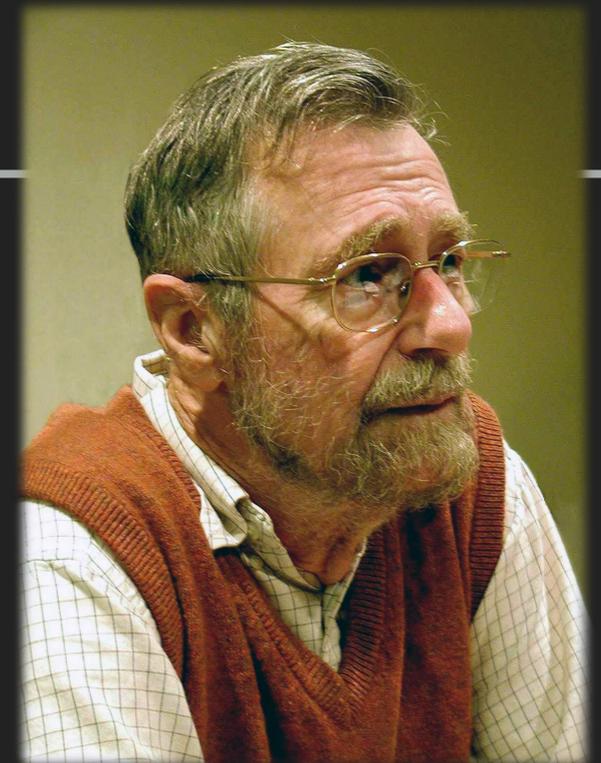


ALGORITMO DI BELLMAN-FORD  
ALGORITMO DI DIJKSTRA

---

# ALGORTIMI E STRUTTURE DATI



Dijkstra

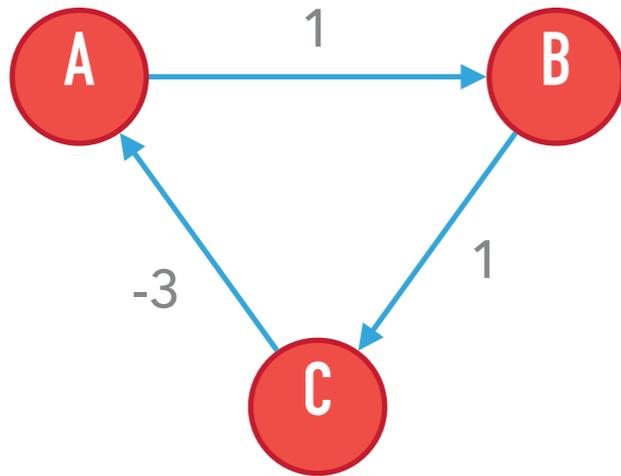
# IL PROBLEMA CHE VOGLIAMO RISOLVERE

- ▶ Il peso di un percorso  $p = (v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$  è la somma dei pesi di tutti gli archi:  $w(p) = \sum_{i=1}^n w_{v_i, v_j}$
- ▶ Assumiamo di volere solo percorsi **semplici**, ovvero privi di cicli (ogni nodo al più una volta in ogni percorso)
- ▶ Dato un grafo si vuole trovare il percorso **di peso minore** tra un vertice di partenza  $s$  e tutti gli altri vertici (o verso un vertice di destinazione  $d$ )

# IL PROBLEMA CHE VOGLIAMO RISOLVERE

- ▶ Il grafo può essere orientato o non orientato
- ▶ A seconda delle assunzioni che facciamo sui pesi possiamo utilizzare algoritmi diversi:
  - ▶ I pesi possono essere negativi:  
algoritmo di Bellman-Ford
  - ▶ I pesi sono solo non negativi:  
algoritmo di Dijkstra

# IL CASO DA EVITARE: CICLI NEGATIVI



Quale è il percorso meno costoso per andare da A a sé stesso?

Intuitivamente diremmo "il percorso vuoto" che ha costo zero

Ma il percorso (A,B),(B,C),(C,A) ha peso totale  $-1$

Quindi non esiste un percorso più corto, dato che possiamo sempre accorciare facendo "un altro giro"

**Assumiamo assenza di cicli di peso negativo**

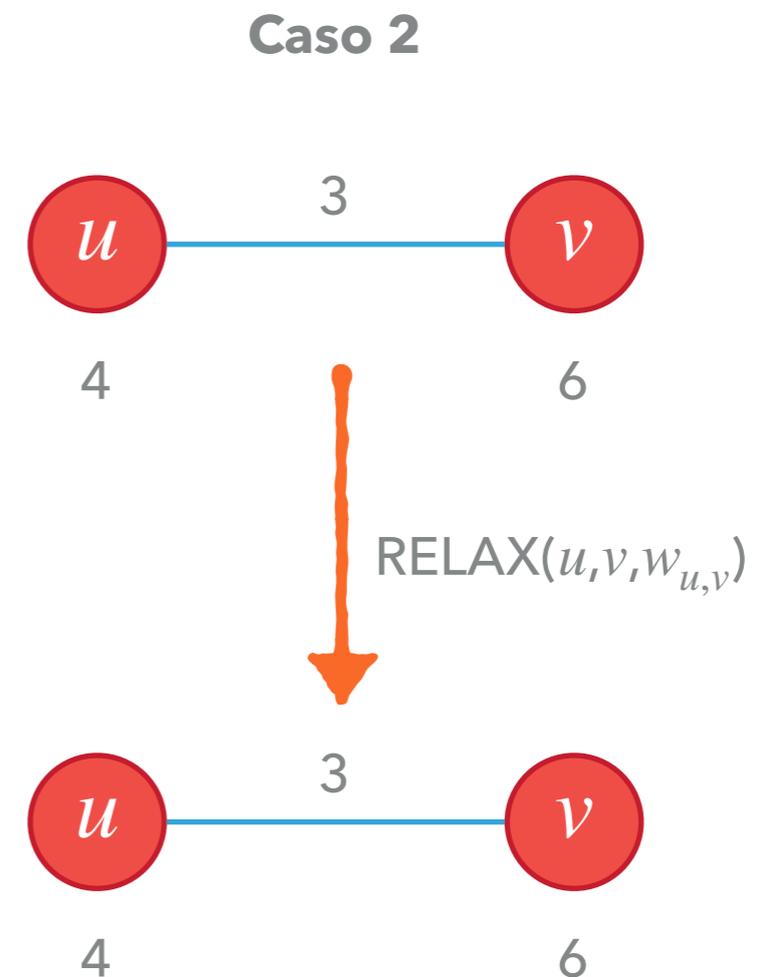
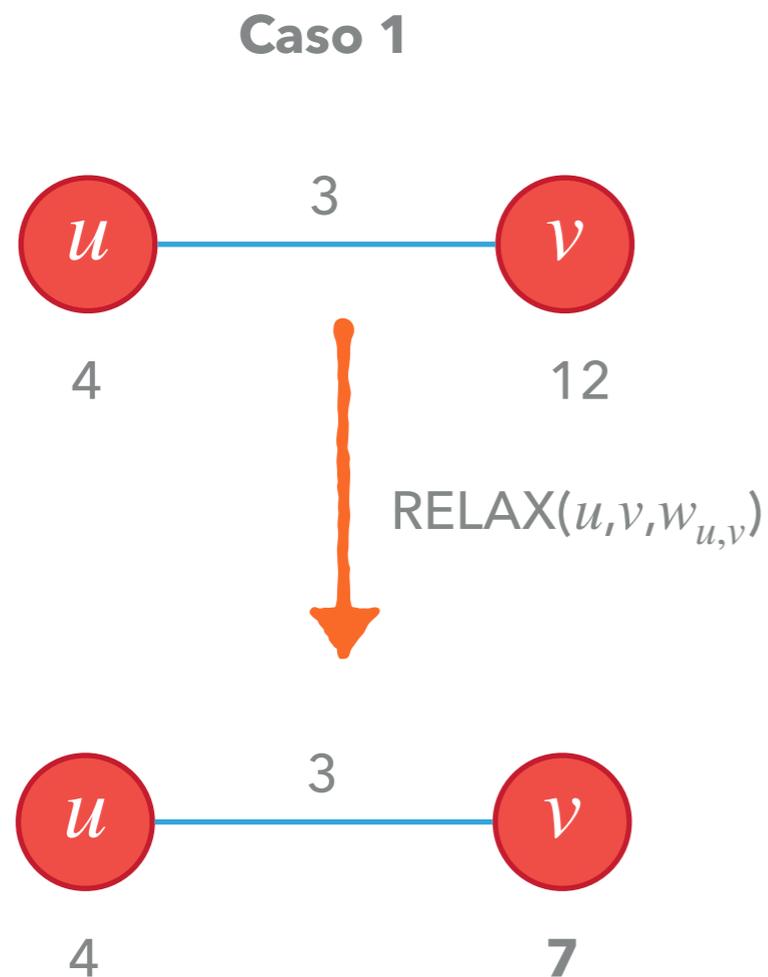
# OPERAZIONE DI RILASSAMENTO/RELAX DEGLI ARCHI

- ▶  $\delta(u, v) = \min\{w(p) \mid p : u \longrightarrow v \text{ semplice}\}$
- ▶ Lemma: Sia  $(u, v) \in E$ , allora  $\delta(s, v) \leq \delta(s, u) + w_{u,v}$
- ▶ Supponiamo di avere una stima della del peso del percorso da  $s$  ad un nodo  $u$ , indicata con  $\text{distanza}[u]$ , e ad un nodo  $v$ , indicata con  $\text{distanza}[v]$  e che esista l'arco  $(u, v)$  di peso  $w_{u,v}$
- ▶ L'operazione di rilassamento consiste nel vedere se possiamo usare la nostra stima per  $u$  per migliorare la stima della distanza per  $v$

# OPERAZIONE DI RILASSAMENTO/RELAX DEGLI ARCHI

- ▶ Se  $\text{distanza}[u] + w_{u,v} < \text{distanza}[v]$  allora possiamo andare da  $s$  a  $v$  passando per  $u$  con un costo minore
- ▶ Se quello è il caso aggiorniamo la nostra stima della distanza:  $\text{distanza}[v] = \text{distanza}[u] + w_{u,v}$
- ▶ Altrimenti non la modifichiamo
- ▶ Indichiamo questa operazione come  $\text{RELAX}(u, v, w_{u,v})$

# OPERAZIONE DI RILASSAMENTO/RELAX DEGLI ARCHI



```
if distanza[u] + w < distanza[v]
    distanza[v] = distanza[u] + w
    predecessore[v] = u
```

# OPERAZIONE DI RILASSAMENTO/RELAX DEGLI ARCHI

- ▶ Assumiamo che  $(u,v)$  appartenga ad un cammino minimo da  $s$  a  $v$ .
- ▶ Assumiamo che prima di RELAX, valga  $\text{distanza}[u] = \delta(s, u)$
- ▶ Allora dopo  $\text{RELAX}(u, v, w_{u,v})$ , varrà  $\text{distanza}[v] = \delta(s, v)$
- ▶ In generale, si dimostra facilmente che se  $d[v] \geq \delta(s, v)$  prima di relax, allora  $d[v] \geq \delta(s, v)$  anche dopo relax.

### IDEA GENERALE: ALGORITMO DI BELLMAN-FORD

- ▶ Il percorso più lungo in un grafo di  $|V|$  nodi è composto da al più  $|V| - 1$  archi (altrimenti siamo in un ciclo)
- ▶ Se effettuiamo  $|V| - 1$  operazioni di rilassamento per ogni arco allora abbiamo che le nostre stime delle distanze non si possono più modificare
- ▶ Abbiamo quindi trovato il percorso di peso/costo minimo da  $s$  a ogni altro nodo

## PSEUDOCODICE

Parametri: grafo  $G$ , nodo sorgente  $s$

# inizialmente impostiamo distanza e predecessore di tutti i nodi

for all  $v \in V$ :

    distanza[ $v$ ] =  $+\infty$

    predecessore[ $v$ ] = None

# e poi impostiamo il nodo  $s$  come sorgente a distanza 0

distanza[ $s$ ] = 0

for  $i$  in range(0,  $|V| - 1$ )

    for all  $(u, v) \in E$  # effettuiamo il rilassamento di tutti gli archi

        RELAX( $u, v, w_{u,v}$ )

**SE DOPO  $|V| - 1$  ITERAZIONI FOSSE ANCORA POSSIBILE AGGIORNARE LE DISTANZE SIGNIFICHEREBBE CHE IL GRAFO CONTIENE UN CICLO NEGATIVO: POSSIAMO USARE L'ALGORITMO PER IDENTIFICARE QUESTO TIPO DI GRAFI**

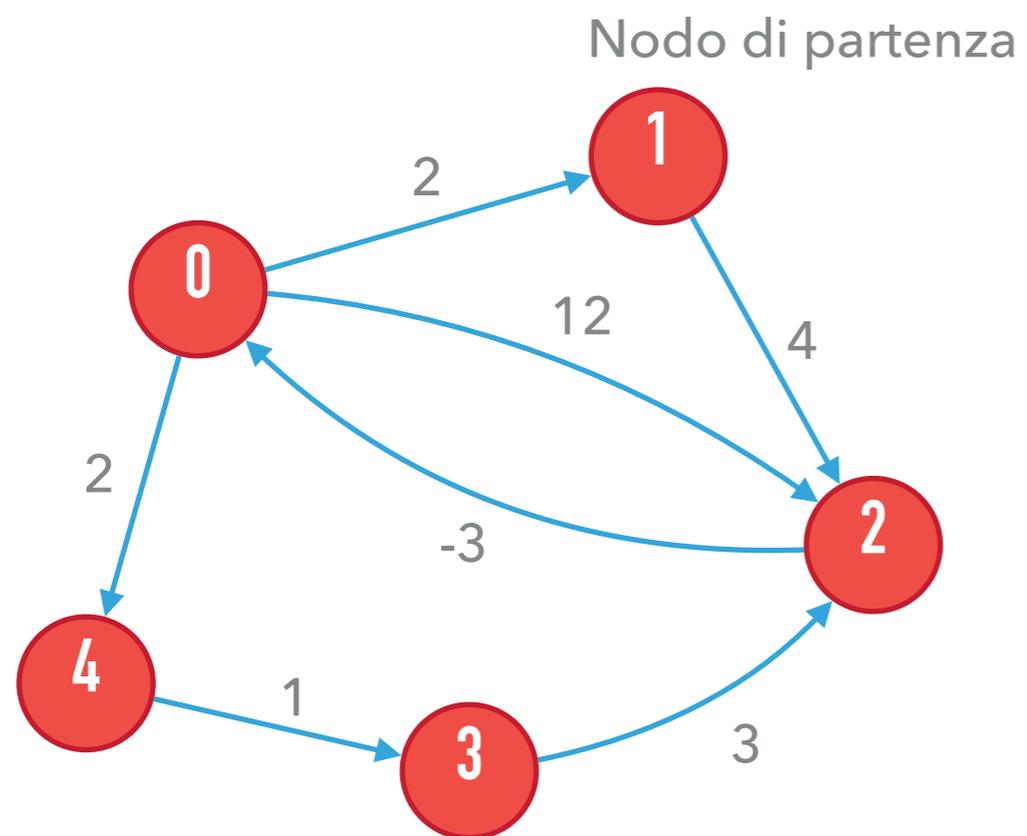
### CICLI DI PESO NEGATIVO

- ▶ Sia  $c : v_0, v_1, \dots, v_k$  un ciclo semplice ( $v_0 = v_k$ , tutti gli altri vertici diversi).
- ▶  $c$  ha peso negativo ( $w(c) < 0$ ) solo se a fine algoritmo esiste  $(u, v) \in E$  tale che  $\text{distanza}[v] > \text{distanza}[u] + w_{u,v}$
- ▶ Per assurdo, se  $c : v_0, v_1, \dots, v_k$  ha peso negativo ma  $\text{distanza}[v_i] \leq \text{distanza}[v_{i-1}] + w_{i-1,i}$  per ogni  $i$ , allora

- ▶ 
$$\sum_{i=1}^k \text{distanza}[v_i] \leq \sum_{i=1}^k \text{distanza}[v_{i-1}] + \sum_{i=1}^k w_{i-1,i} \text{ e quindi}$$

- ▶ 
$$\sum_{i=1}^k w_{i-1,i} \geq 0, \text{ assurdo.}$$

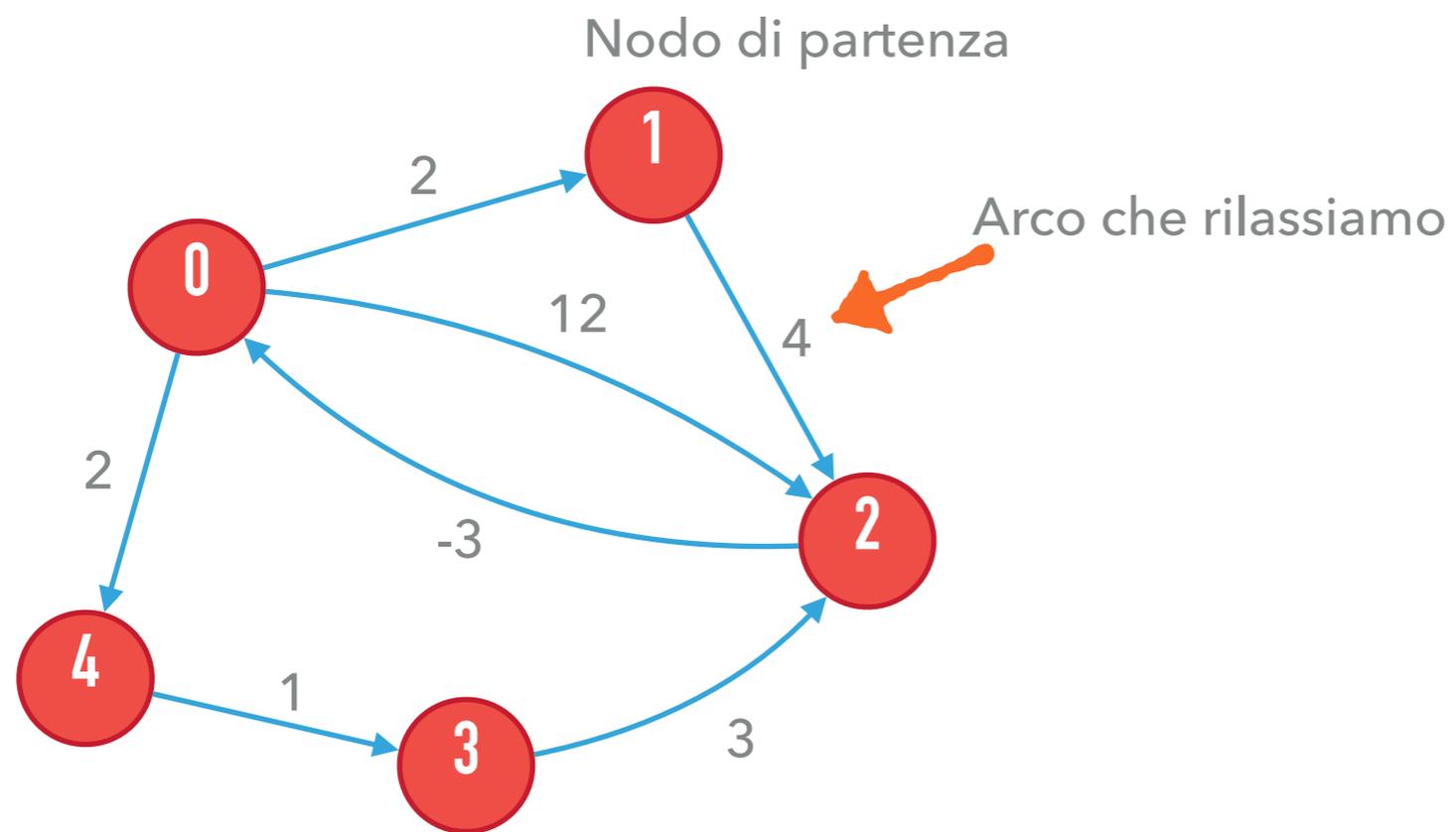
## ESEMPIO DI ESECUZIONE



Per l'esempio usiamo un grafo orientato, altrimenti non potremmo mai avere pesi negativi (avremmo un ciclo negativo)

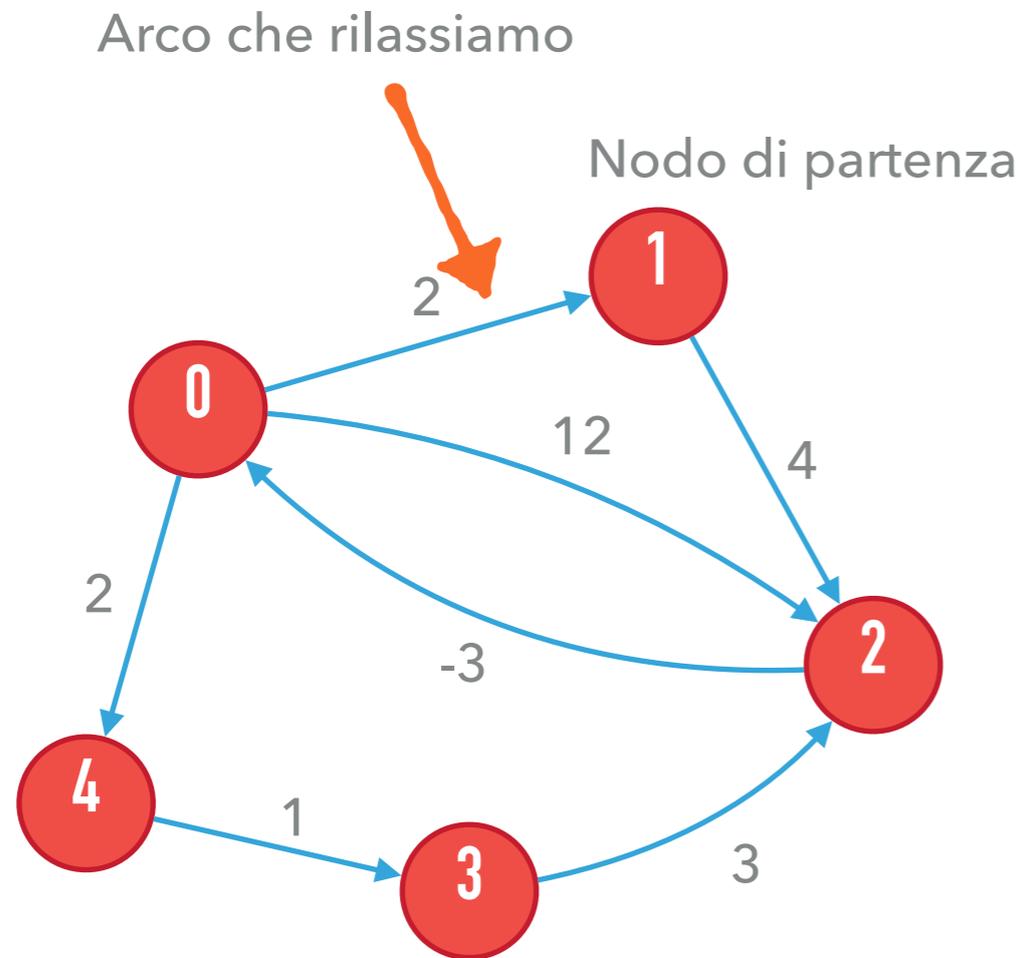
Vertice	Peso
0	$\infty$
1	0
2	$\infty$
3	$\infty$
4	$\infty$

## ESEMPIO DI ESECUZIONE



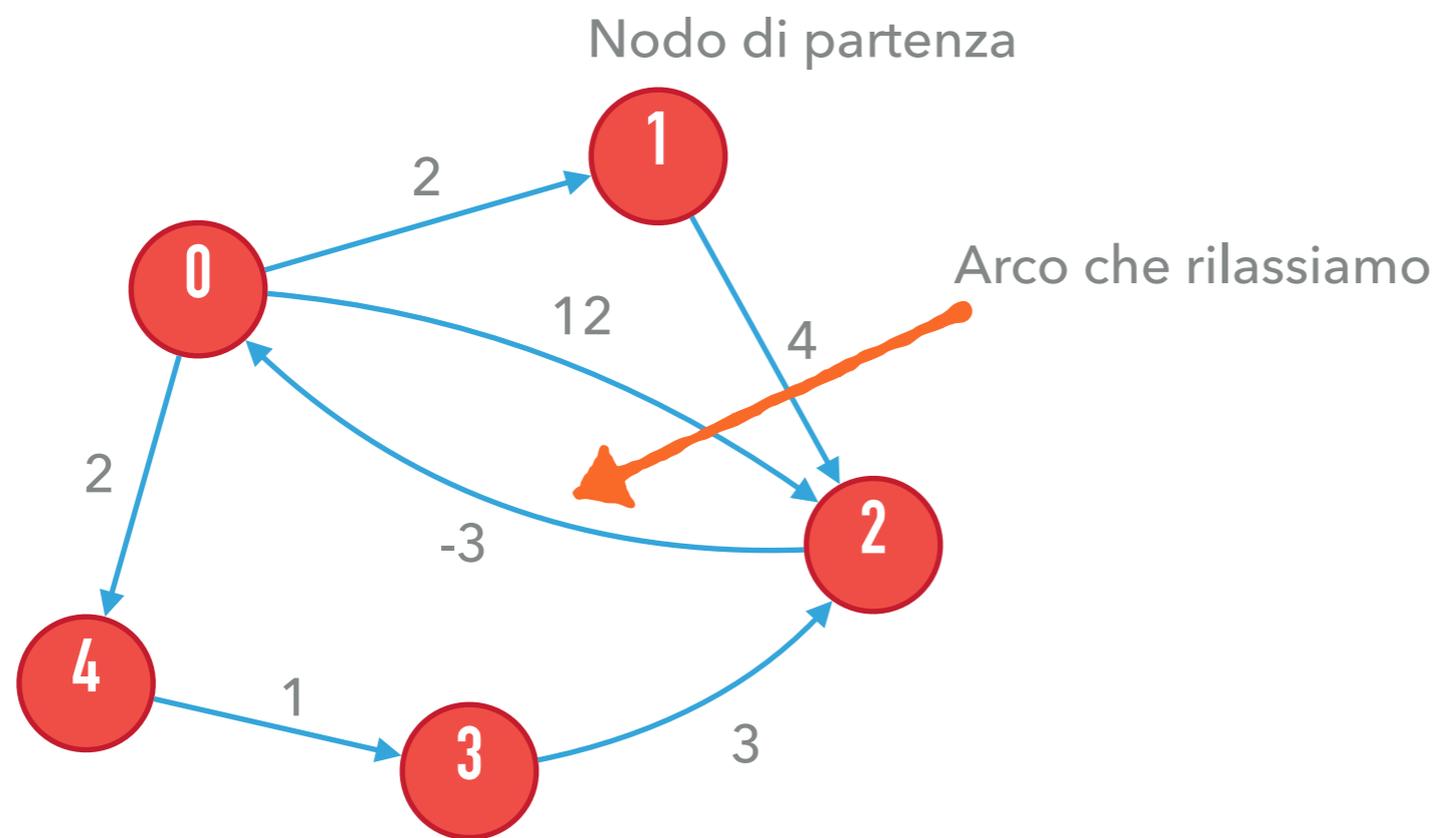
Vertice	Peso
0	$\infty$
1	0
2	4
3	$\infty$
4	$\infty$

## ESEMPIO DI ESECUZIONE



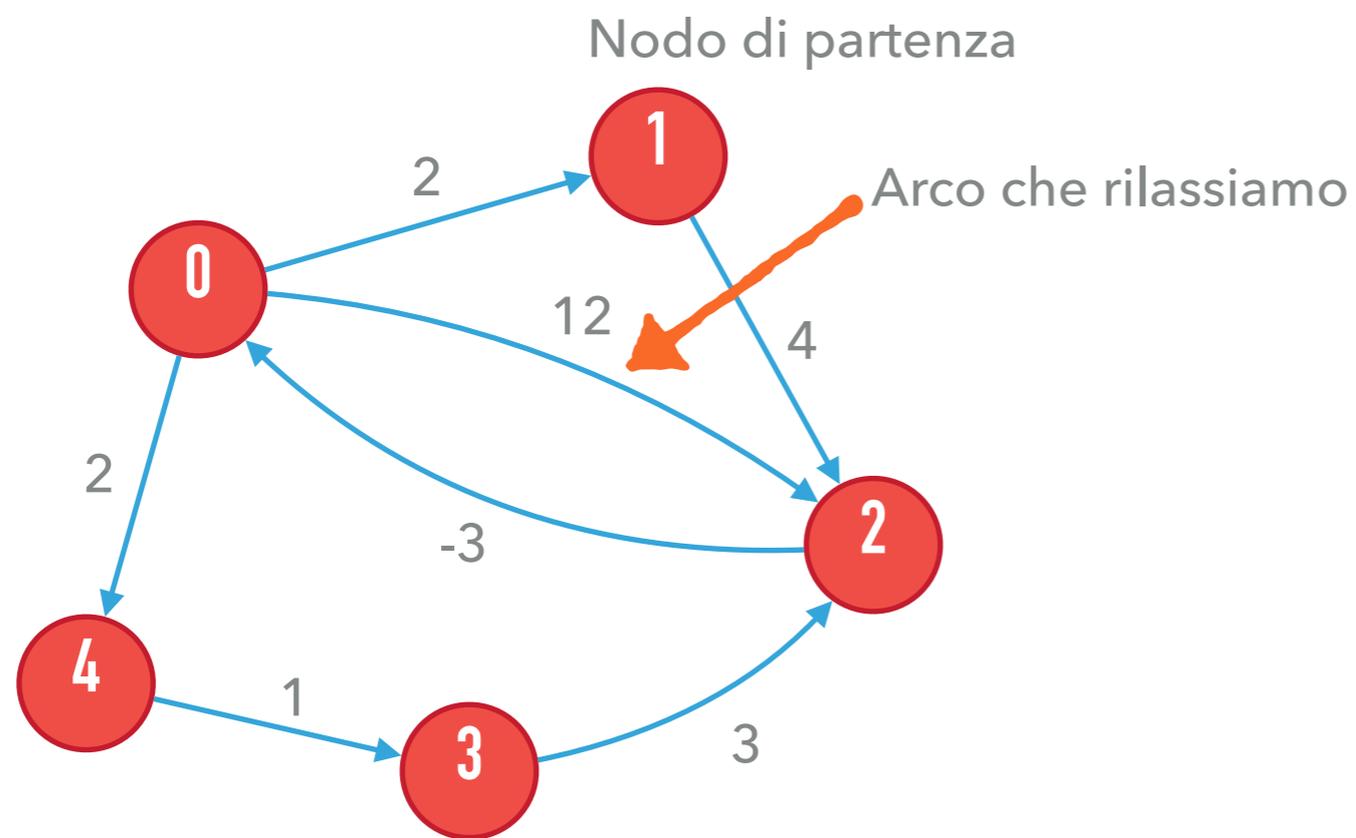
Vertice	Peso
0	$\infty$
1	0
2	4
3	$\infty$
4	$\infty$

## ESEMPIO DI ESECUZIONE



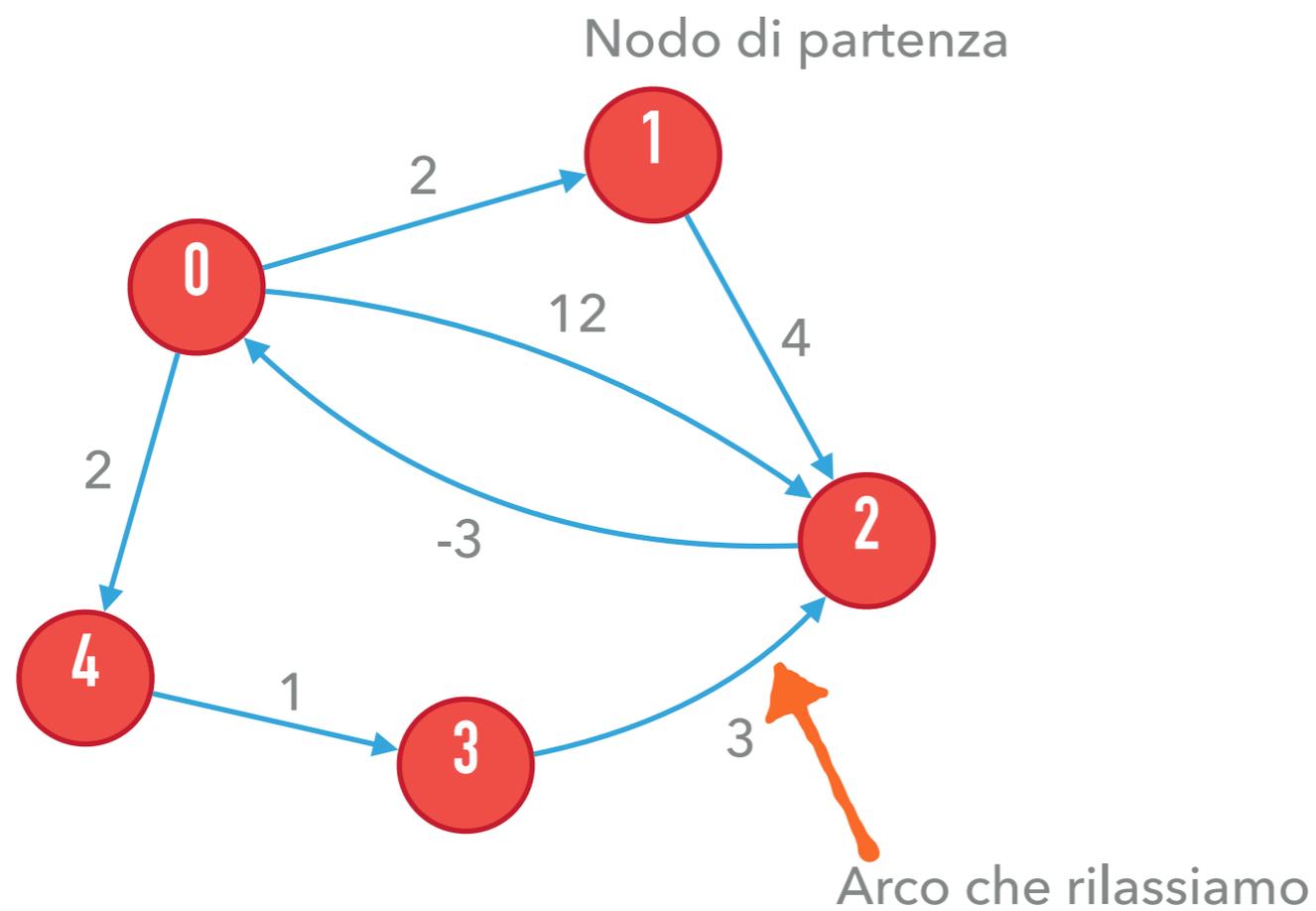
Vertice	Peso
0	1
1	0
2	4
3	$\infty$
4	$\infty$

## ESEMPIO DI ESECUZIONE



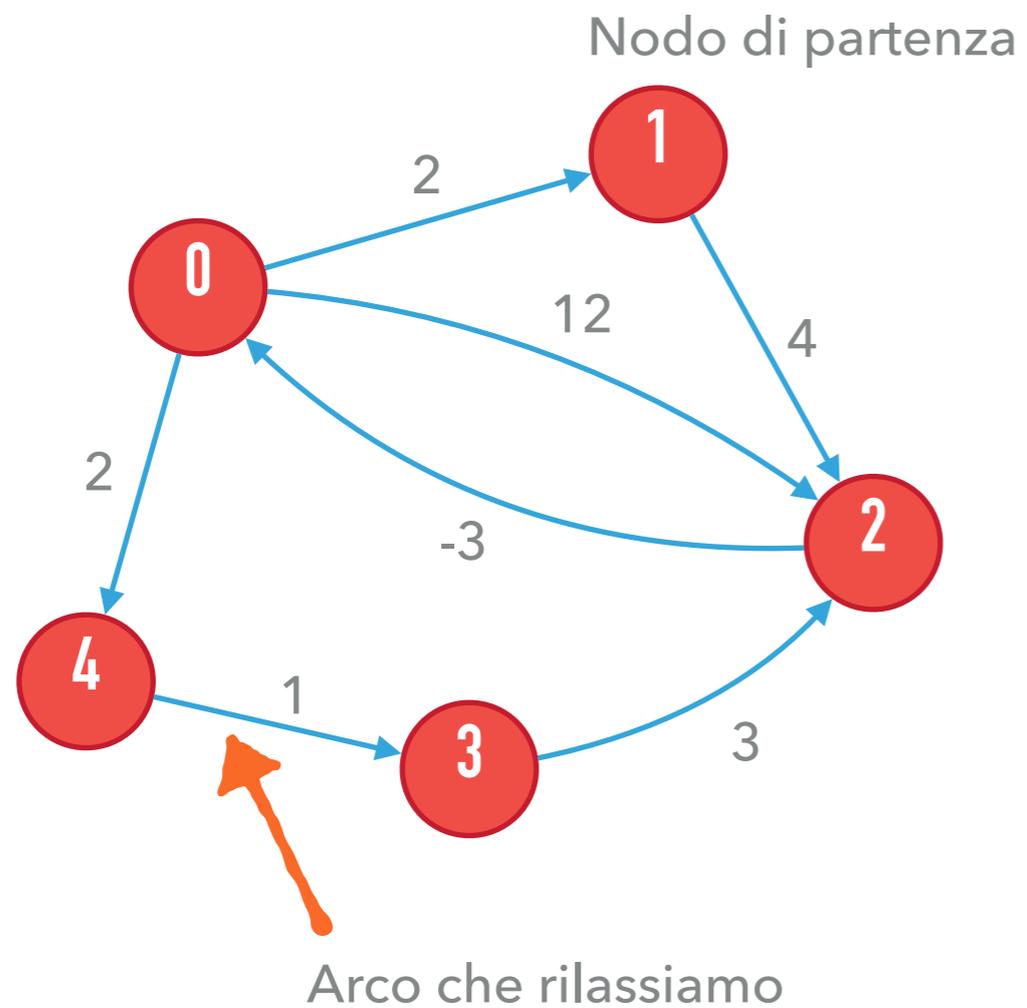
Vertice	Peso
0	1
1	0
2	4
3	$\infty$
4	$\infty$

## ESEMPIO DI ESECUZIONE



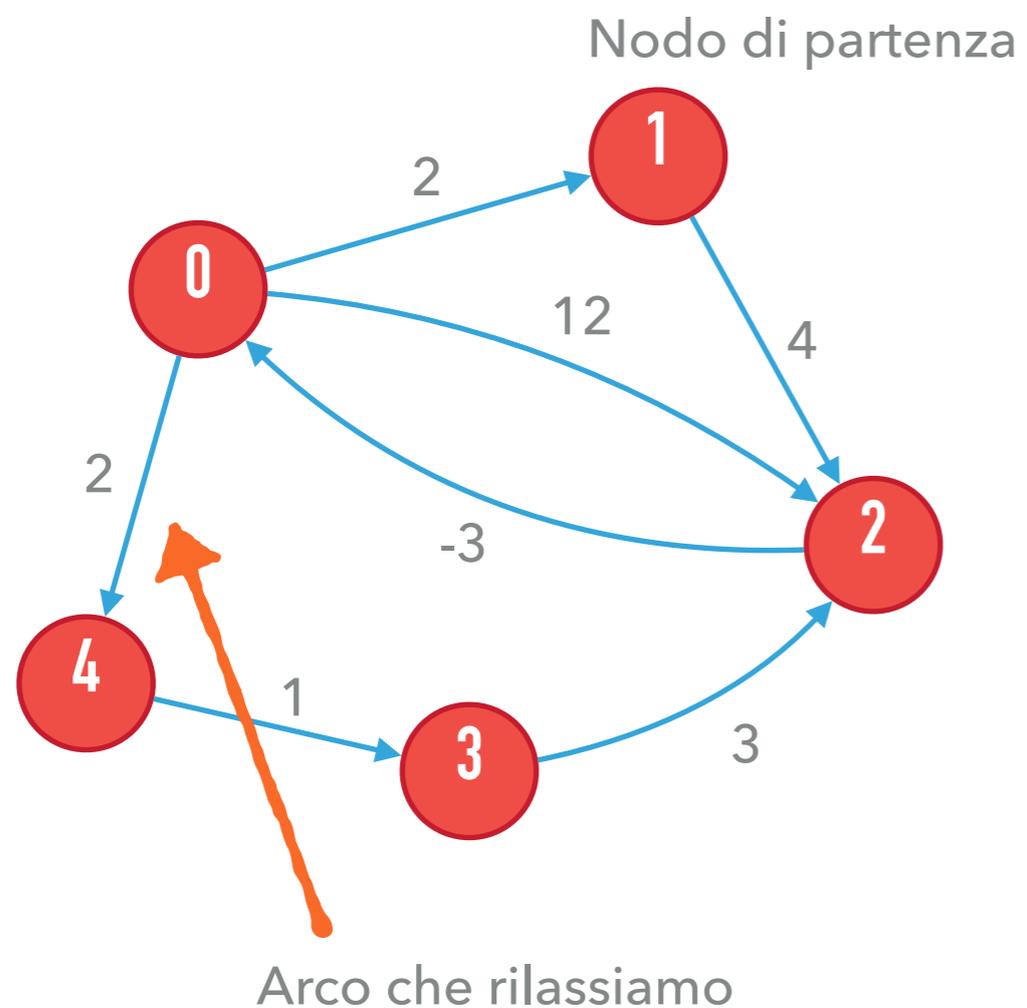
Vertice	Peso
0	1
1	0
2	4
3	$\infty$
4	$\infty$

## ESEMPIO DI ESECUZIONE



Vertice	Peso
0	1
1	0
2	4
3	$\infty$
4	$\infty$

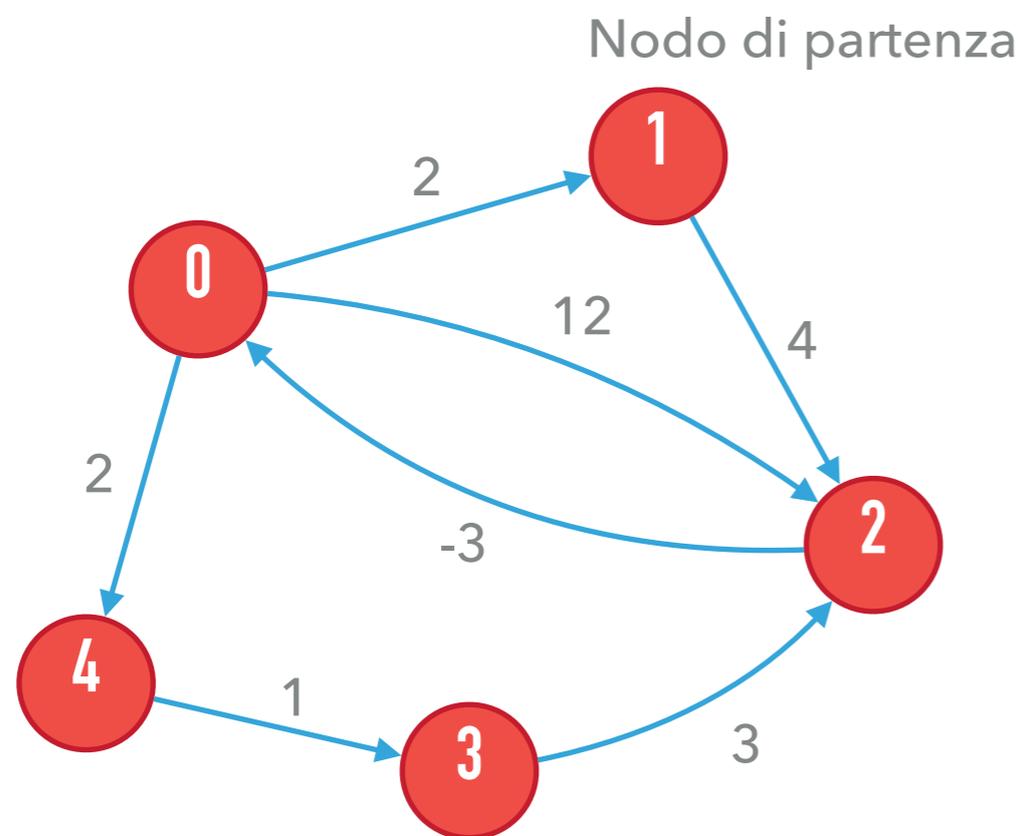
## ESEMPIO DI ESECUZIONE



Ora svolgiamo altre 3 iterazioni su tutti i nodi...

Vertice	Peso
0	1
1	0
2	4
3	$\infty$
4	3

## ESEMPIO DI ESECUZIONE



Ottenendo questo risultato:

Vertice	Peso
0	1
1	0
2	4
3	4
4	3

# ALGORITMO DI BELLMAN-FORD: COMPLESSITÀ

- ▶ La singola operazione di rilassamento richiede tempo costante
- ▶ Ogni iterazione del ciclo interno richiede  $O(E)$  operazioni di rilassamento
- ▶ Ed il ciclo esterno esegue  $O(V)$  volte
- ▶ Il costo totale è quindi  $O(VE)$
- ▶ Peggior di  $O(V + E)$  richiesto da BFS nel caso di grafo non pesato (o con pesi tutti uguali alla stessa costante positiva)

# ALGORITMO DI DIJKSTRA

- ▶ L'algoritmo di Dijkstra è applicabile nel caso più ristretto in cui tutti i pesi sono **non negativi**
- ▶ Questo caso si presenta spesso in casi pratici: tempi di percorrenza, distanze in km, etc
- ▶ Rispetto all'algoritmo di Bellman-Ford riesce ad essere più efficiente sfruttando una coda di priorità: una struttura dati che permette di inserire elementi e rimuovere l'elemento di valore minimo

# IDEA GENERALE: ALGORITMO DI DIJKSTRA

- ▶ Teniamo una lista di nodi non visitati
- ▶ Il nodo iniziale ha distanza 0 e tutti gli altri infinito
- ▶ Prendiamo il nodo  $v$  con distanza minima tra quelli non visitati:
  - ▶ Per ogni vicino aggiorniamo la distanza vedendo se possiamo raggiungerlo più velocemente passando da  $v$
  - ▶ Rimuoviamo  $v$  dalla lista dei nodi non visitati
- ▶ Continuiamo finché non possiamo più aggiornare le distanze dei nodi

## PSEUDOCODICE

Parametri: grafo  $G$ , nodo sorgente  $s$

# inizialmente impostiamo distanza e predecessore di tutti i nodi

for all  $v \in V$ :

    distanza[ $v$ ] =  $+\infty$  # indichiamo distanza[ $v$ ] anche come  $d[v]$

    predecessore[ $v$ ] = None

distanza[ $s$ ] = 0 # e poi impostiamo il nodo  $s$  come sorgente a distanza 0

$S = \{\}$  # insieme dei nodi già visitati (inizialmente vuoto)

$Q = \text{coda\_di\_priorità}(V)$  # coda di priorità con tutti i vertici già inseriti

while  $Q$  is non-empty

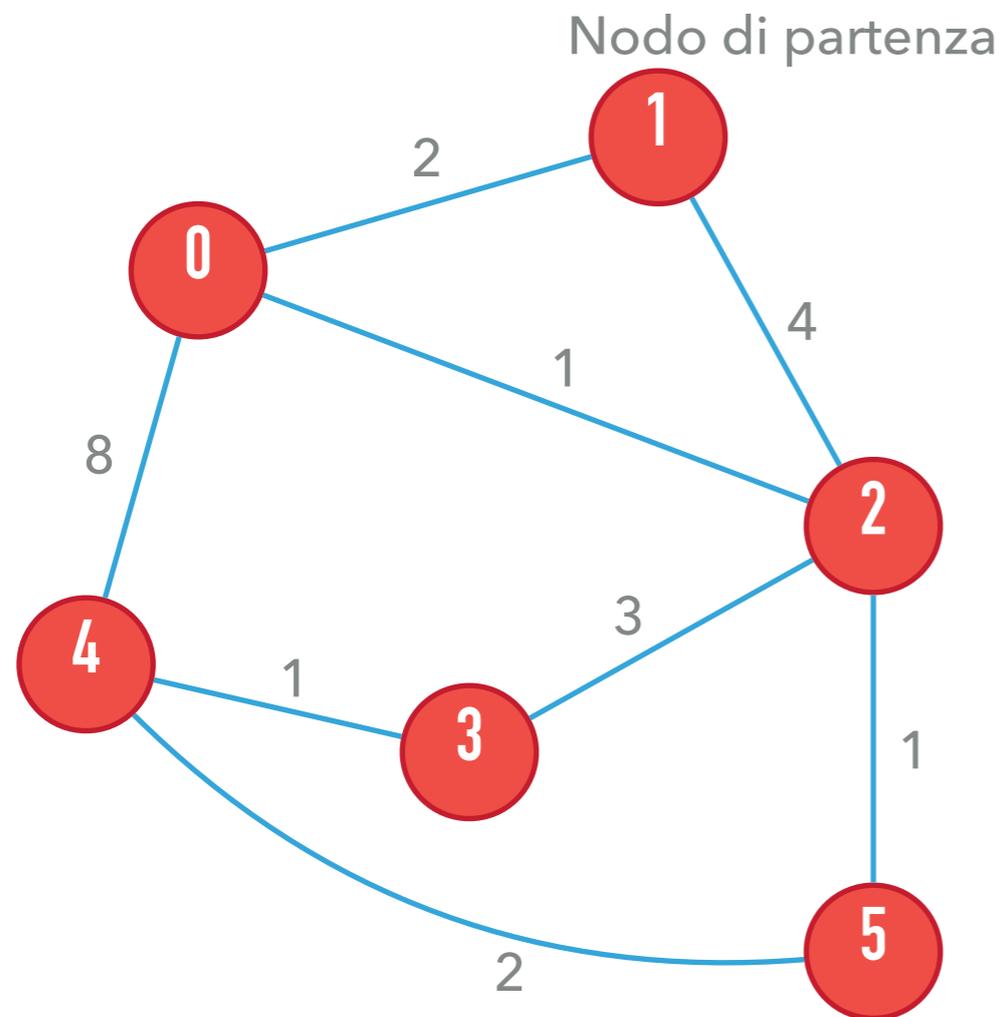
$u = \text{estrai\_minimo}(Q)$  # estraiamo il prossimo vertice

$S = S \cup \{u\}$  # lo inseriamo nella lista dei visitati

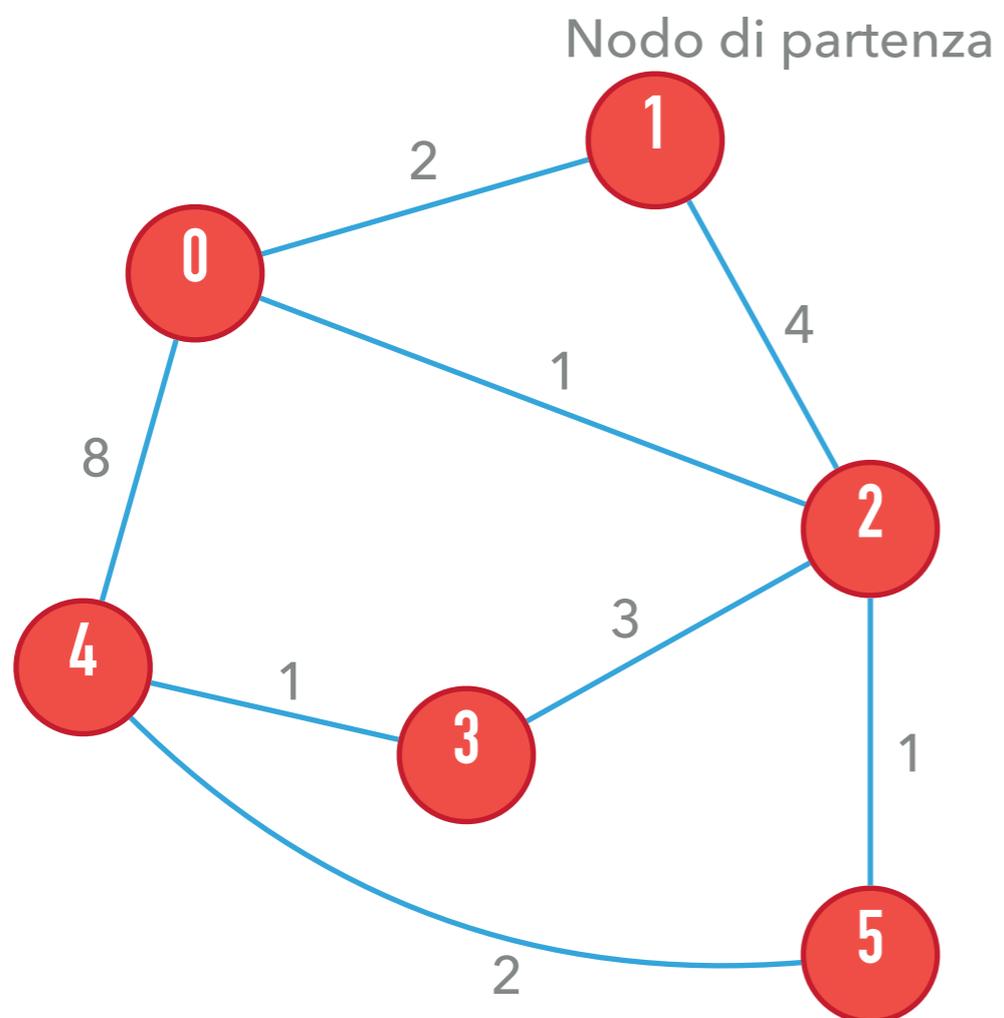
    for all  $v$  adiacenti a  $u$  # e aggiorniamo la distanza di tutti i vicini

        RELAX( $u, v, w_{u,v}$ )

## ESEMPIO DI ESECUZIONE



## ESEMPIO DI ESECUZIONE



Lista dei nodi non visitati



Vertice	Peso
0	$\infty$
1	0
2	$\infty$
3	$\infty$
4	$\infty$
5	$\infty$

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

1

Lista dei nodi non visitati

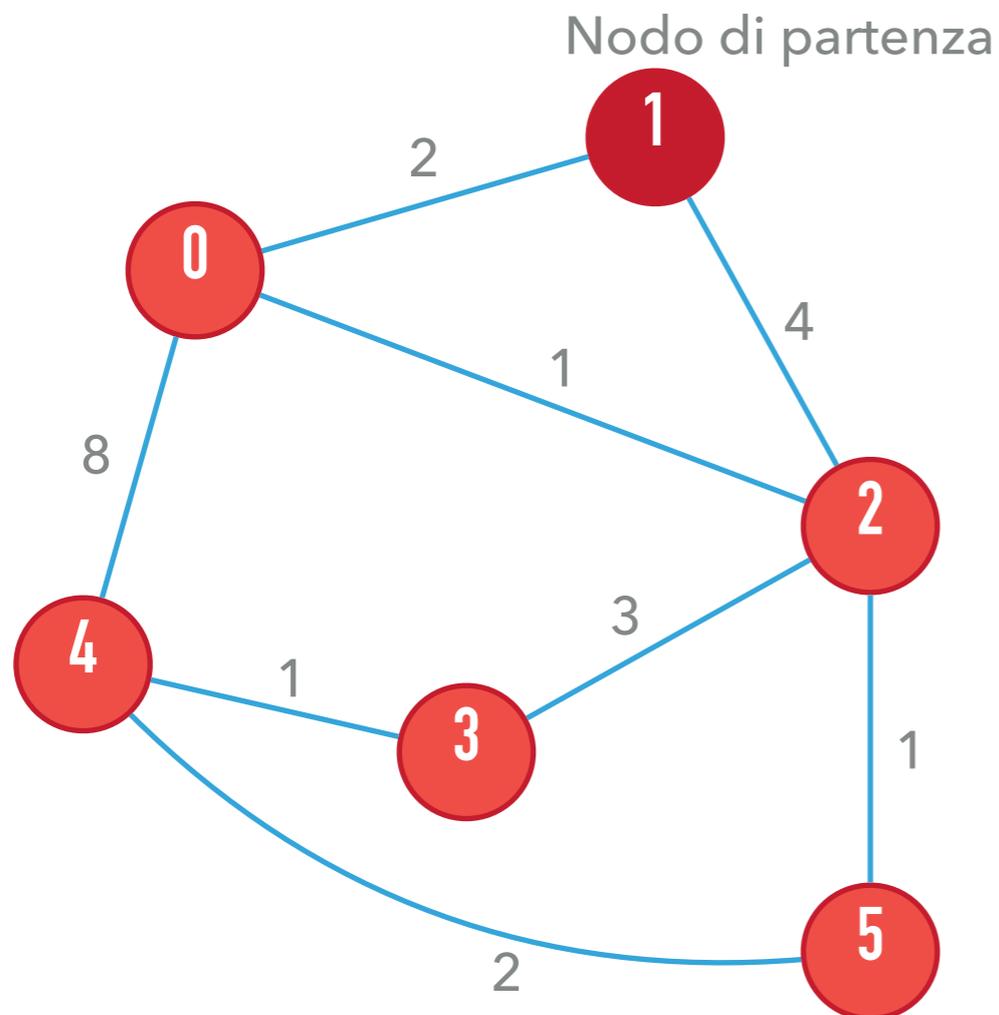
0

2

3

4

5



Vertice	Peso
---------	------

0

$\infty$

1

0

2

$\infty$

3

$\infty$

4

$\infty$

5

$\infty$

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

1

Lista dei nodi non visitati

0

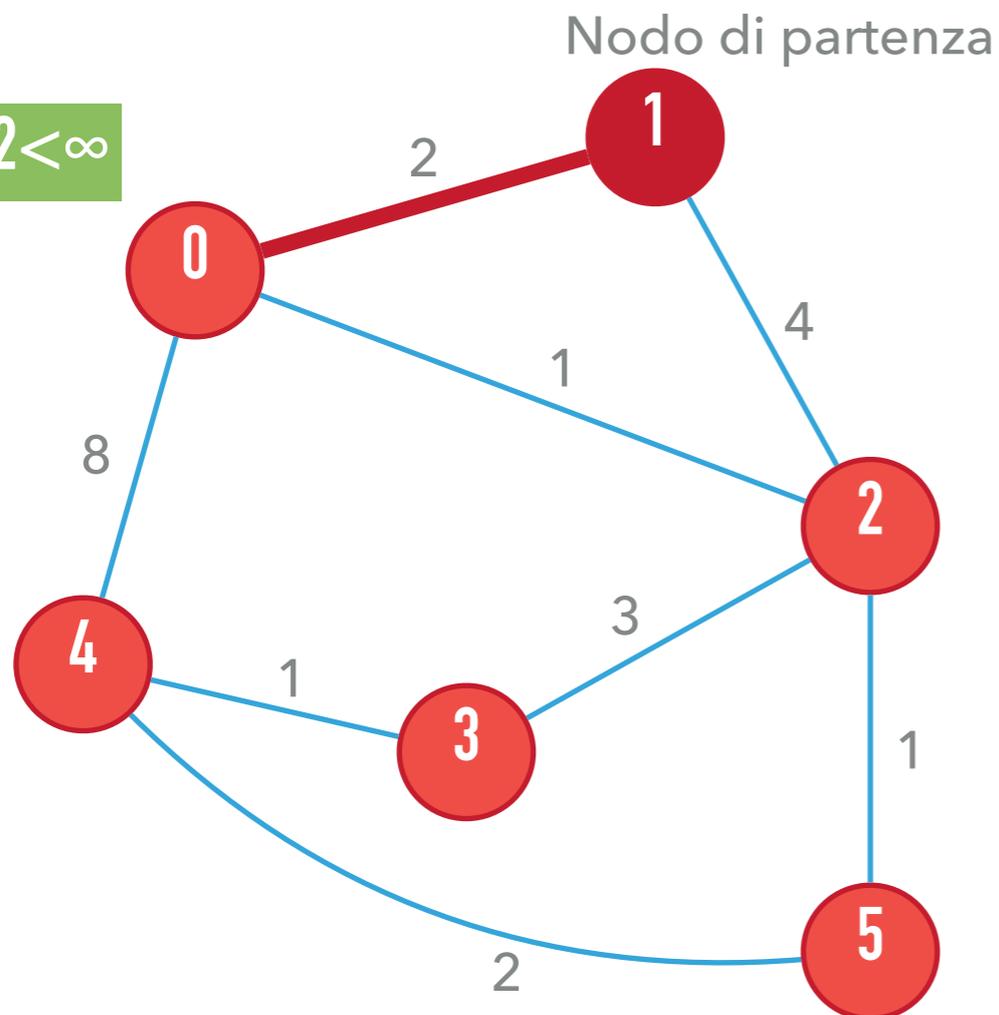
2

3

4

5

$0+2 < \infty$



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	$\infty$
---	----------

3	$\infty$
---	----------

4	$\infty$
---	----------

5	$\infty$
---	----------

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

1

Lista dei nodi non visitati

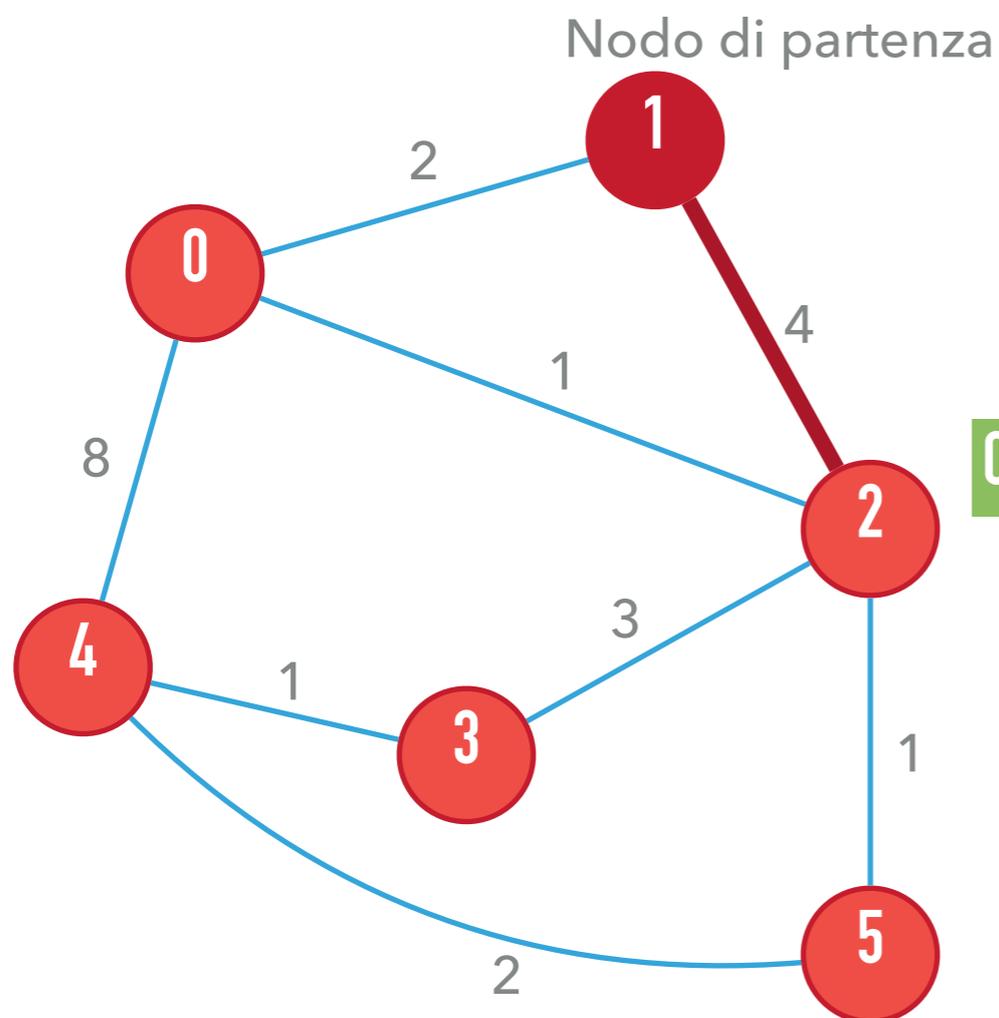
0

2

3

4

5



$$0+4 < \infty$$

Vertice	Peso
---------	------

0

2

1

0

2

4

3

$\infty$

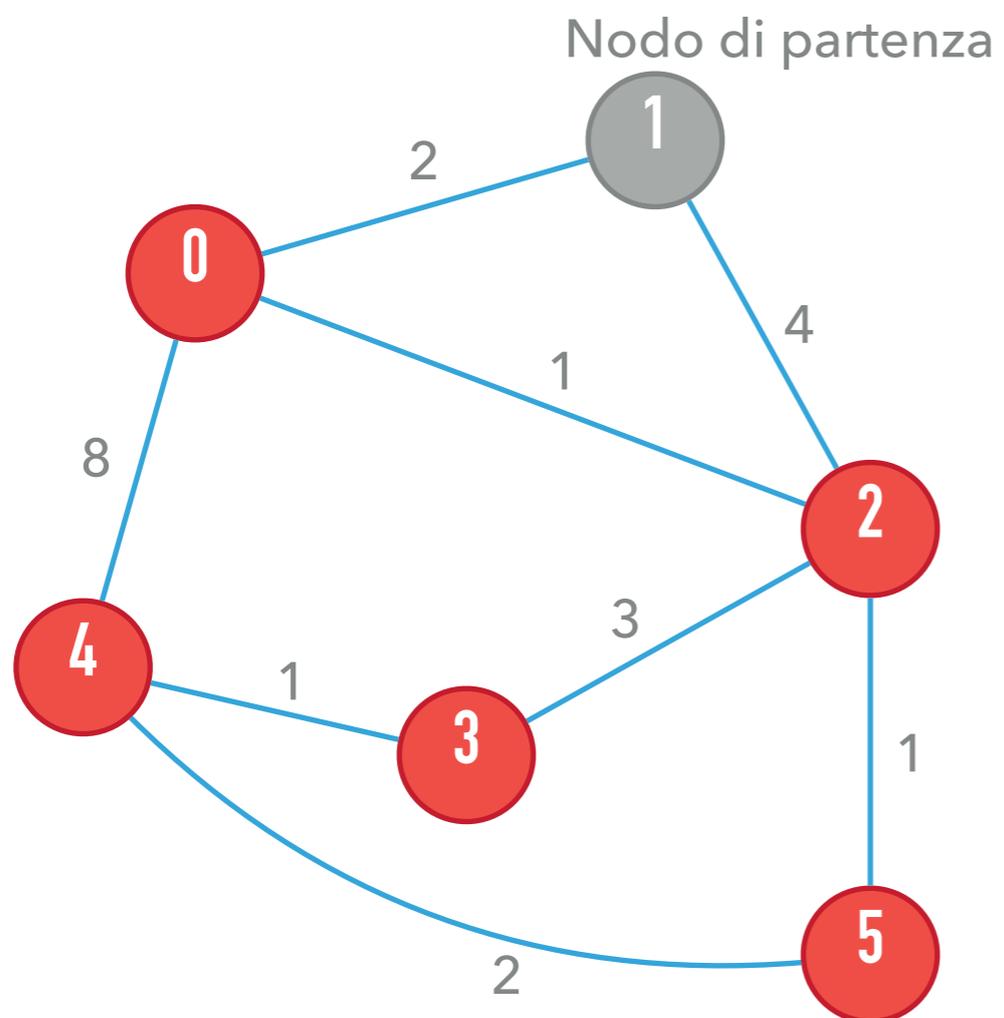
4

$\infty$

5

$\infty$

## ESEMPIO DI ESECUZIONE



Lista dei nodi non visitati



Vertice	Peso
0	2
1	0
2	4
3	$\infty$
4	$\infty$
5	$\infty$

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

0

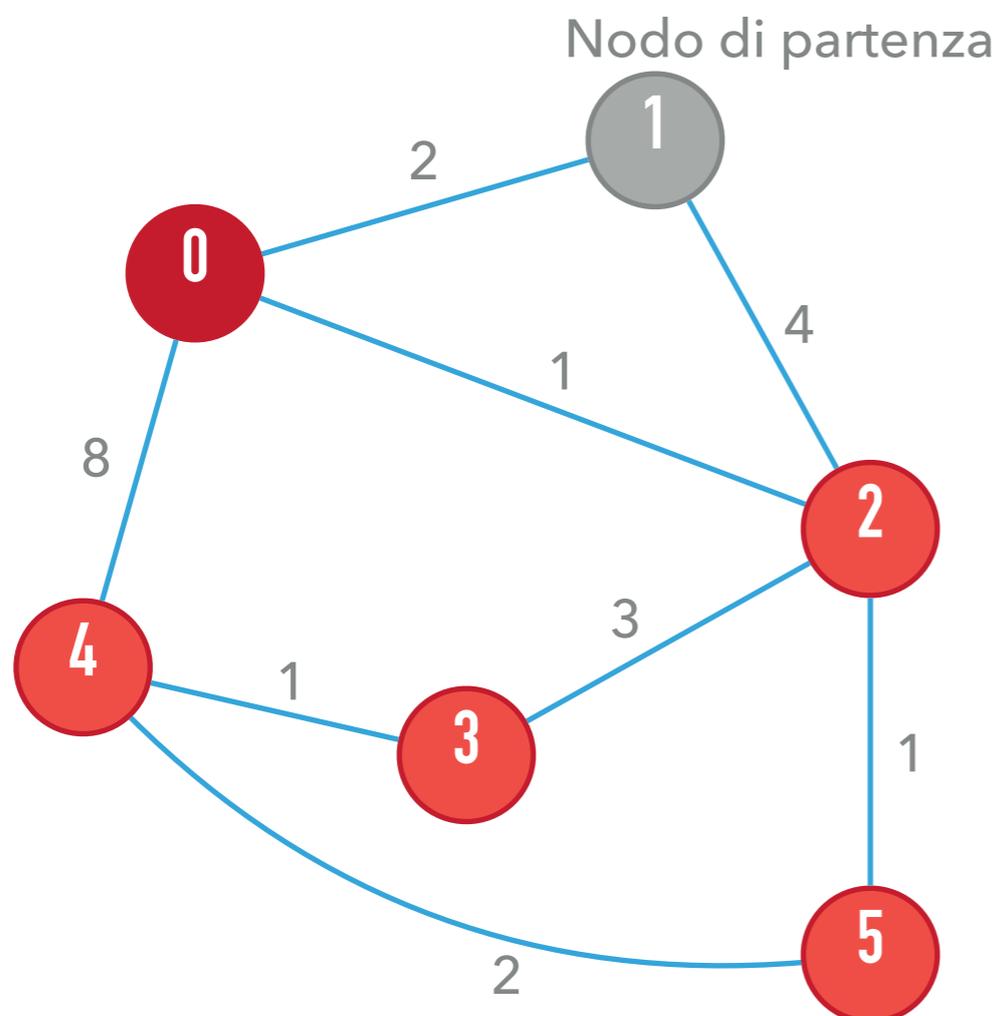
Lista dei nodi non visitati

2

3

4

5



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	4
---	---

3	$\infty$
---	----------

4	$\infty$
---	----------

5	$\infty$
---	----------

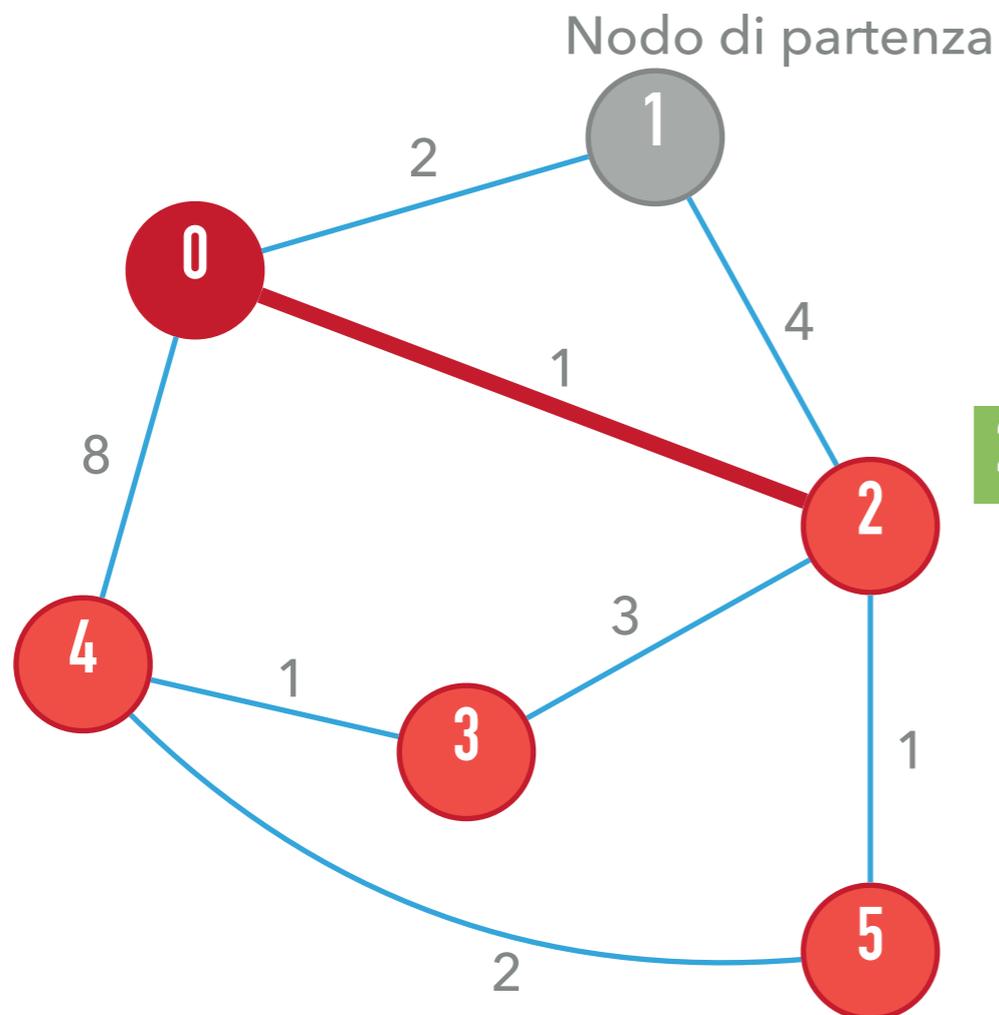
## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

0

Lista dei nodi non visitati

2 3 4 5



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	3
---	---

3	$\infty$
---	----------

4	$\infty$
---	----------

5	$\infty$
---	----------

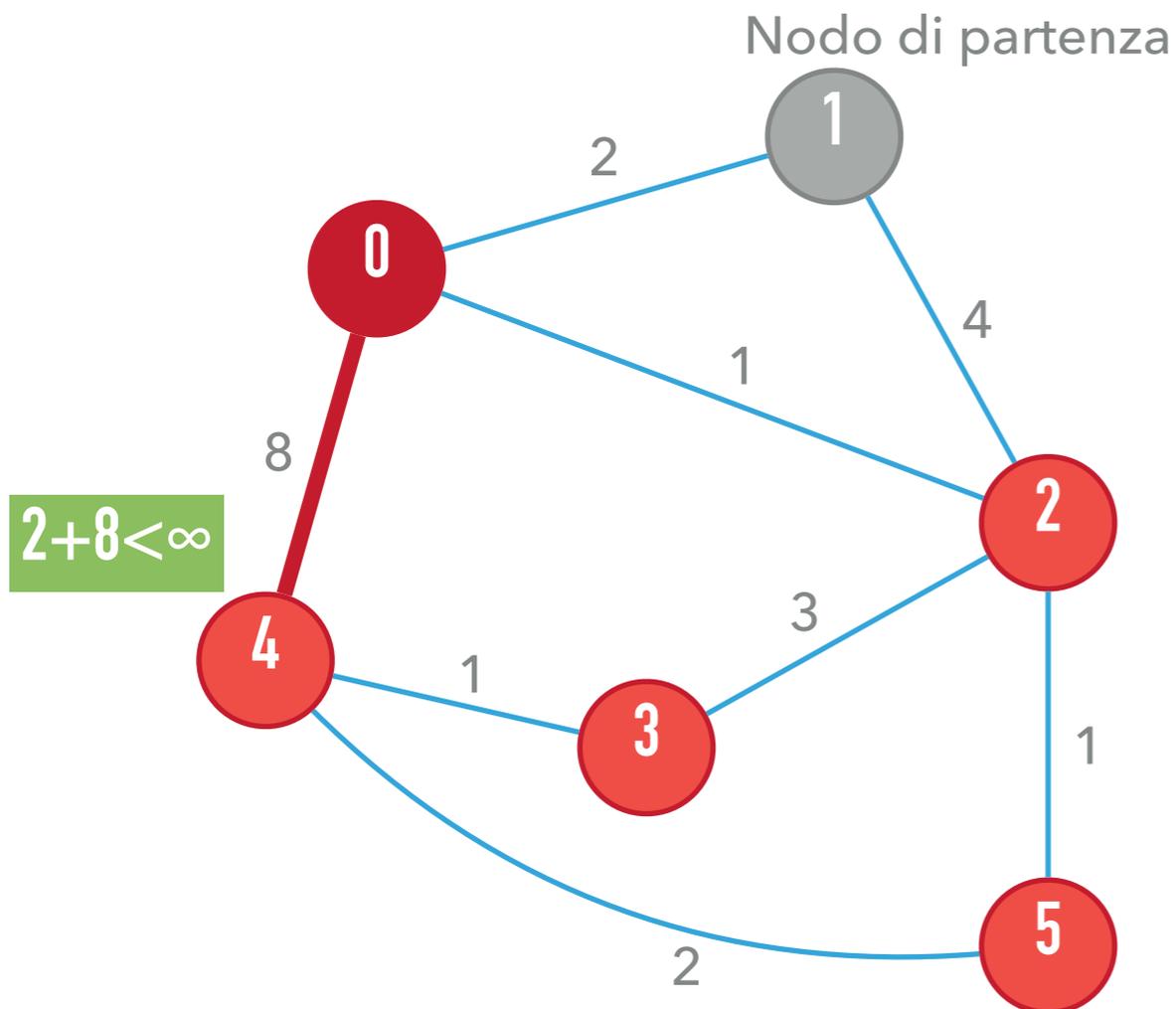
## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

0

Lista dei nodi non visitati

2 3 4 5



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

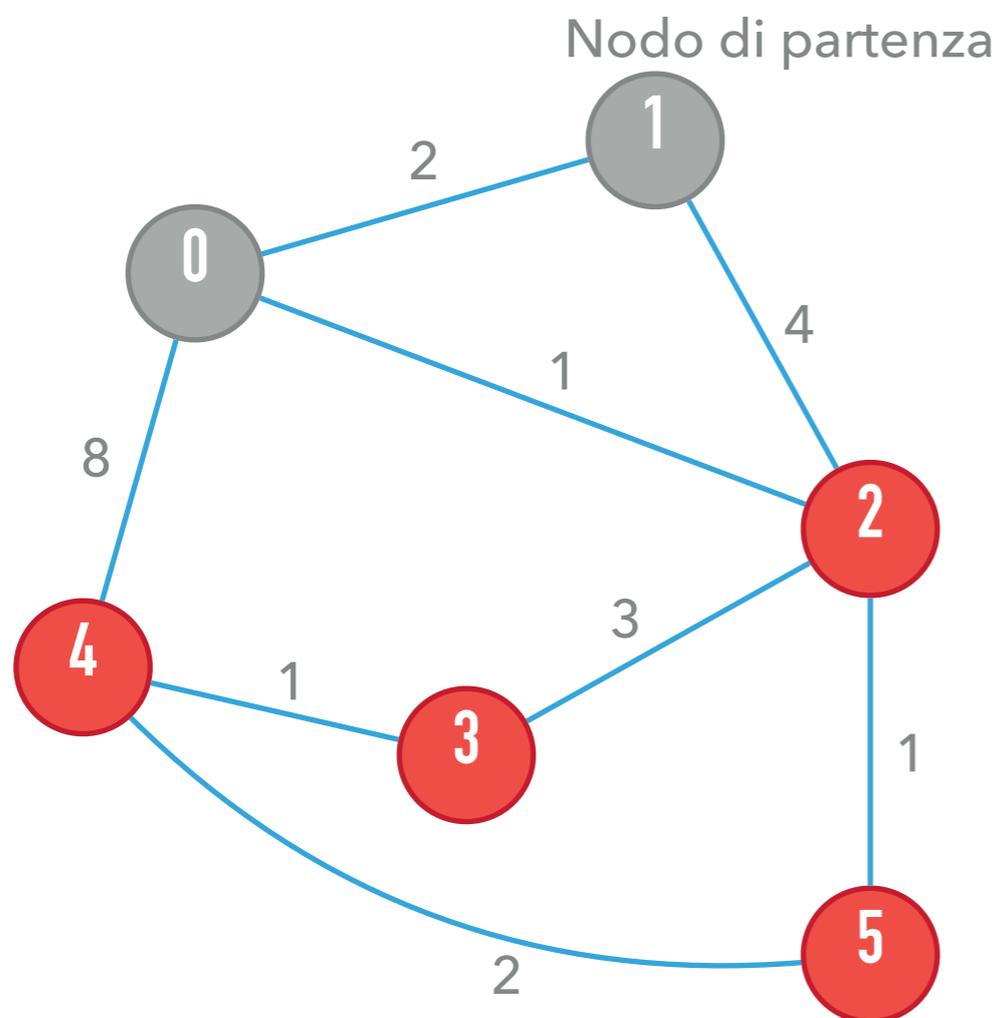
2	3
---	---

3	$\infty$
---	----------

4	10
---	----

5	$\infty$
---	----------

## ESEMPIO DI ESECUZIONE



Lista dei nodi non visitati



Vertice	Peso
0	2
1	0
2	3
3	$\infty$
4	10
5	$\infty$

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

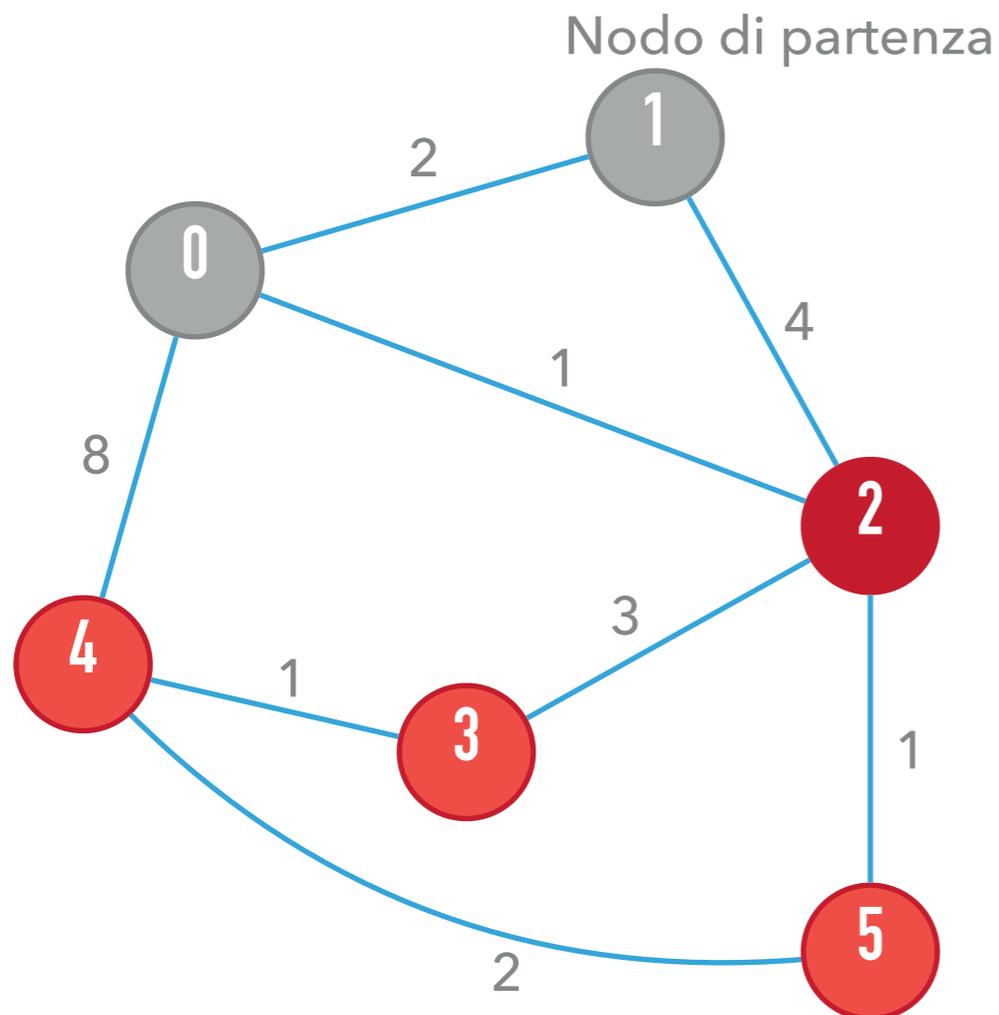
2

Lista dei nodi non visitati

3

4

5



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	3
---	---

3	$\infty$
---	----------

4	10
---	----

5	$\infty$
---	----------

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

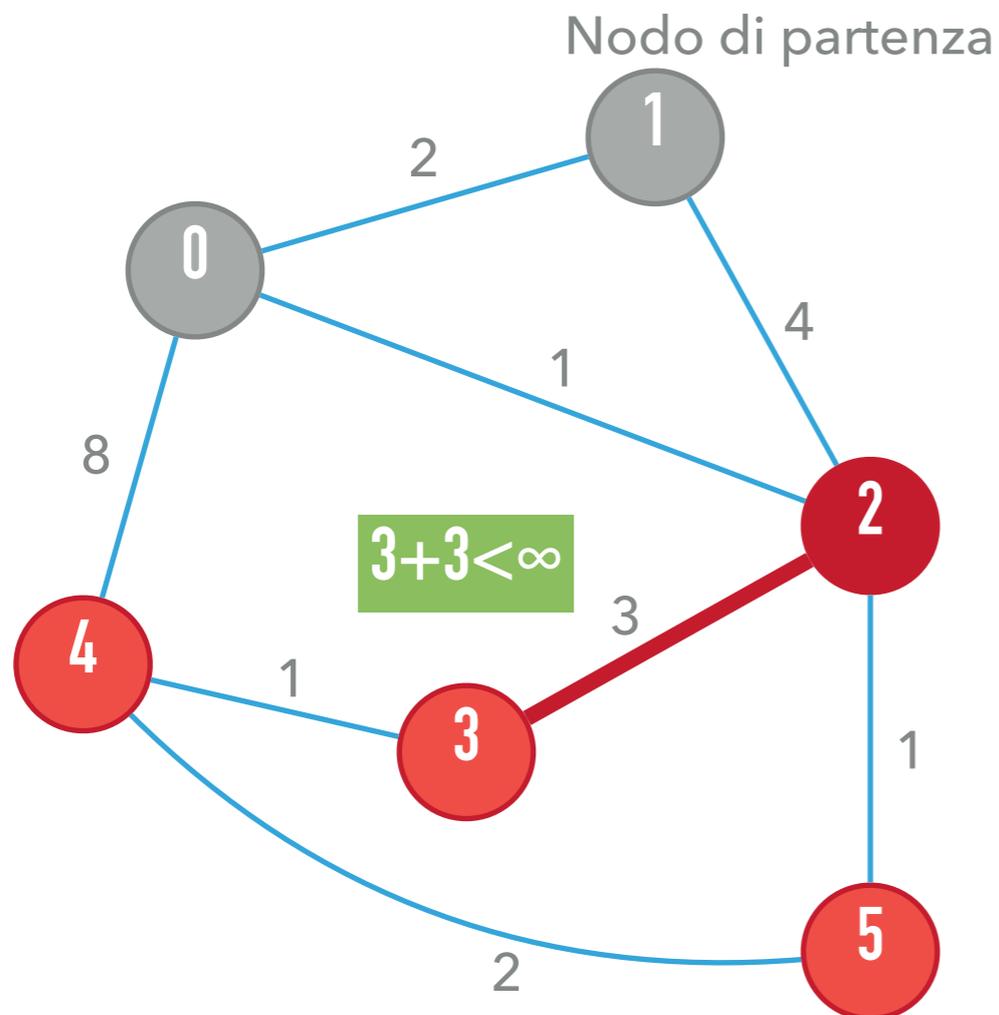
2

Lista dei nodi non visitati

3

4

5



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	3
---	---

3	6
---	---

4	10
---	----

5	∞
---	---

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

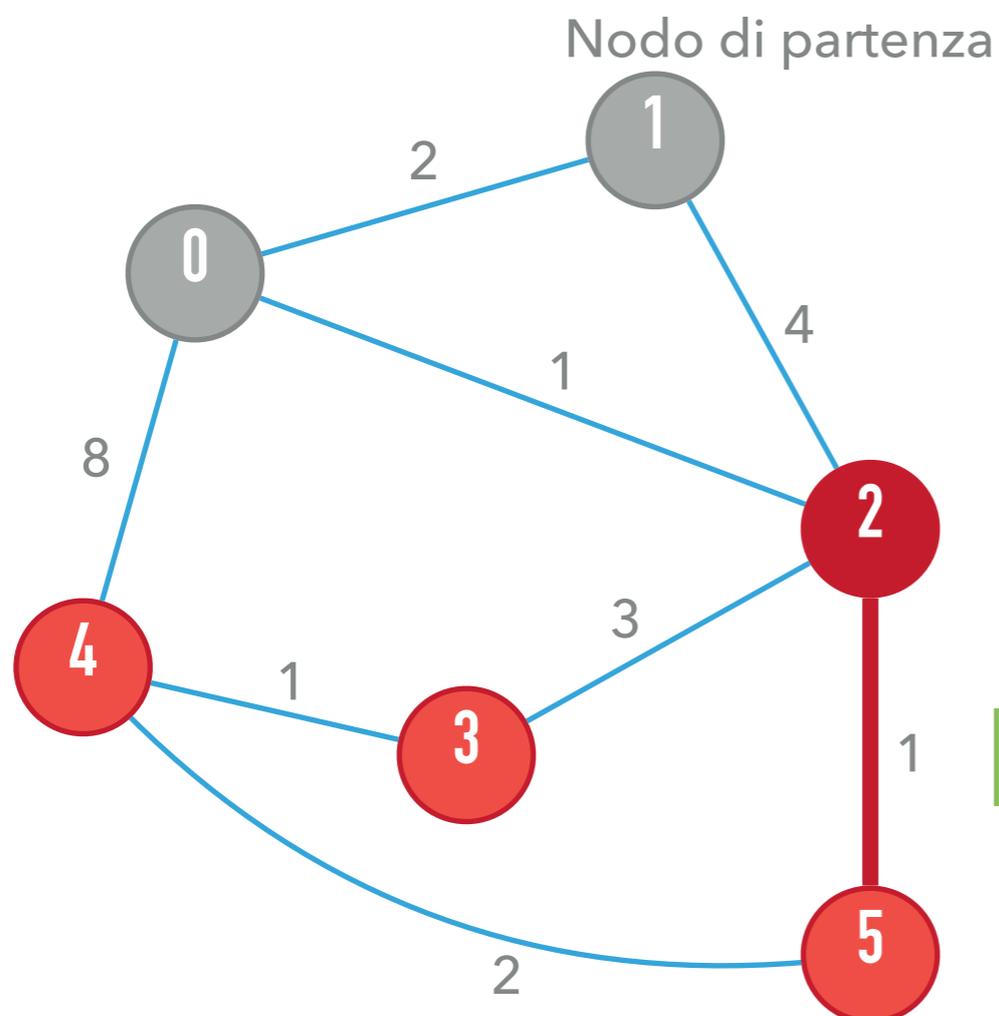
2

Lista dei nodi non visitati

3

4

5



$$3+1 < \infty$$

Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

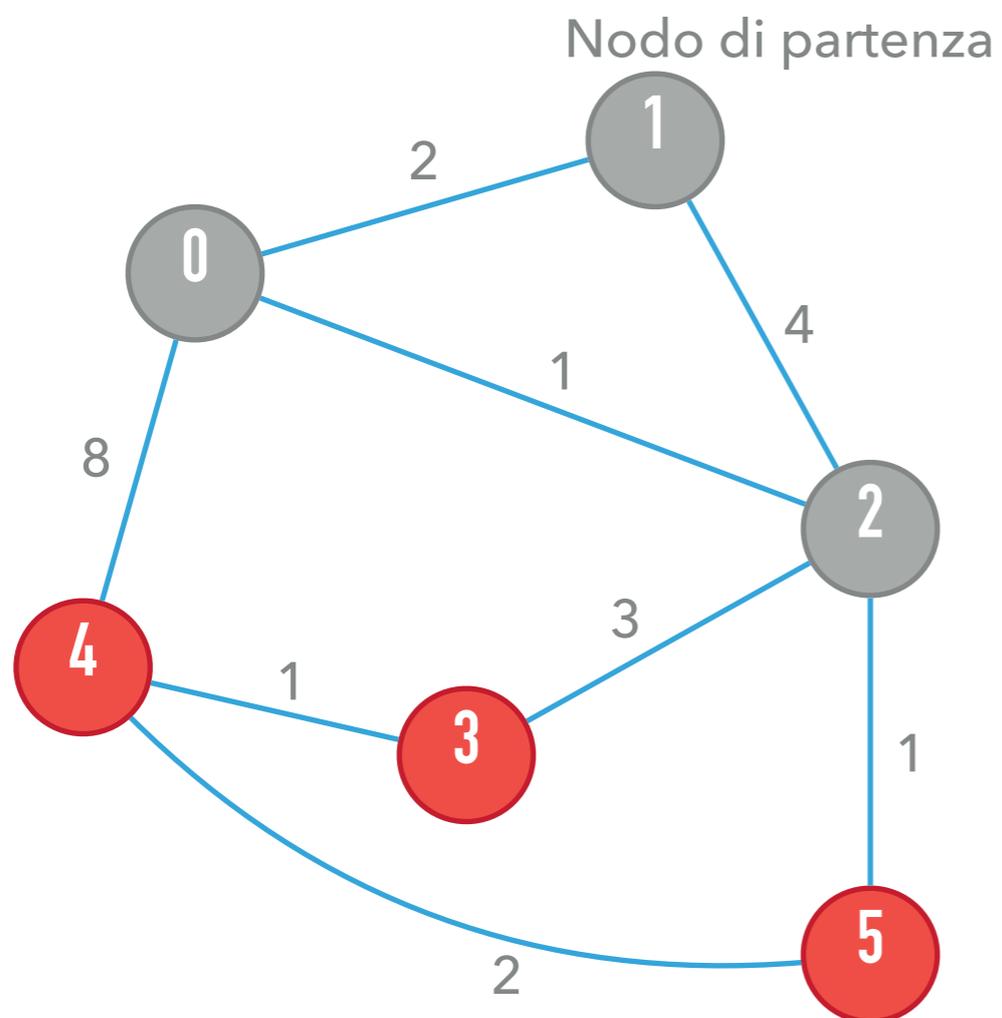
2	3
---	---

3	6
---	---

4	10
---	----

5	4
---	---

## ESEMPIO DI ESECUZIONE



Lista dei nodi non visitati



Vertice	Peso
0	2
1	0
2	3
3	6
4	10
5	4

## ESEMPIO DI ESECUZIONE

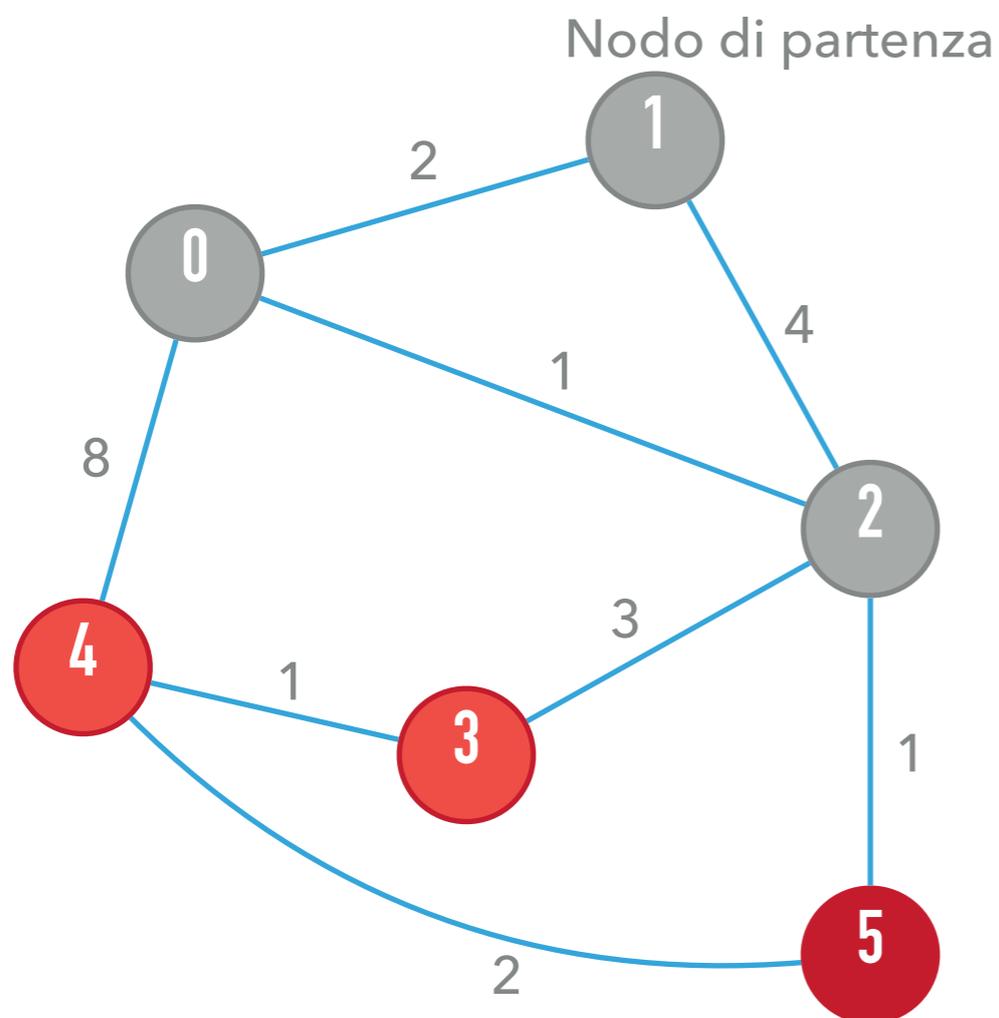
Nodo con distanza minima:

5

Lista dei nodi non visitati

3

4



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	3
---	---

3	6
---	---

4	10
---	----

5	4
---	---

## ESEMPIO DI ESECUZIONE

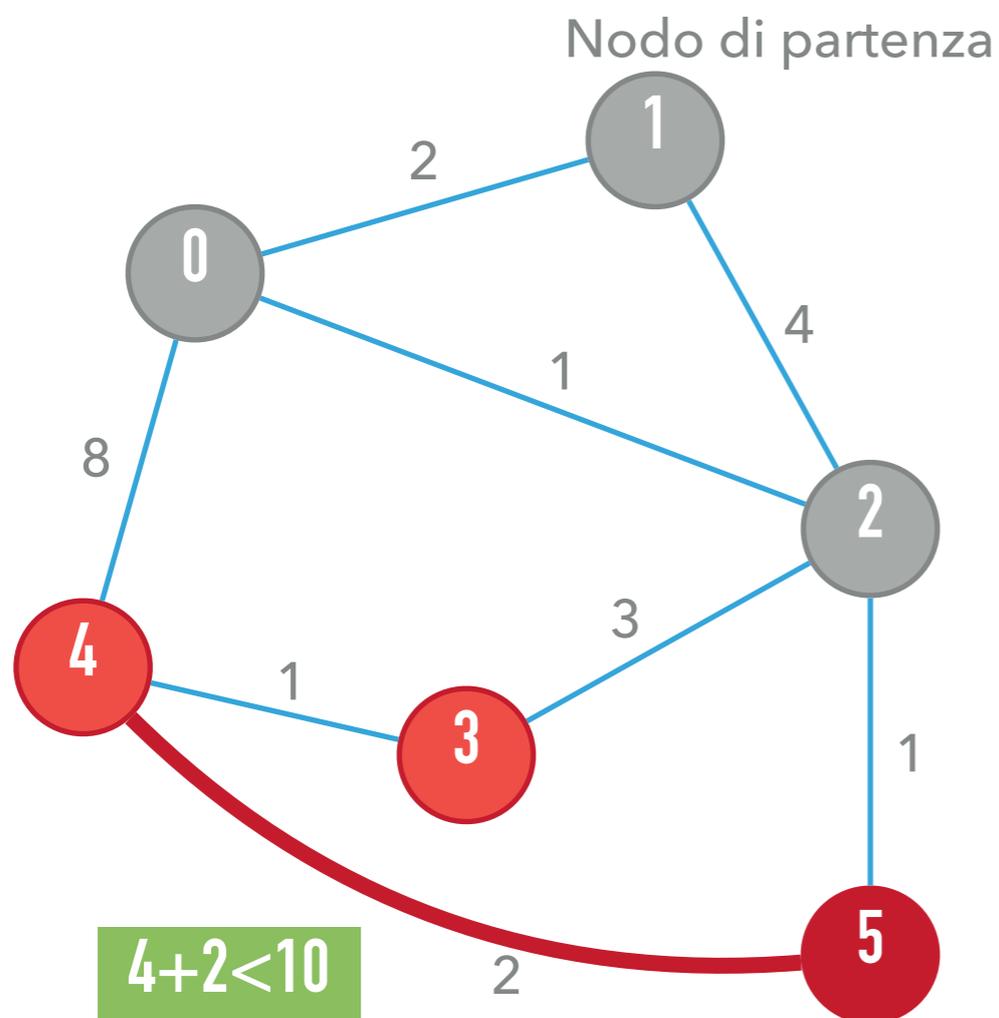
Nodo con distanza minima:

5

Lista dei nodi non visitati

3

4



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

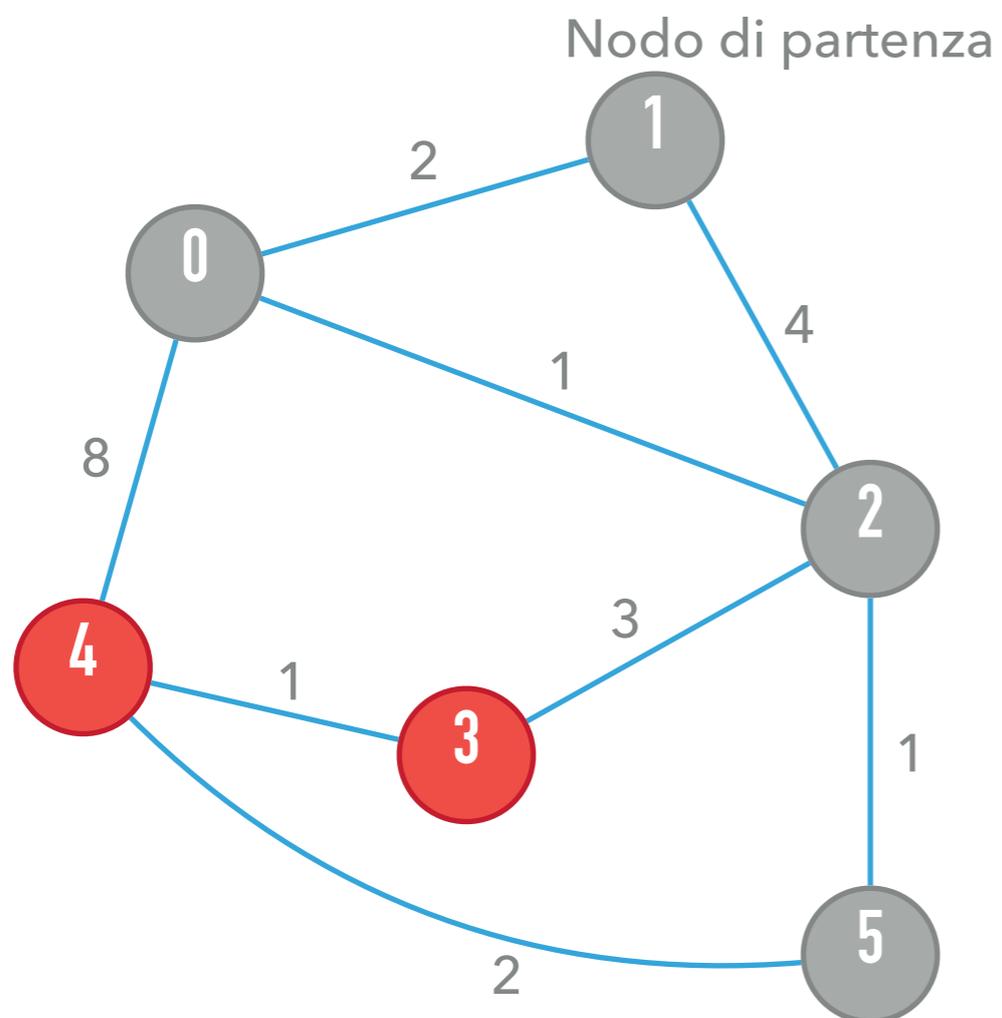
2	3
---	---

3	6
---	---

4	6
---	---

5	4
---	---

## ESEMPIO DI ESECUZIONE



Lista dei nodi non visitati



Vertice	Peso
0	2
1	0
2	3
3	6
4	6
5	4

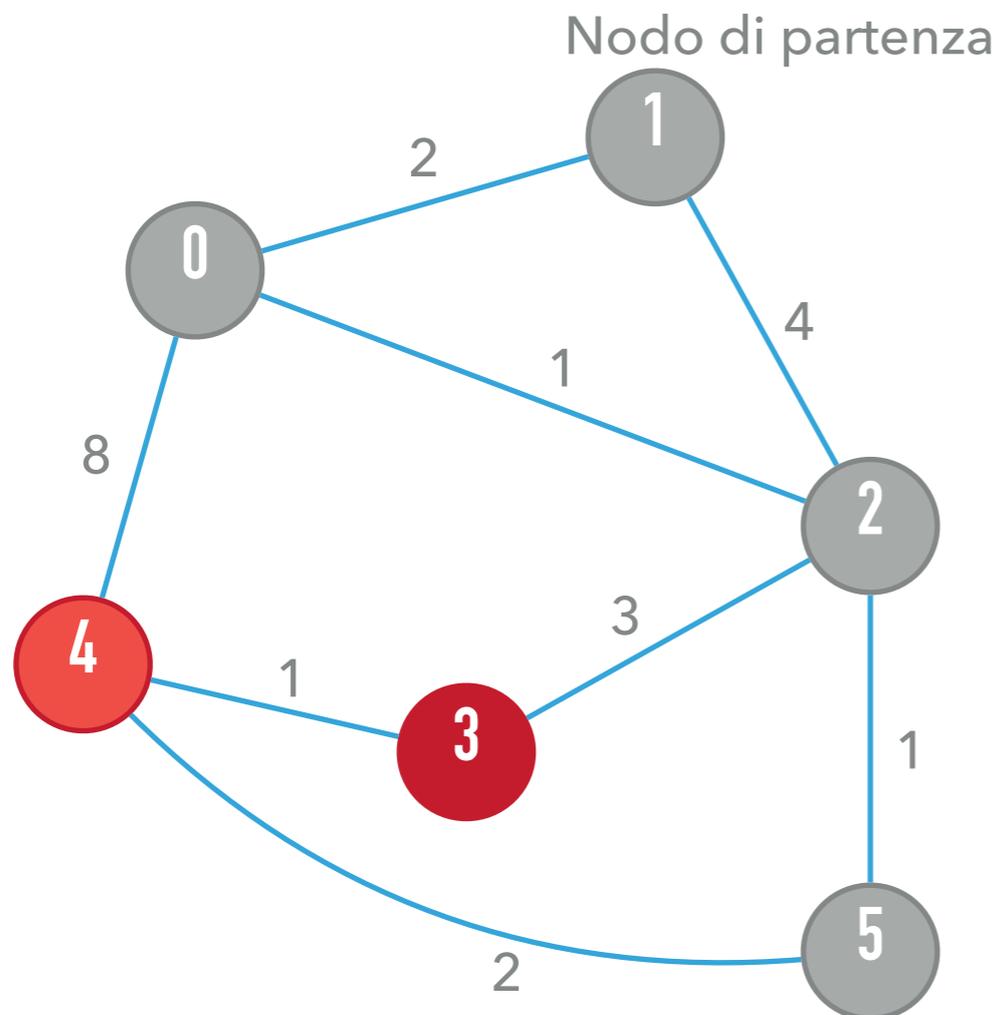
## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

3

Lista dei nodi non visitati

4



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

2	3
---	---

3	6
---	---

4	6
---	---

5	4
---	---

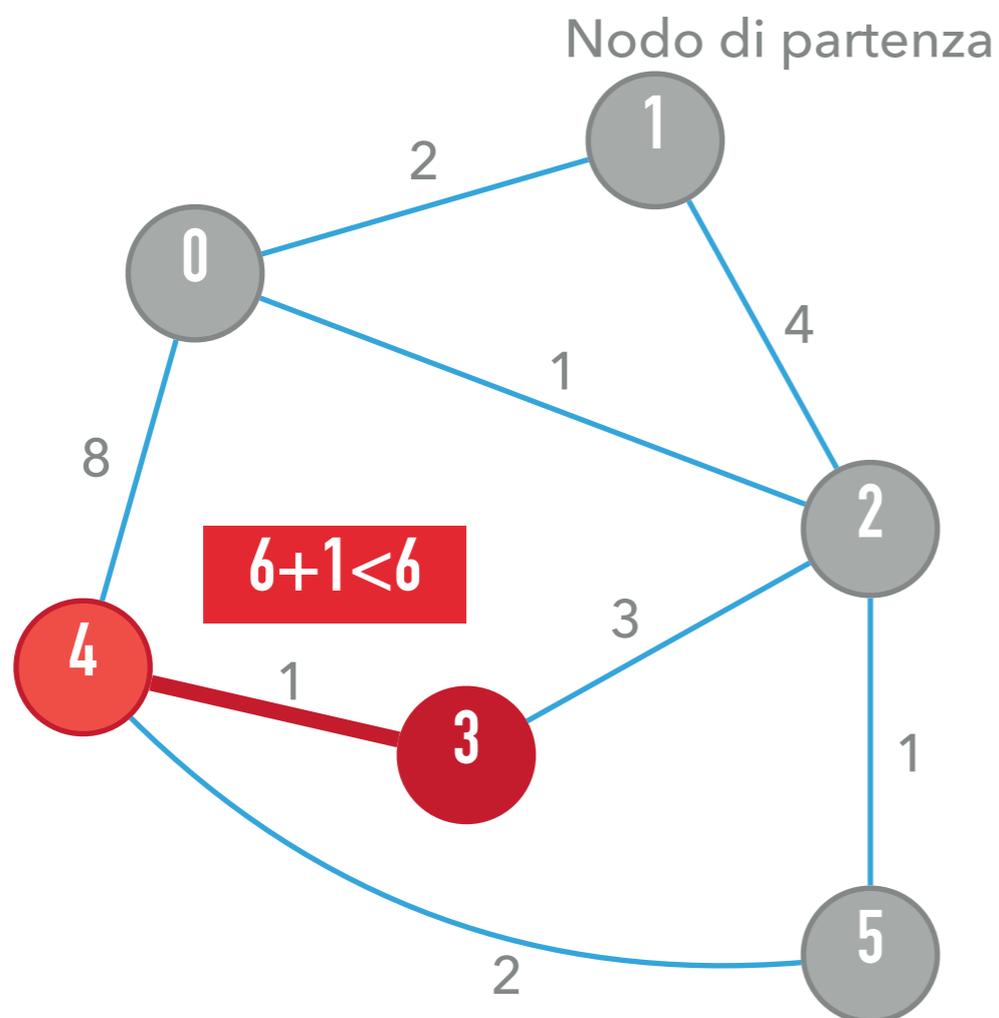
## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

3

Lista dei nodi non visitati

4



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

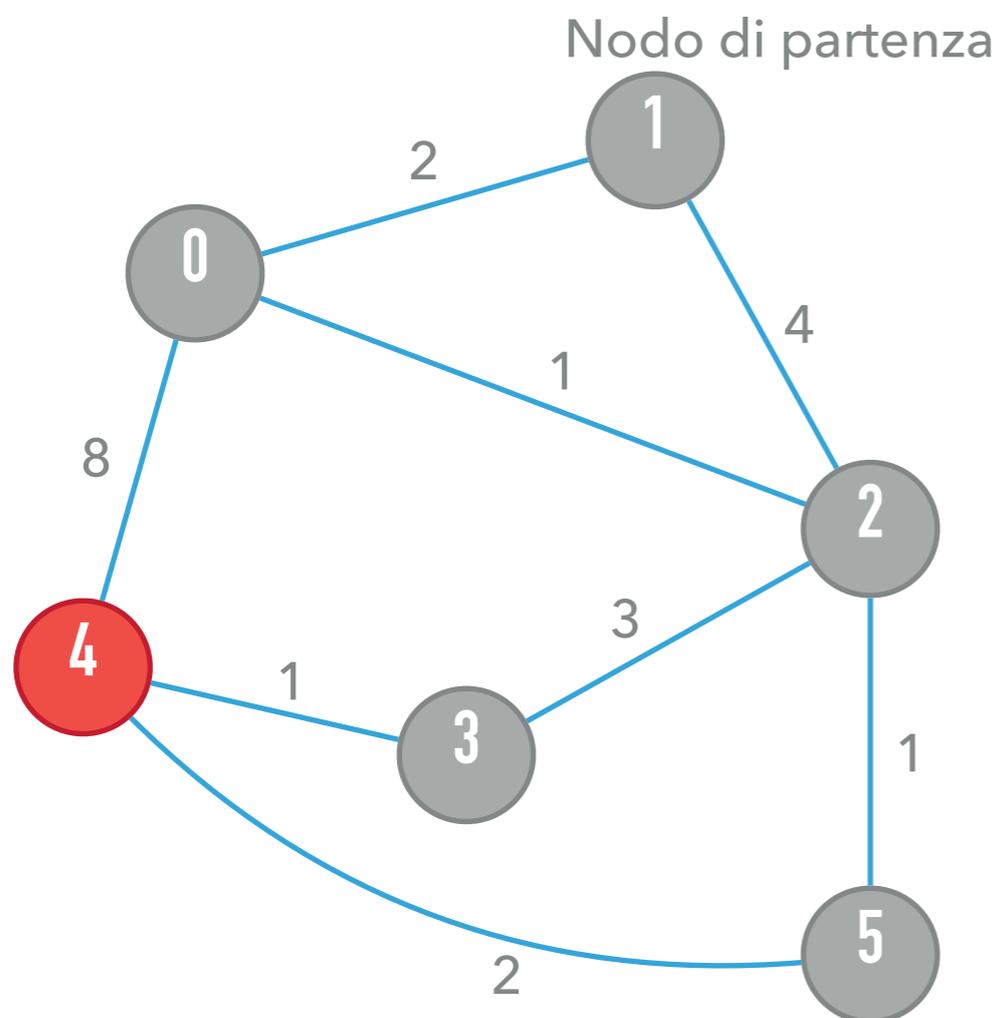
2	3
---	---

3	6
---	---

4	6
---	---

5	4
---	---

## ESEMPIO DI ESECUZIONE



Lista dei nodi non visitati

4

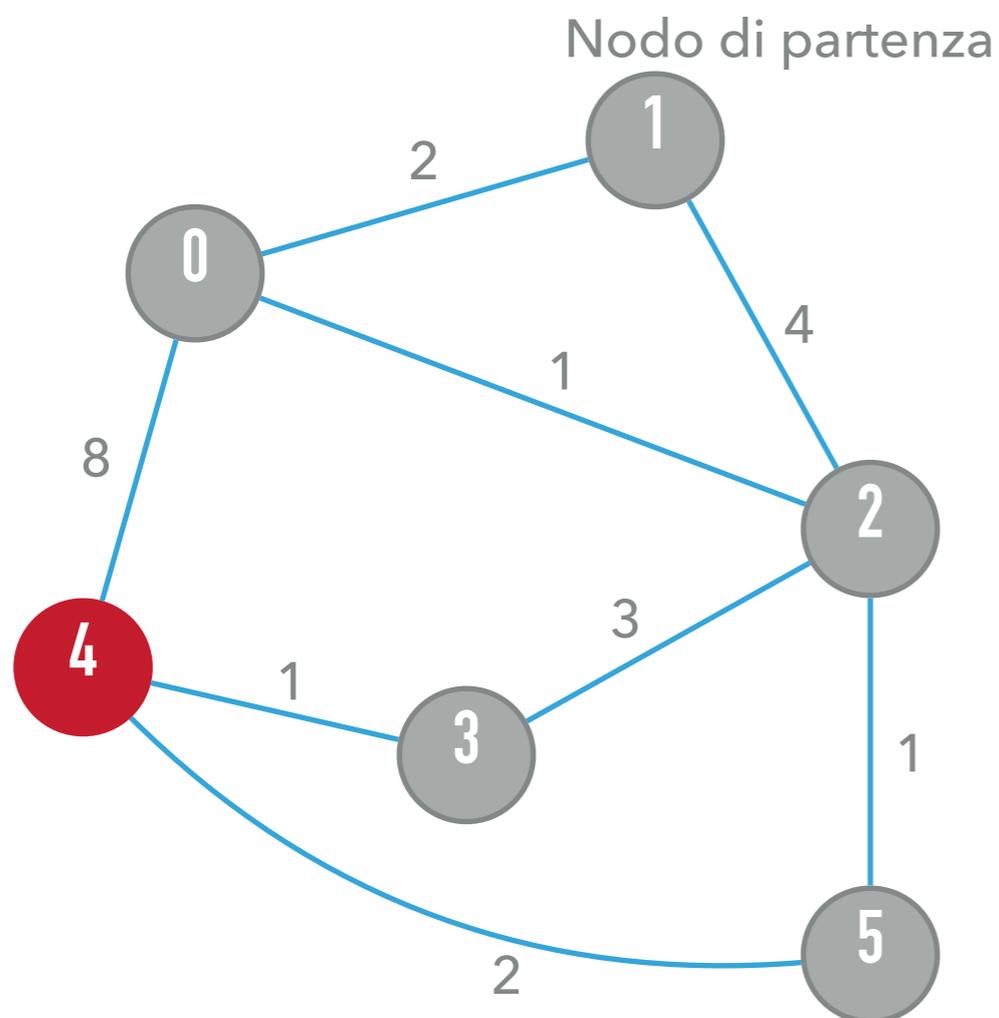
Vertice	Peso
0	2
1	0
2	3
3	6
4	6
5	4

## ESEMPIO DI ESECUZIONE

Nodo con distanza minima:

4

Lista dei nodi non visitati



Vertice	Peso
---------	------

0	2
---	---

1	0
---	---

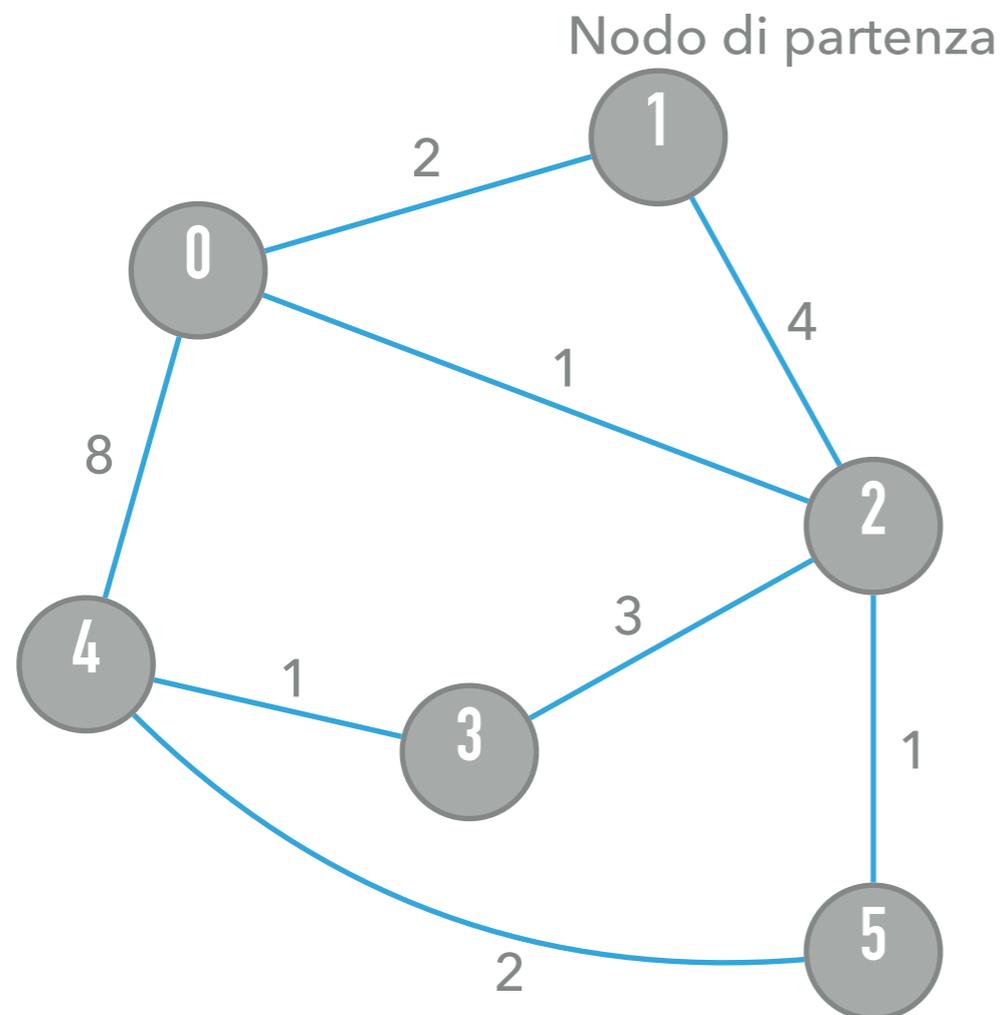
2	3
---	---

3	6
---	---

4	6
---	---

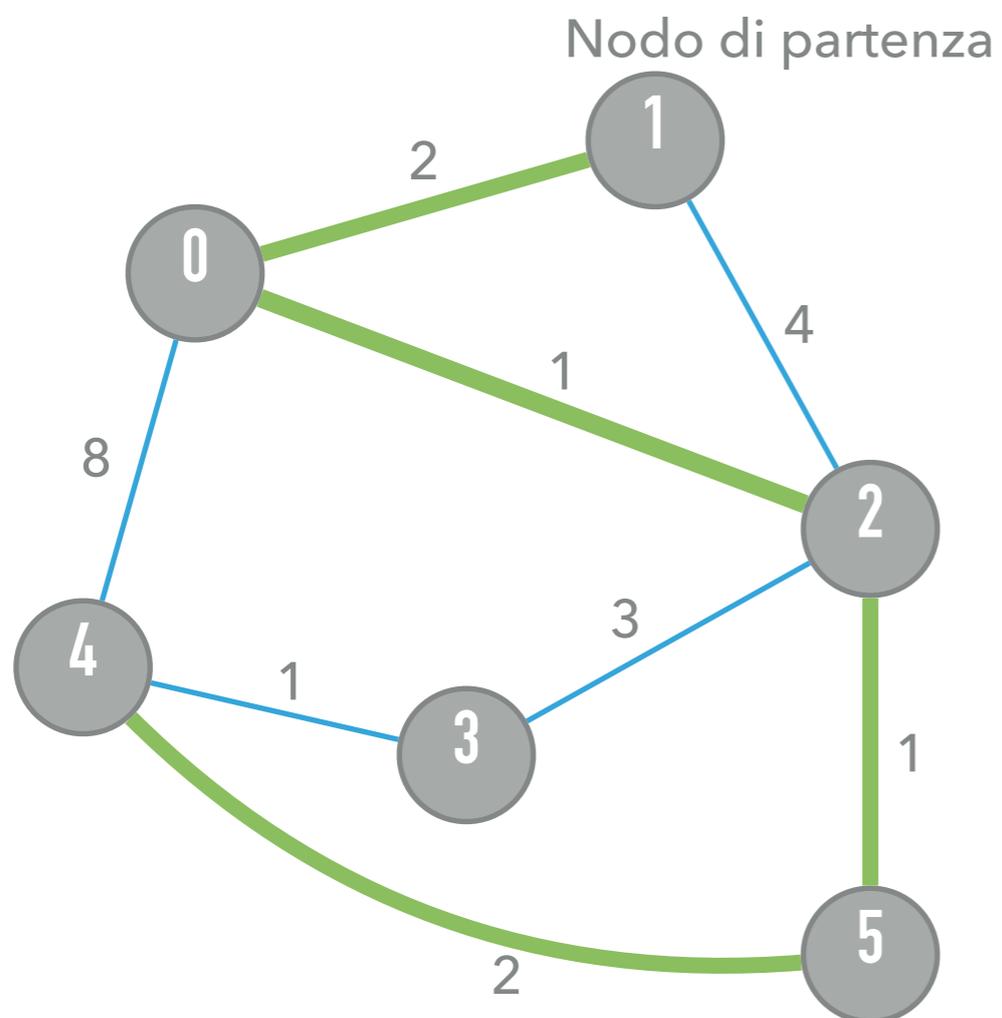
5	4
---	---

## ESEMPIO DI ESECUZIONE



Vertice	Peso
0	2
1	0
2	3
3	6
4	6
5	4

## ESEMPIO DI ESECUZIONE

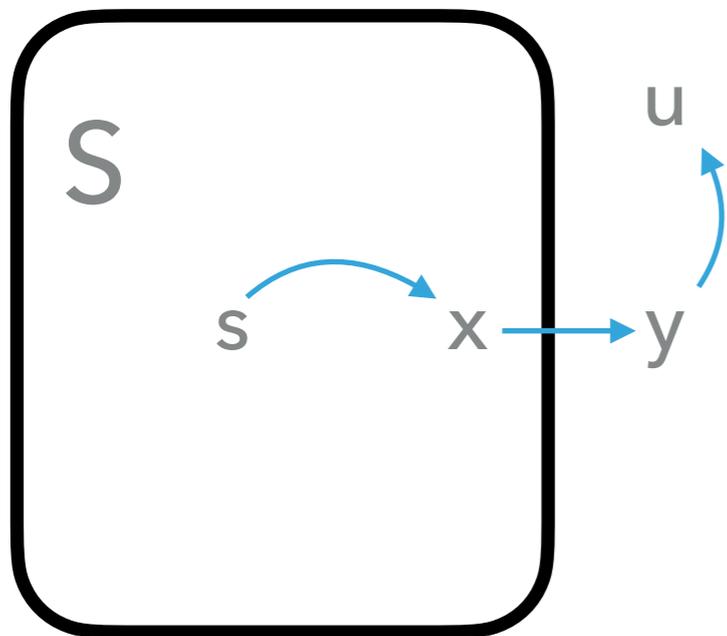


Percorso di lunghezza minima  
per andare dal nodo (1) al nodo (4)

Vertice	Peso
0	2
1	0
2	3
3	6
4	6
5	4

## CORRETTEZZA DI DIJKSTRA

- ▶ Alla fine di Dijkstra,  $distanza[v] = d[v] = \delta(s, v), \forall v \in V$
- ▶ Teorema:  $\forall u$ , all'inserimento di  $u$  in  $S$ , vale  $d[u] = \delta(s, u)$ .
- ▶ Per assurdo, sia  $u$  il primo vertice inserito in  $S$  che viola questa proprietà, e  $y$  il primo vertice su un **cammino minimo** da  $s$  a  $u$  non ancora in  $S$  all'inserimento di  $u$ .



- ▶ Quindi  $y \in Q$  e  $d[y] \geq d[u]$
- ▶  $x \in S \Rightarrow d[x] = \delta(s, x)$ , per scelta di  $u$
- ▶ Il cammino  $s \rightarrow x \rightarrow y$  è minimo, ergo  $d[y] = \delta(s, x) + w_{x,y} = \delta(s, y)$ , perché  $x$  è stato rilassato e valeva  $d[x] = \delta(s, x)$  in tal momento.
- ▶ Da  $d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$  e  $d[u] \leq d[y]$  (scelta di  $u$ ) segue  $d[u] = \delta(s, u)$ , assurdo.

# CONSIDERAZIONI SUL TEMPO DI CALCOLO

- ▶ Visitiamo ogni nodo al più una volta: quando lo estraiamo dalla lista
- ▶ Visitiamo ogni arco al più una volta: la prima volta che incontriamo una delle sue estremità
- ▶ Il tempo di calcolo quindi dipende da questi due valori e dalle operazioni di:

- ▶ Trovare il minimo tra i nodi non visitati
- ▶ Aggiornare la distanza dei nodi

Dipendono da come rappresentiamo il grafo e le altre strutture che ci servono

# CONSIDERAZIONI SUL TEMPO DI CALCOLO

- ▶ Se usiamo un array per mantenere le distanze di ogni nodo:
  - ▶ Ogni aggiornamento di distanza richiede  $O(1)$ , dato che è sufficiente cambiare il valore
  - ▶ Trovare il minimo richiede tempo  $O(V)$  perché l'array va scandito
  - ▶ Otteniamo quindi tempo  $O(V^2 + E) = O(V^2)$  perché per ogni nodo dobbiamo estrarre il minimo e per ogni arco aggiornare un peso

# CONSIDERAZIONI SUL TEMPO DI CALCOLO

- ▶ Se abbiamo  $E = o(V^2/\log V)$  possiamo usare un min-heap, (abbiamo usato un sistema simile nell'algoritmo di heapsort)
- ▶ Nel caso di min-heap:
  - ▶ Rimozione del minimo:  $O(\log V)$
  - ▶ Aggiornamento del peso:  $O(\log V)$
  - ▶ Otteniamo quindi  $O(V \log V + E \log V) = O((V + E)\log V)$  che è meglio dell'implementazione come array quando abbiamo  $o(V^2/\log V)$  archi