ) do
     $u \leftarrow S.deleteMin()$ ;  $X \leftarrow X \cup \{u\}$ 
    for each ( arco  $(u, v)$  in  $G$  ) do
      if ( $D_{sv} = +\infty$ ) then
         $S.insert(v, D_{su} + w(u, v))$ 
         $D_{sv} \leftarrow D_{su} + w(u, v)$ 
        rendi  $u$  padre di  $v$  in  $\hat{T}$ 
      else if ( $D_{su} + w(u, v) < D_{sv}$ ) then
         $S.decreaseKey(v, D_{sv} - D_{su} - w(u, v))$ 
         $D_{sv} \leftarrow D_{su} + w(u, v)$ 
        rendi  $u$  nuovo padre di  $v$  in  $\hat{T}$ 
  return  $\hat{T}$ 
```

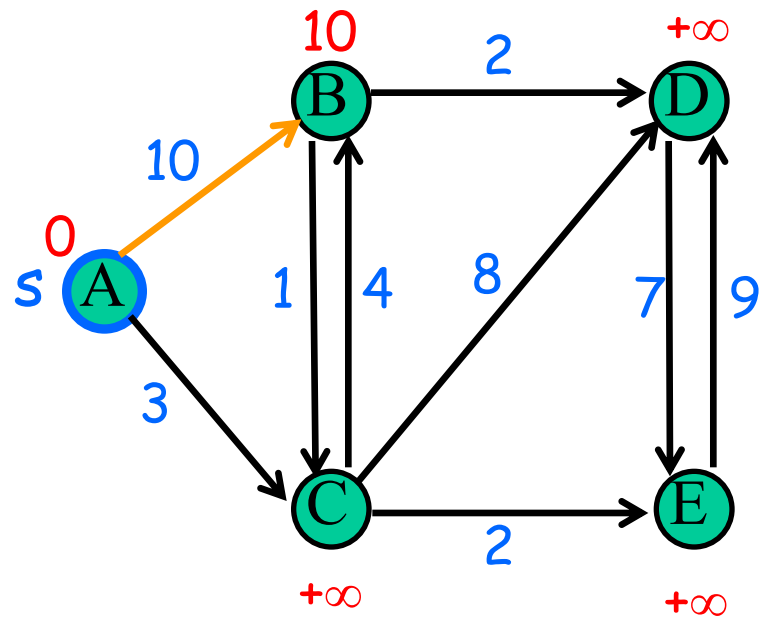
**Nota:**  $\hat{T}$  è un albero che contiene tutti i nodi in  $X$  più i nodi correntemente contenuti nella coda di priorità (nodi arancioni); è composto cioè dagli archi di  $T$  (albero dei cammini minimi ristretto ai nodi in  $X$ ) più gli archi arancioni (potenziali archi da aggiungere a  $T$ )

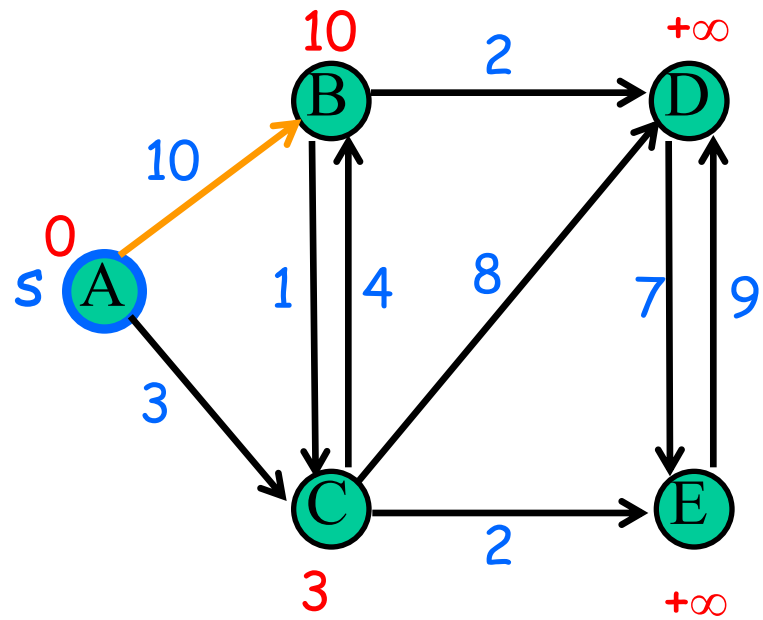


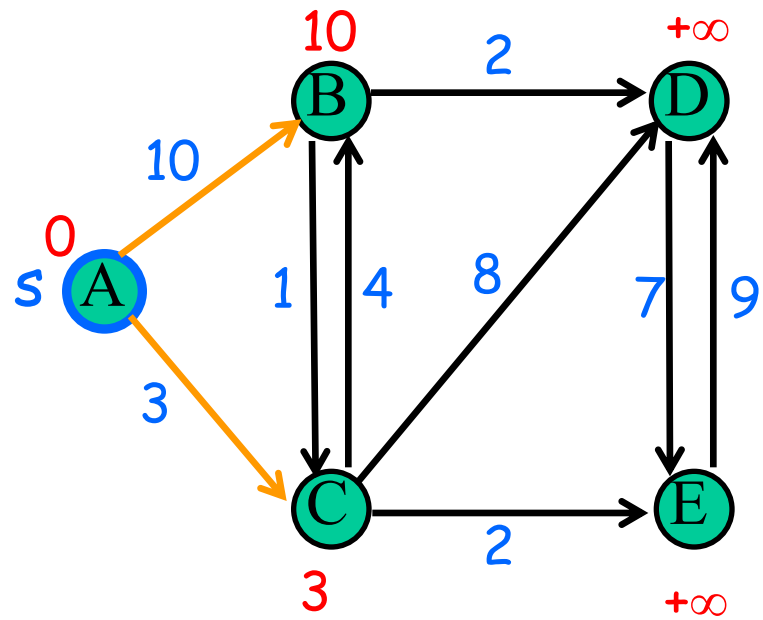


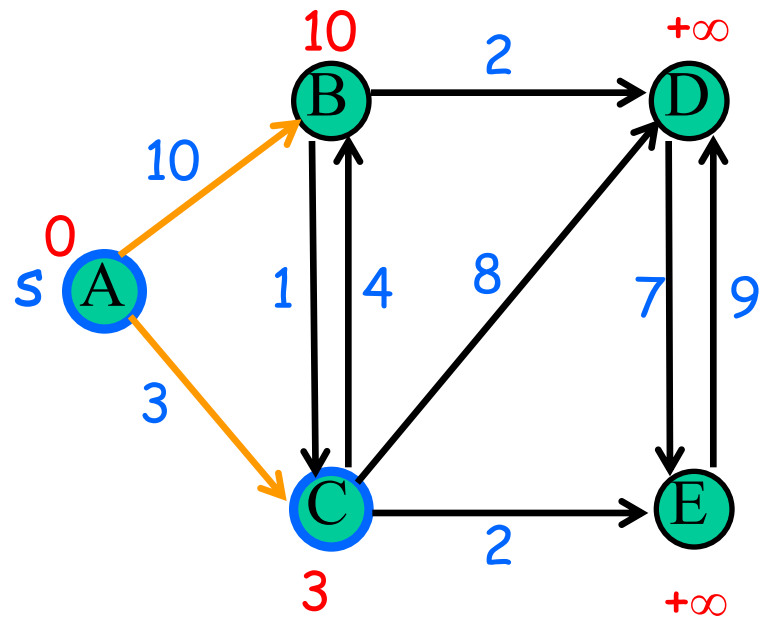


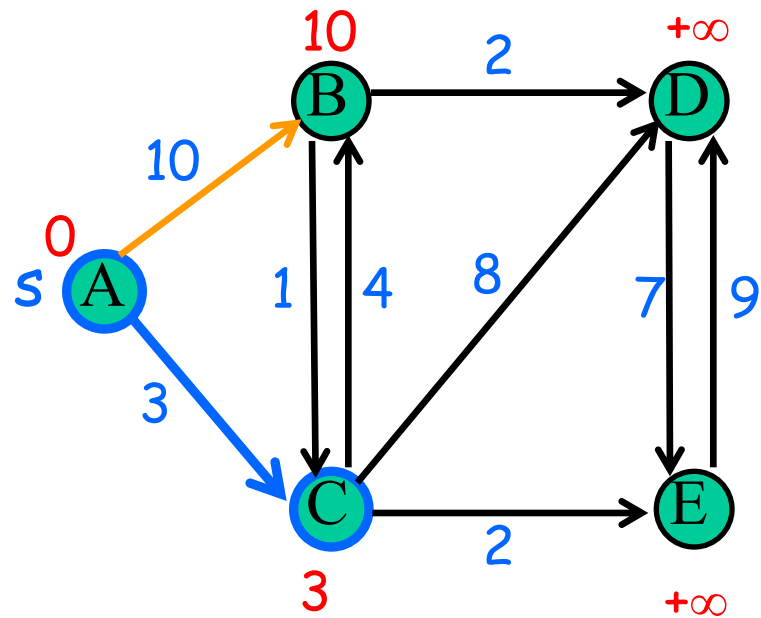


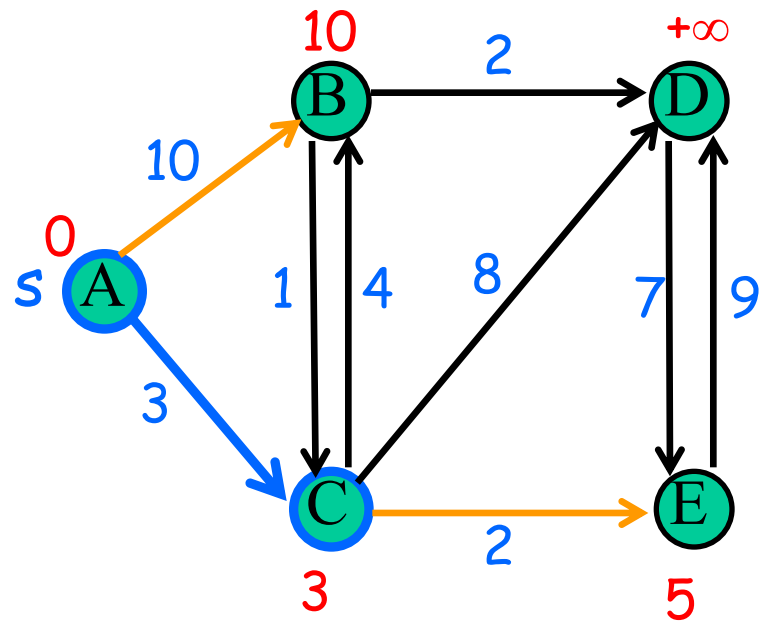


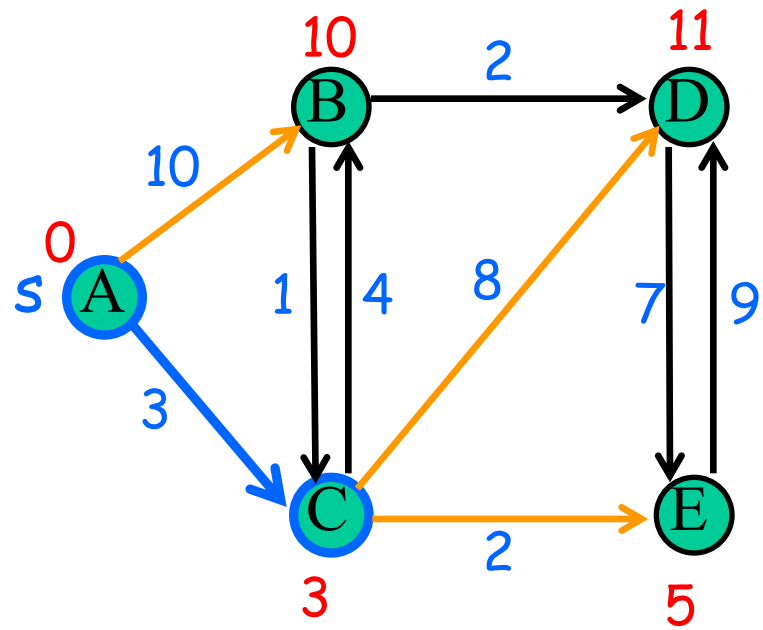




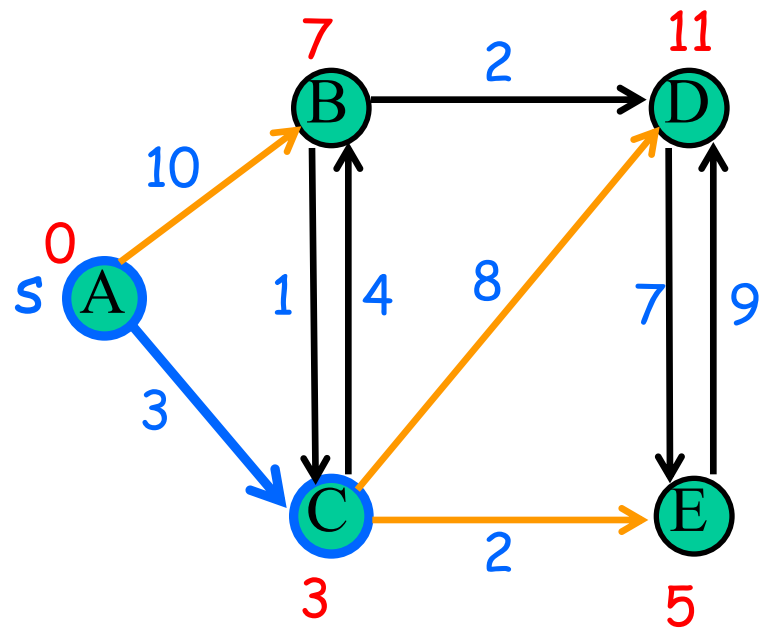


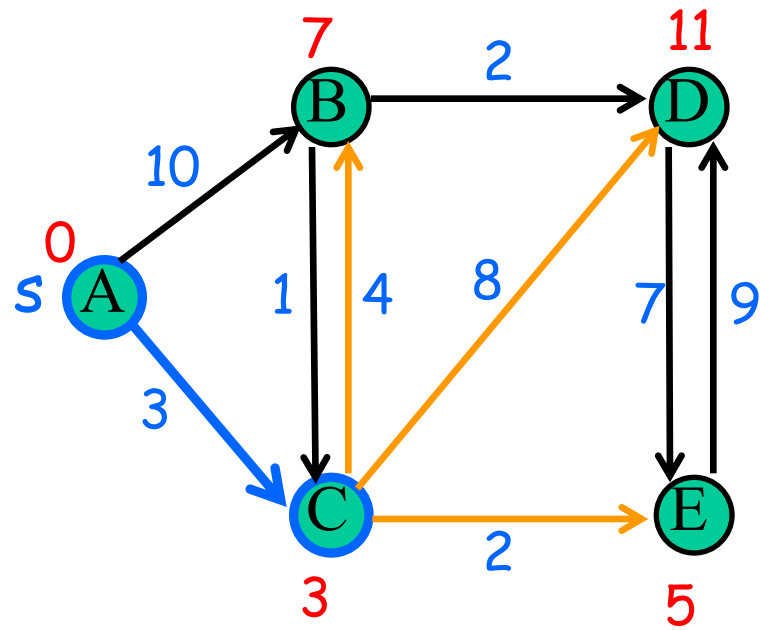


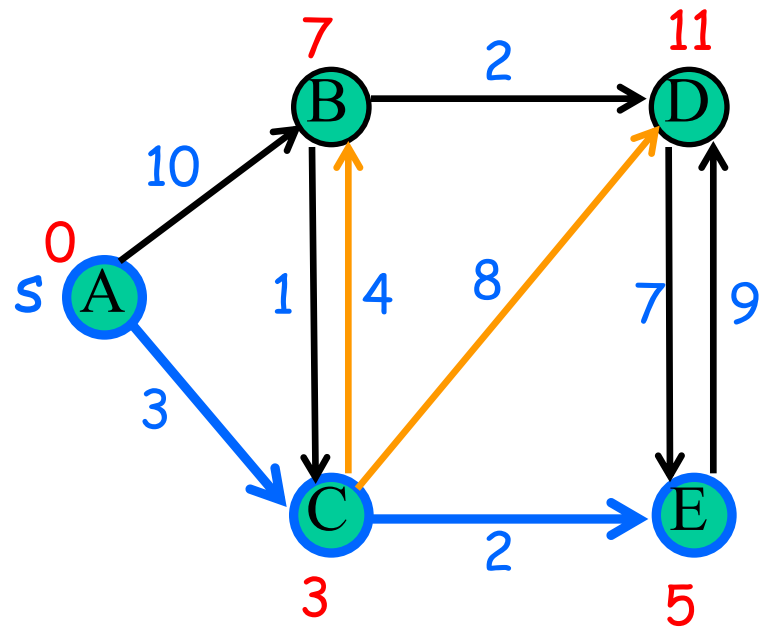


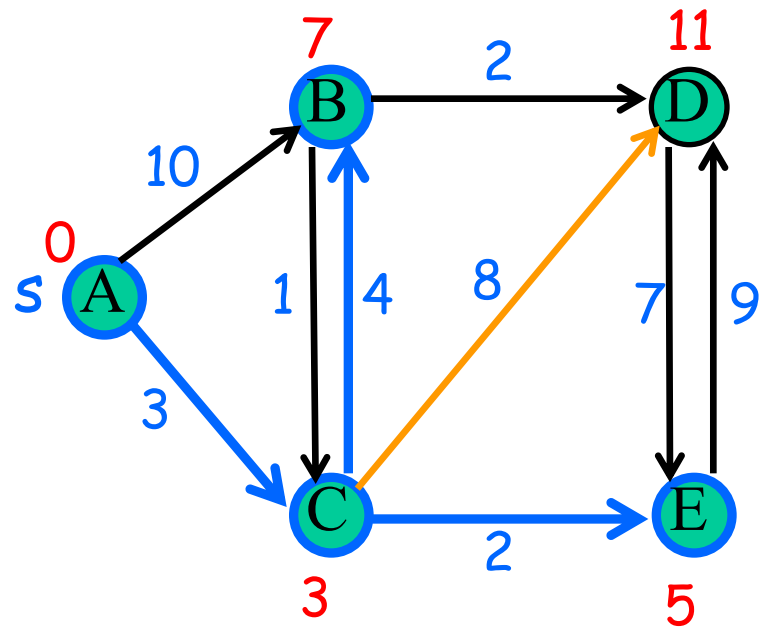


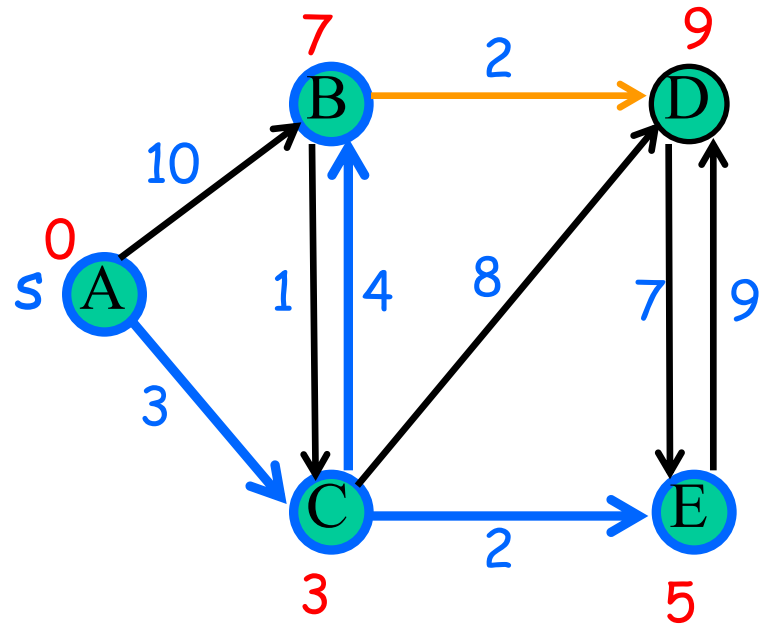


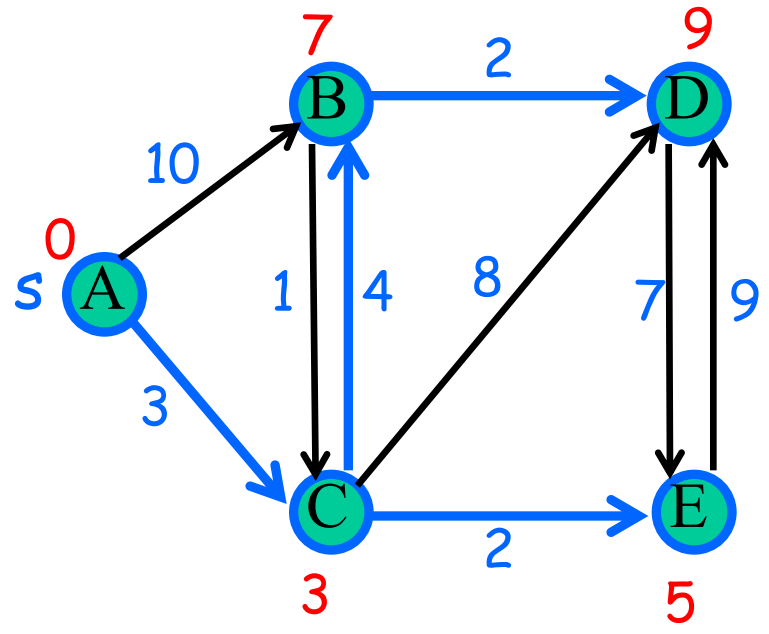


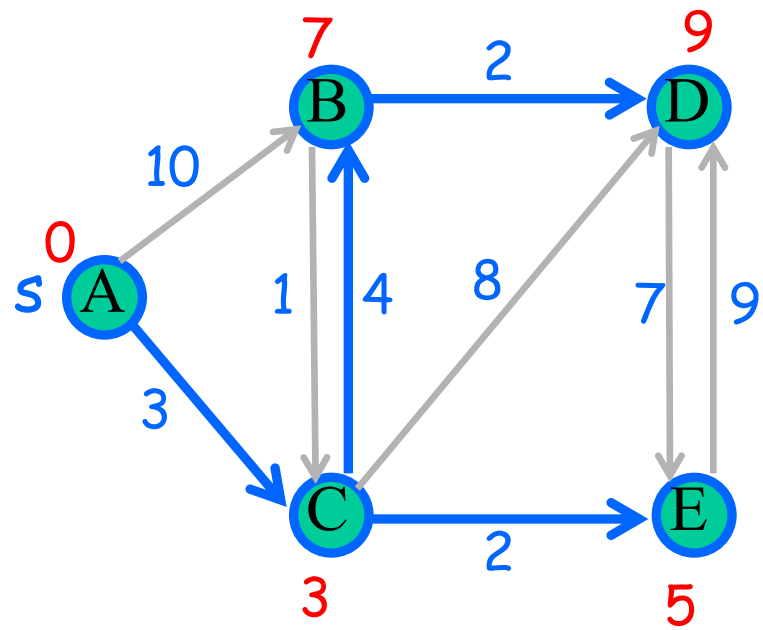














correttezza

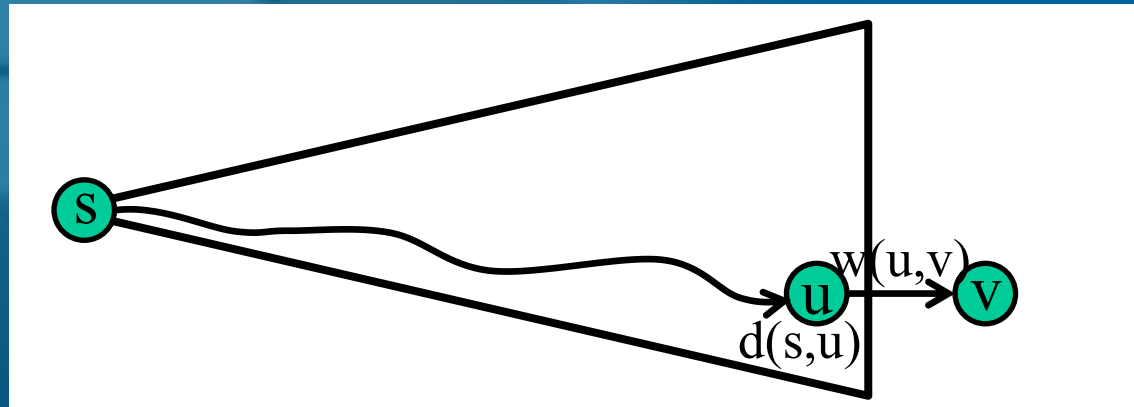


# Estendere l'albero dei cammini minimi

## Lemma di Dijkstra (1959)

Sia  $G=(V,E,w)$  (diretto o non diretto) con pesi non negativi, e sia  $T$  un sottoalbero dell'albero dei cammini minimi radicato in  $s$  che include  $s$  ma non include tutti i vertici raggiungibili da  $s$ . Sia  $(u,v)$  l'arco che **minimizza** la quantità  $d(s,t) + w(t,z)$ , per ogni  $t \in T$  e  $z \notin T$ . Allora,  $(u,v)$  appartiene a un cammino minimo da  $s$  a  $v$ .

**Dim.:** Supponiamo per assurdo che  $(u,v)$  non appartenga ad un cammino minimo da  $s$  a  $v$ , e quindi che  $d(s,v) < d(s,u) + w(u,v)$ . Allora,  $d(s,v)$  è la lunghezza di un cammino minimo da  $s$  a  $v$  che non passa per  $(u,v)$ .

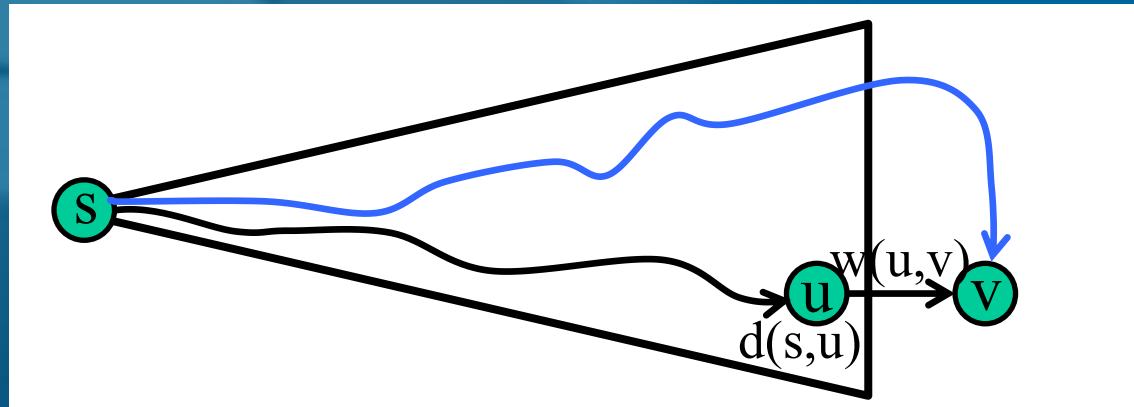


# Estendere l'albero dei cammini minimi

## Lemma di Dijkstra (1959)

Sia  $G=(V,E,w)$  (diretto o non diretto) con pesi non negativi, e sia  $T$  un sottoalbero dell'albero dei cammini minimi radicato in  $s$  che include  $s$  ma non include tutti i vertici raggiungibili da  $s$ . Sia  $(u,v)$  l'arco che **minimizza** la quantità  $d(s,t) + w(t,z)$ , per ogni  $t \in T$  e  $z \notin T$ . Allora,  $(u,v)$  appartiene a un cammino minimo da  $s$  a  $v$ .

**Dim.:** Supponiamo per assurdo che  $(u,v)$  non appartenga ad un cammino minimo da  $s$  a  $v$ , e quindi che  $d(s,v) < d(s,u) + w(u,v)$ . Allora,  $d(s,v)$  è la lunghezza di un cammino minimo da  $s$  a  $v$  che non passa per  $(u,v)$ .

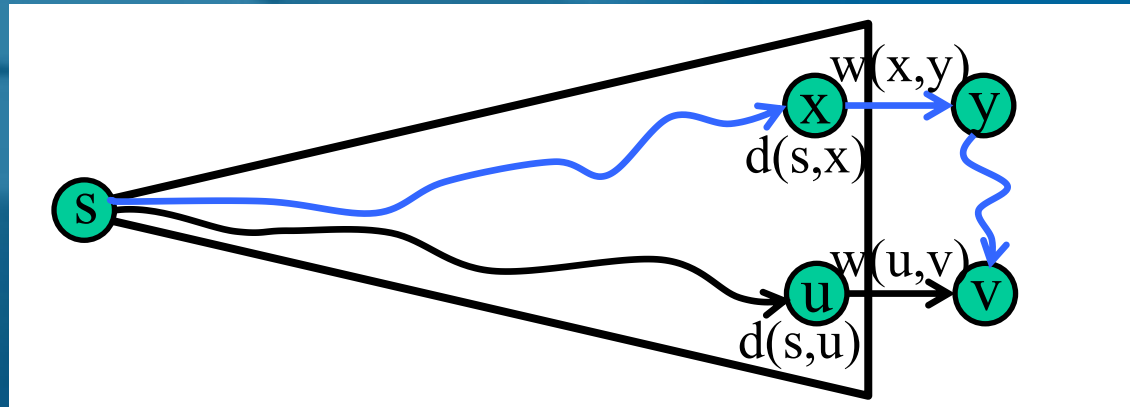


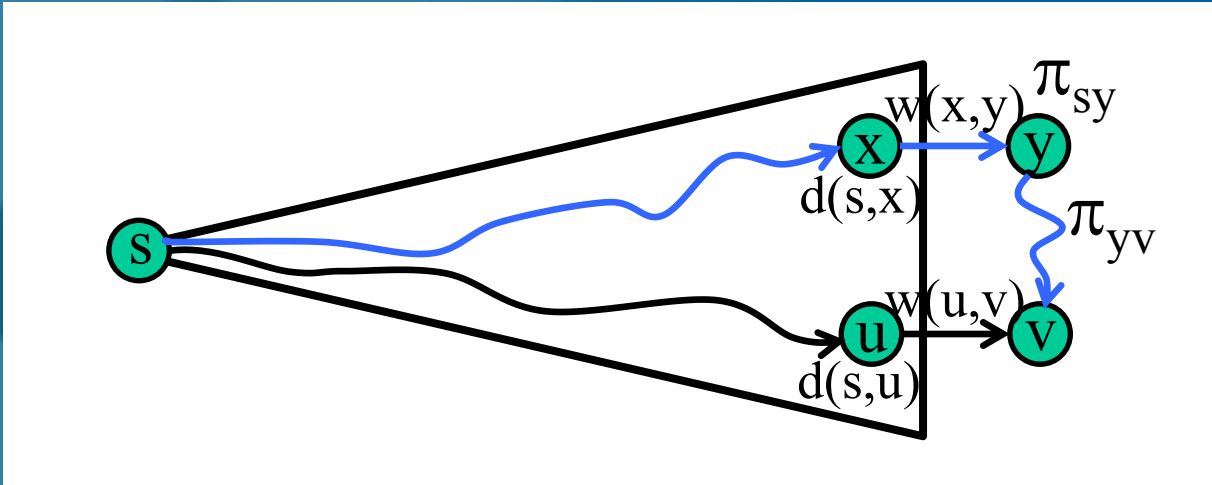
# Estendere l'albero dei cammini minimi

## Lemma di Dijkstra (1959)

Sia  $G=(V,E,w)$  (diretto o non diretto) con pesi non negativi, e sia  $T$  un sottoalbero dell'albero dei cammini minimi radicato in  $s$  che include  $s$  ma non include tutti i vertici raggiungibili da  $s$ . Sia  $(u,v)$  l'arco che **minimizza** la quantità  $d(s,t) + w(t,z)$ , per ogni  $t \in T$  e  $z \notin T$ . Allora,  $(u,v)$  appartiene a un cammino minimo da  $s$  a  $v$ .

**Dim.:** Supponiamo per assurdo che  $(u,v)$  non appartenga ad un cammino minimo da  $s$  a  $v$ , e quindi che  $d(s,v) < d(s,u) + w(u,v)$ . Allora,  $d(s,v)$  è la lunghezza di un cammino minimo da  $s$  a  $v$  che non passa per  $(u,v)$ . Tale cammino, per uscire da  $T$ , deve allora passare per un qualche arco  $(x,y) \neq (u,v)$ , con  $x \in T$  e  $y \notin T$ . Sia quindi  $\pi_{sv} = \langle s, \dots, x, y, \dots, v \rangle$ .





Per la **minimalità dei sottocammini di un cammino minimo**:

$$w(\pi_{sv}) = w(\pi_{sy}) + w(\pi_{yv}) = d(s,x) + w(x,y) + w(\pi_{yv}).$$

**Quindi:**  $w(\pi_{sv}) < d(s,u) + w(u,v)$

$$d(s,x) + w(x,y) + \underbrace{w(\pi_{yv})}_{\geq 0} < d(s,u) + w(u,v)$$

$$d(s,x) + w(x,y) < d(s,u) + w(u,v)$$

ma  $(u,v)$  minimizza  $d(s,t) + w(t,z)$  per ogni  $t \in T$  e  $z \notin T$ , quindi:

$$d(s,x) + w(x,y) \geq d(s,u) + w(u,v)$$

**assurdo!**





analisi della complessità

```

algoritmo Dijkstra(grafo  $G$ , vertice  $s$ )  $\rightarrow$  albero
  for each ( vertice  $u$  in  $G$  ) do  $D_{su} \leftarrow +\infty$ 
   $\hat{T} \leftarrow$  albero formato dal solo nodo  $s$ ;  $X \leftarrow \emptyset$ 
  CodaPriorita  $S$ 
   $D_{ss} \leftarrow 0$ 
   $S.insert(s, 0)$ 
  while ( not  $S.isEmpty()$  ) do
     $u \leftarrow S.deleteMin()$ ;  $X \leftarrow X \cup \{u\}$ 
    for each ( arco  $(u, v)$  in  $G$  ) do
      if ( $D_{sv} = +\infty$ ) then
         $S.insert(v, D_{su} + w(u, v))$ 
         $D_{sv} \leftarrow D_{su} + w(u, v)$ 
        rendi  $u$  padre di  $v$  in  $\hat{T}$ 
      else if ( $D_{su} + w(u, v) < D_{sv}$ ) then
         $S.decreaseKey(v, D_{sv} - D_{su} - w(u, v))$ 
         $D_{sv} \leftarrow D_{su} + w(u, v)$ 
        rendi  $u$  nuovo padre di  $v$  in  $\hat{T}$ 
  return  $\hat{T}$ 

```

se si escludono le  
operazioni sulla  
coda con priorità:

tempo  
 $O(m+n)$

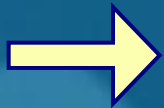
quanto costano le  
operazioni sulla  
coda con priorità?

# Tempo di esecuzione: implementazioni elementari

Supponendo che il grafo  $G$  sia rappresentato tramite liste di adiacenza, e supponendo che tutti i nodi siano connessi ad  $s$ , avremo  $n$  `insert`,  $n$  `deleteMin` e al più  $m$  `decreaseKey` nella coda di priorità, al costo di:

	Insert	DelMin	DecKey
Array non ord.	$O(1)$	$O(n)$	$O(1)$
Array ordinato	$O(n)$	$O(1)$	$O(n)$
Lista non ord.	$O(1)$	$O(n)$	$O(1)$
Lista ordinata	$O(n)$	$O(1)$	$O(n)$

- $n \cdot O(1) + n \cdot O(n) + O(m) \cdot O(1) = O(n^2)$  con array non ordinati
- $n \cdot O(n) + n \cdot O(1) + O(m) \cdot O(n) = O(m \cdot n)$  con array ordinati
- $n \cdot O(1) + n \cdot O(n) + O(m) \cdot O(1) = O(n^2)$  con liste non ordinate
- $n \cdot O(n) + n \cdot O(1) + O(m) \cdot O(n) = O(m \cdot n)$  con liste ordinate





# Tempo di esecuzione: implementazioni efficienti

Supponendo che il grafo  $G$  sia rappresentato tramite liste di adiacenza, e supponendo che tutti i nodi siano connessi ad  $s$ , avremo  $n$  `insert`,  $n$  `deleteMin` e al più  $m$  `decreaseKey` nella coda di priorità, al costo di:

	Insert	DelMin	DecKey
Heap binario	$O(\log n)$	$O(\log n)$	$O(\log n)$
Heap Binom.	$O(\log n)$	$O(\log n)$	$O(\log n)$
Heap Fibon.	$O(1)$	$O(\log n)^*$ (ammortizzata)	$O(1)^*$ (ammortizzata)

- $n \cdot O(\log n) + n \cdot O(\log n) + O(m) \cdot O(\log n) = O(m \cdot \log n)$  utilizzando heap binari o binomiali

- $n \cdot O(1) + n \cdot O(\log n)^* + O(m) \cdot O(1)^* = O(m + n \cdot \log n)$  utilizzando heap di Fibonacci

soluzione migliore: mai peggiore,  
a volte meglio delle altre



# Tempo di esecuzione: implementazioni efficienti

Supponendo che il grafo  $G$  sia rappresentato tramite liste di adiacenza, e supponendo che tutti i nodi siano connessi ad  $s$ , avremo  $n$  insert,  $n$  deleteMin e al più  $m$  decreaseKey nella coda di priorità, al costo di:

	Insert	DelMin	DecKey
Heap binario	$O(\log n)$	$O(\log n)$	$O(\log n)$
Heap Binom.	$O(\log n)$	$O(\log n)$	$O(\log n)$
Heap Fibon.	$O(1)$	$O(\log n)^*$ (ammortizzata)	$O(1)^*$ (ammortizzata)

- $n \cdot O(\log n)$  utilizzando
- $n \cdot O(1)$  heap di

tempo  
 $O(m+n \log n)$

$O(\log n)$   
 $n$ ) utilizzando

soluzione migliore e mai peggiore, a volte meglio delle altre

# Osservazione sulla `decreaseKey`

- Ricordiamo che le complessità computazionali esposte per la `decreaseKey` sono valide supponendo di avere un **puntatore diretto** all'elemento su cui eseguire l'operazione. Come possiamo garantire tale condizione?
- Semplicemente mantenendo un puntatore tra il nodo **v** nell'array dei nodi della lista di adiacenza del grafo e l'elemento nella coda di priorità associato al nodo **v**; tale puntatore viene inizializzato nella fase di inserimento di quest'ultimo all'interno della coda.