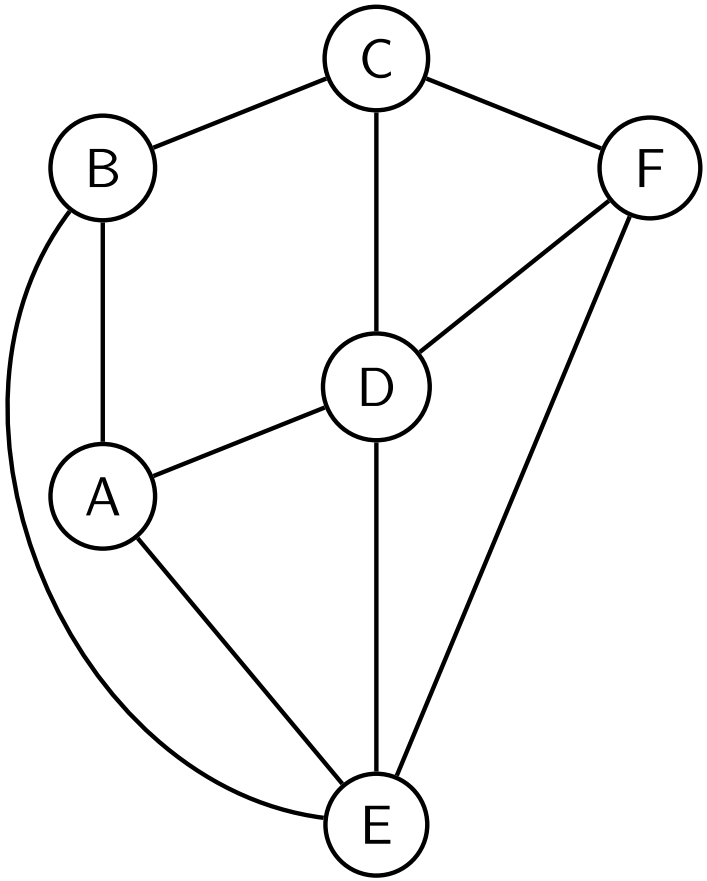
Grafi, visite e cammini minimi

Giada Franz, Edoardo Morassutto, Noemi Gambirasio

Volterra, 6 dicembre 2023

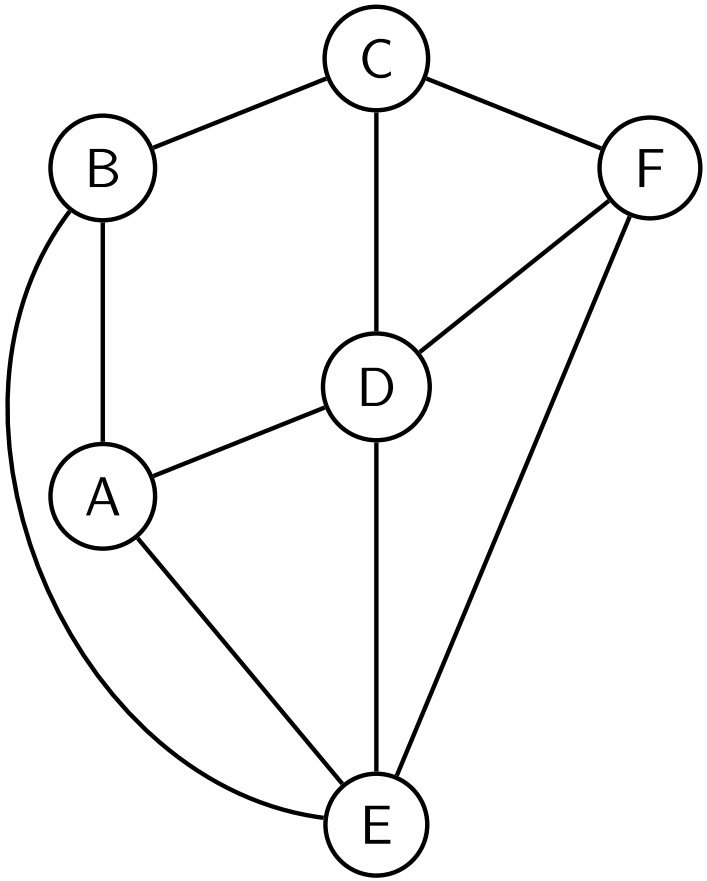
Cos'è un grafo?

Un insieme di N vertici



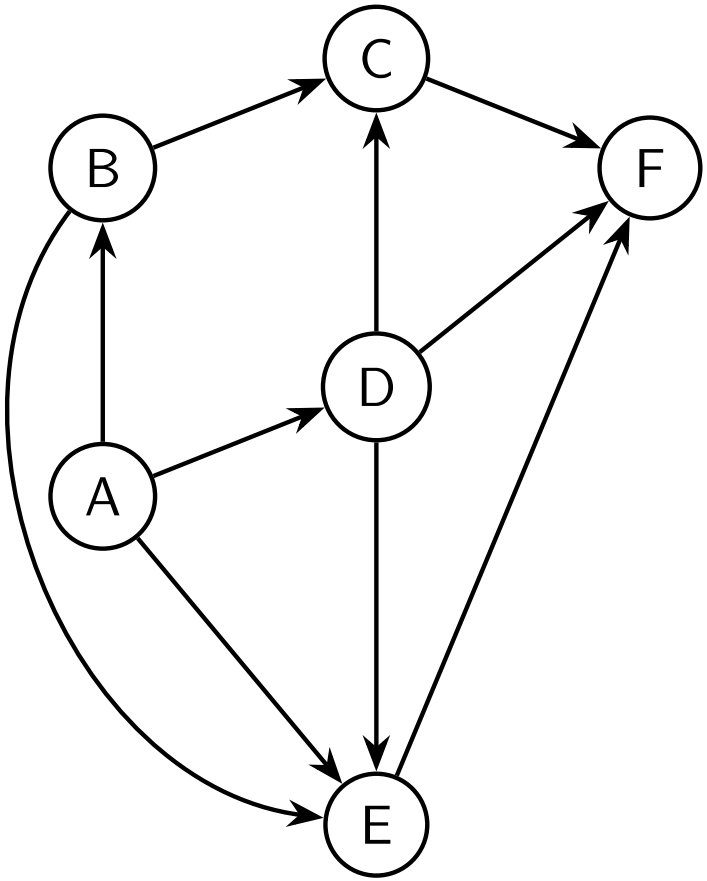
Cos'è un grafo?

Un insieme di N vertici e un insieme di M archi (coppie di nodi)



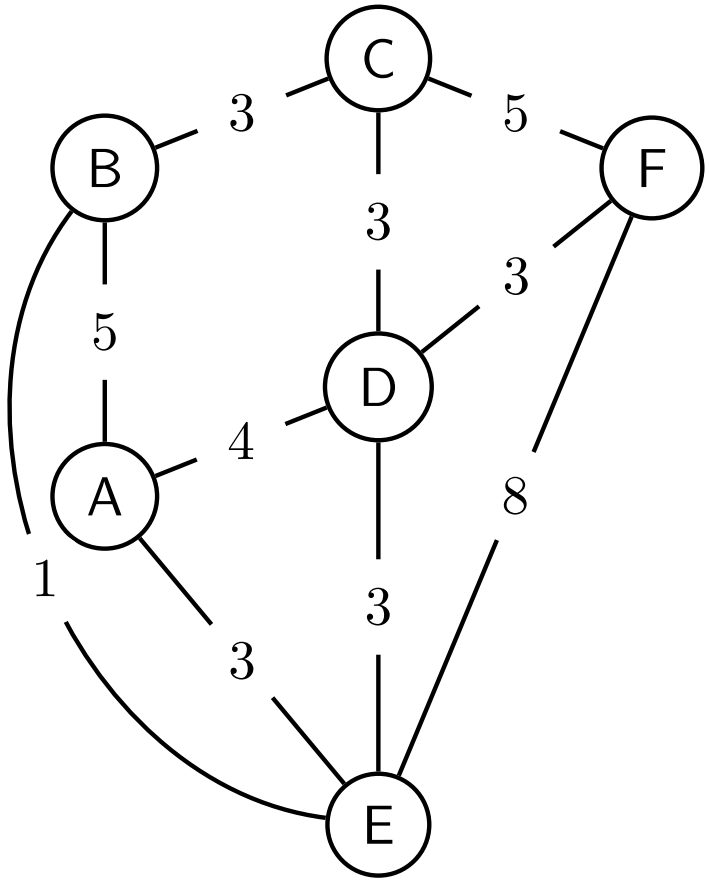
Cos'è un grafo?

Un insieme di N vertici e un insieme di M archi (coppie di nodi), eventualmente orientati



Cos'è un grafo?

Un insieme di N vertici e un insieme di M archi (coppie di nodi), eventualmente orientati e/o pesati.



Come memorizzare un grafo

Esistono principalmente 3 modi per memorizzare un grafo:

- Liste di adiacenza

Come memorizzare un grafo

Esistono principalmente 3 modi per memorizzare un grafo:

- Liste di adiacenza
- Matrice di adiacenza

Come memorizzare un grafo

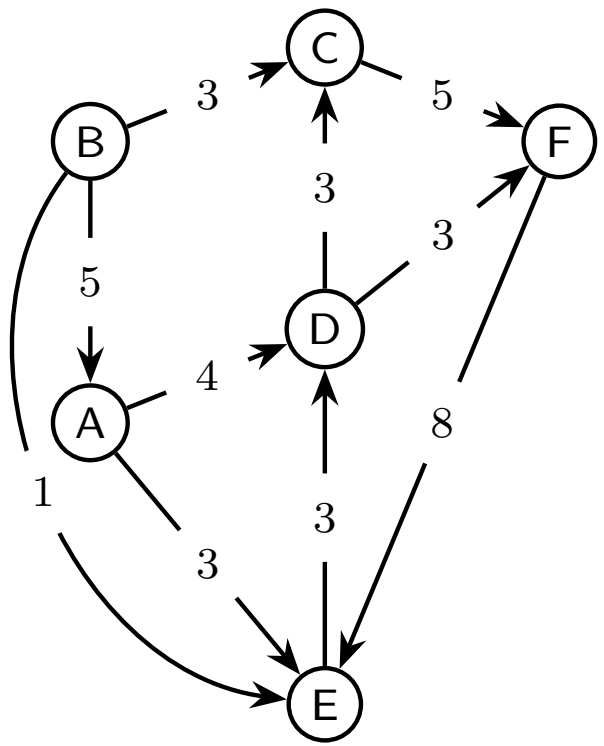
Esistono principalmente 3 modi per memorizzare un grafo:

- Liste di adiacenza
- Matrice di adiacenza
- Lista di archi

Liste di adiacenza

Definizione

Dato un grafo, memorizziamo, per ciascun nodo, una lista con tutti gli archi che partono da quel nodo.

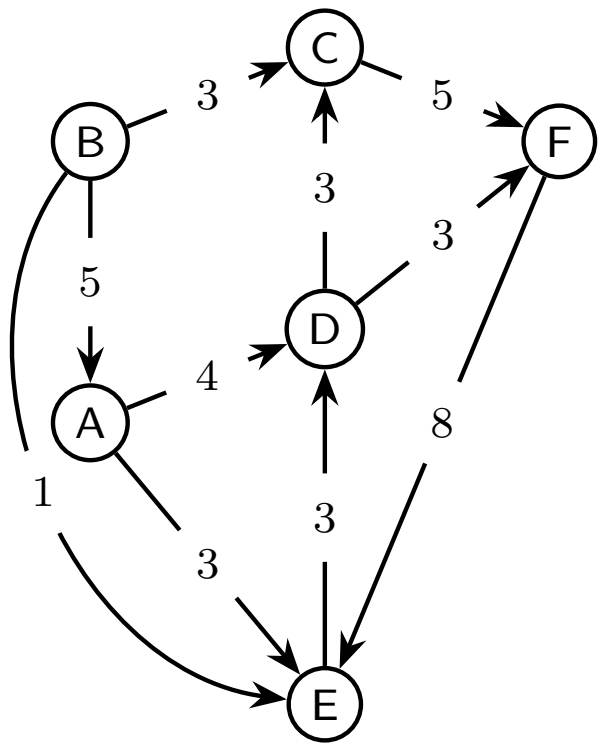


$$A \to \{(D,4),(E,3)\}$$
$$B \to \{(A,5),(C,3),(E,1)\}$$
$$C \to \{(F,5)\}$$
$$D \to \{(C,3),(F,3)\}$$
$$E \to \{(D,3)\}$$
$$F \to \{(E,8)\}$$

Liste di adiacenza

Definizione

Dato un grafo, memorizziamo, per ciascun nodo, una lista con tutti gli archi che partono da quel nodo.



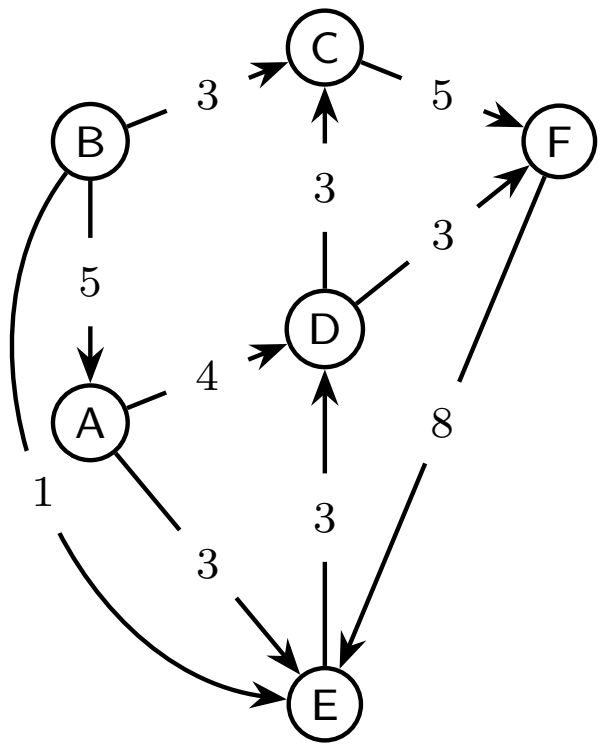
$$A \to \{(D,4),(E,3)\}$$
$$B \to \{(A,5),(C,3),(E,1)\}$$
$$C \to \{(F,5)\}$$
$$D \to \{(C,3),(F,3)\}$$
$$E \to \{(D,3)\}$$
$$F \to \{(E,8)\}$$

Cosa cambia se non ci sono i pesi?

Liste di adiacenza

Definizione

Dato un grafo, memorizziamo, per ciascun nodo, una lista con tutti gli archi che partono da quel nodo.



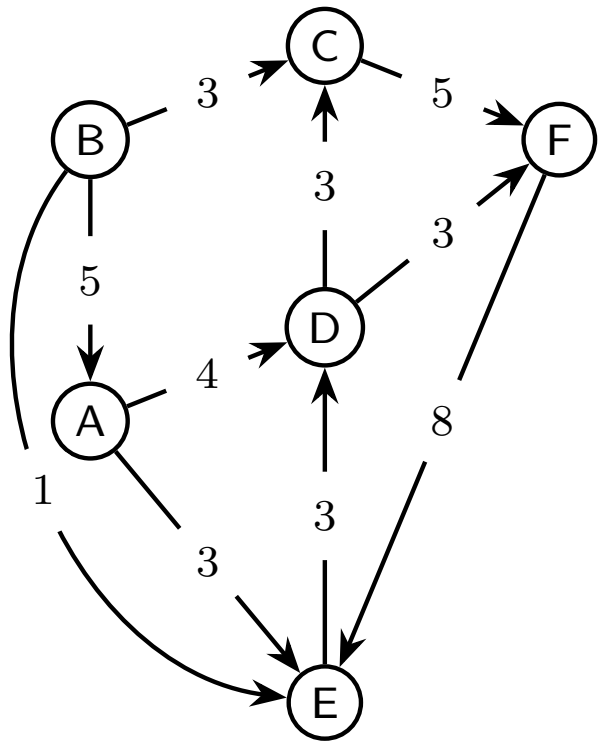
$$A \to \{(D,4),(E,3)\}$$
$$B \to \{(A,5),(C,3),(E,1)\}$$
$$C \to \{(F,5)\}$$
$$D \to \{(C,3),(F,3)\}$$
$$E \to \{(D,3)\}$$
$$F \to \{(E,8)\}$$

Cosa cambia se non ci sono i pesi?
Cosa cambia se il grafo non è diretto?

Matrice di adiacenza

Definizione

Dato un grafo con N nodi, memorizziamo in una matrice $N \times N$, in posizione (i, j), informazioni sull'arco $i \to j$.

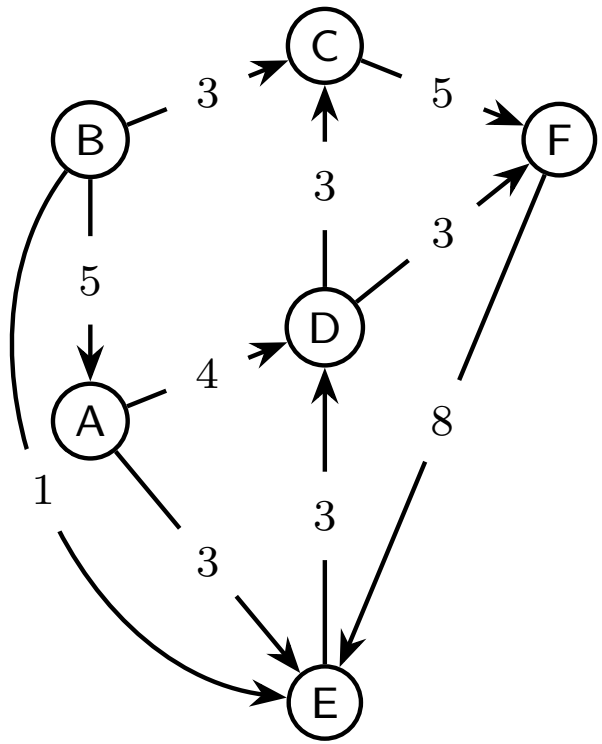


↗	A	B	C	D	E	F
A				4	3	
B	5		3		1	
C						5
D			3			3
E				3		
F					8	

Matrice di adiacenza

Definizione

Dato un grafo con N nodi, memorizziamo in una matrice $N \times N$, in posizione (i, j), informazioni sull'arco $i \to j$.



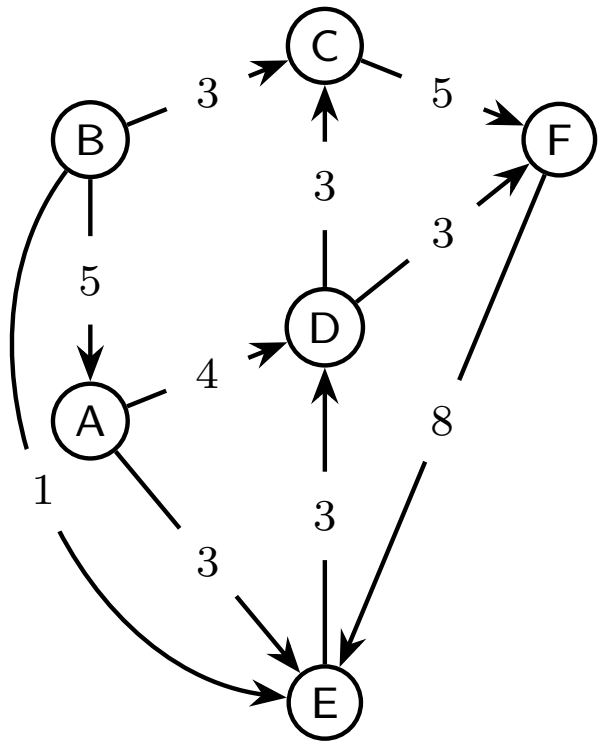
↗	A	B	C	D	E	F
A				4	3	
B	5		3		1	
C						5
D			3			3
E				3		
F					8	

Cosa mettere nelle celle vuote?

Matrice di adiacenza

Definizione

Dato un grafo con N nodi, memorizziamo in una matrice $N \times N$, in posizione (i, j), informazioni sull'arco $i \to j$.



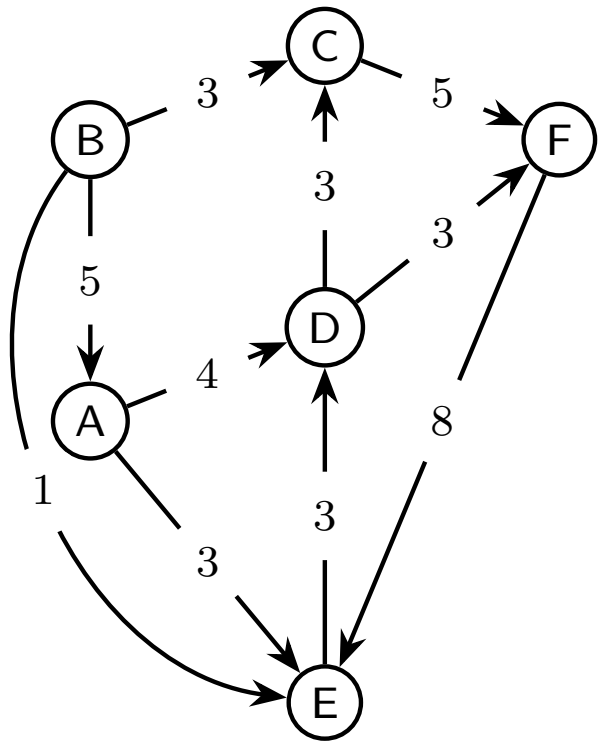
↗	A	B	C	D	E	F
A				4	3	
B	5		3		1	
C						5
D			3			3
E				3		
F					8	

Cosa mettere nelle celle vuote?
Cosa cambia se non ci sono i pesi?

Matrice di adiacenza

Definizione

Dato un grafo con N nodi, memorizziamo in una matrice $N \times N$, in posizione (i, j), informazioni sull'arco $i \to j$.



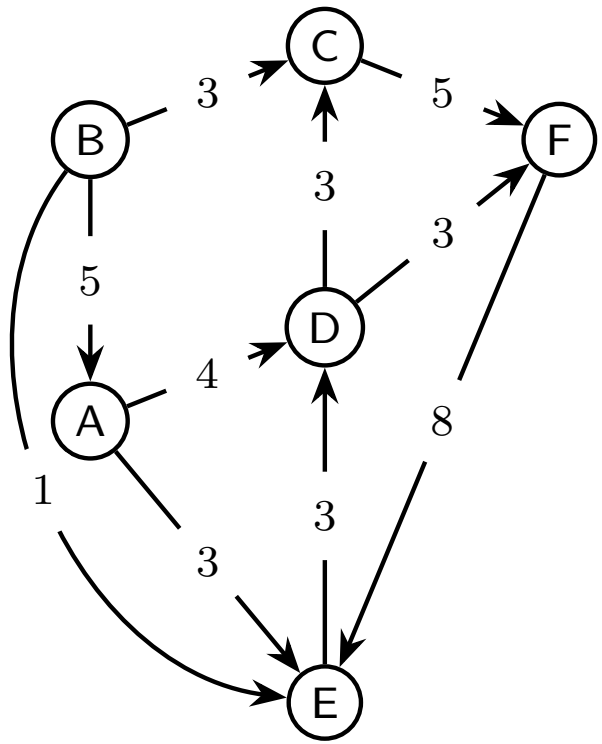
↗	A	B	C	D	E	F
A				4	3	
B	5		3		1	
C						5
D			3			3
E				3		
F					8	

Cosa mettere nelle celle vuote?
Cosa cambia se non ci sono i pesi?
Cosa cambia se il grafo non è diretto?

Lista di archi

Definizione

Dato un grafo con M archi, memorizziamo in un vettore la lista di tutti gli M archi.



$$
\begin{aligned}
E = \{&(A, D, 4),\\
&(A, E, 3),\\
&(B, A, 5),\\
&(B, C, 3),\\
&(B, E, 1),\\
&(C, F, 5),\\
&(D, C, 3),\\
&(D, F, 3),\\
&(E, D, 3),\\
&(F, E, 8)\}
\end{aligned}
$$

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste			

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*		

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo			

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$		

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi			

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$		

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco			

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$		

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco			

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*		

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Memoria usata			

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Memoria usata	$\mathcal{O}(N+M)$		

m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Memoria usata	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	

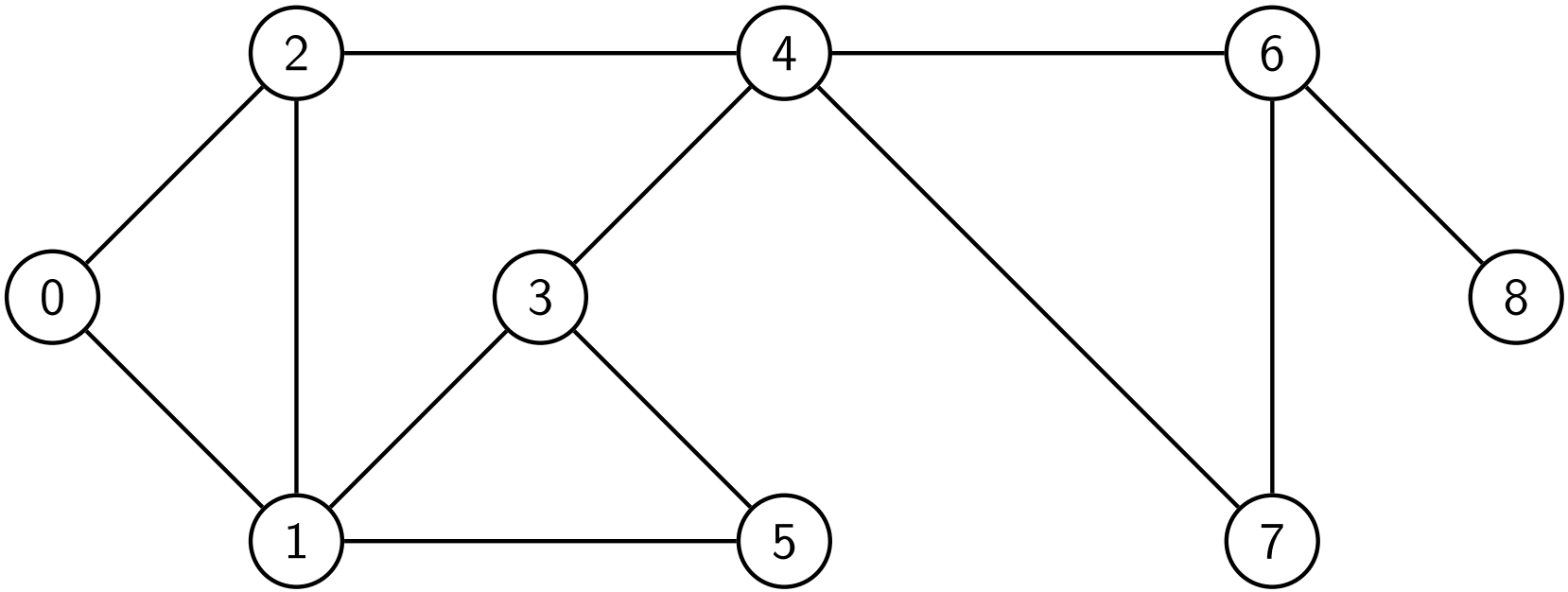
m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

Confronto tra le tre strategie

	Adj list	Adj mat	Edge list
Sapere se un arco esiste	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Iterare archi di un nodo	$\mathcal{O}(m)$	$\mathcal{O}(N)$	$\mathcal{O}(M)$*
Iterare tutti gli archi	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$
Aggiungere un (nuovo) arco	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$
Rimuovere un arco	$\mathcal{O}(m)$*	$\mathcal{O}(1)$	$\mathcal{O}(M)$*
Memoria usata	$\mathcal{O}(N+M)$	$\mathcal{O}(N^2)$	$\mathcal{O}(M)$

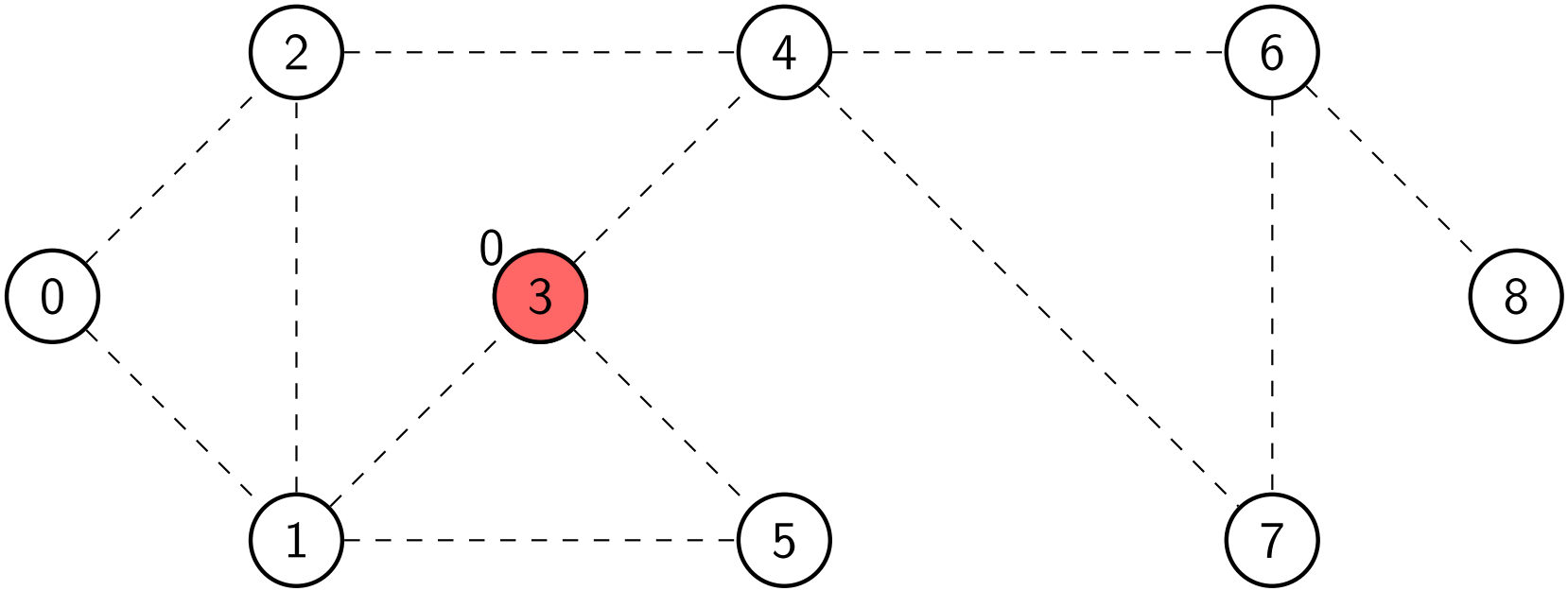
m numero di archi del nodo interessato
* si può fare meglio usando delle strutture dati

BFS o visita in ampiezza



BFS o visita in ampiezza

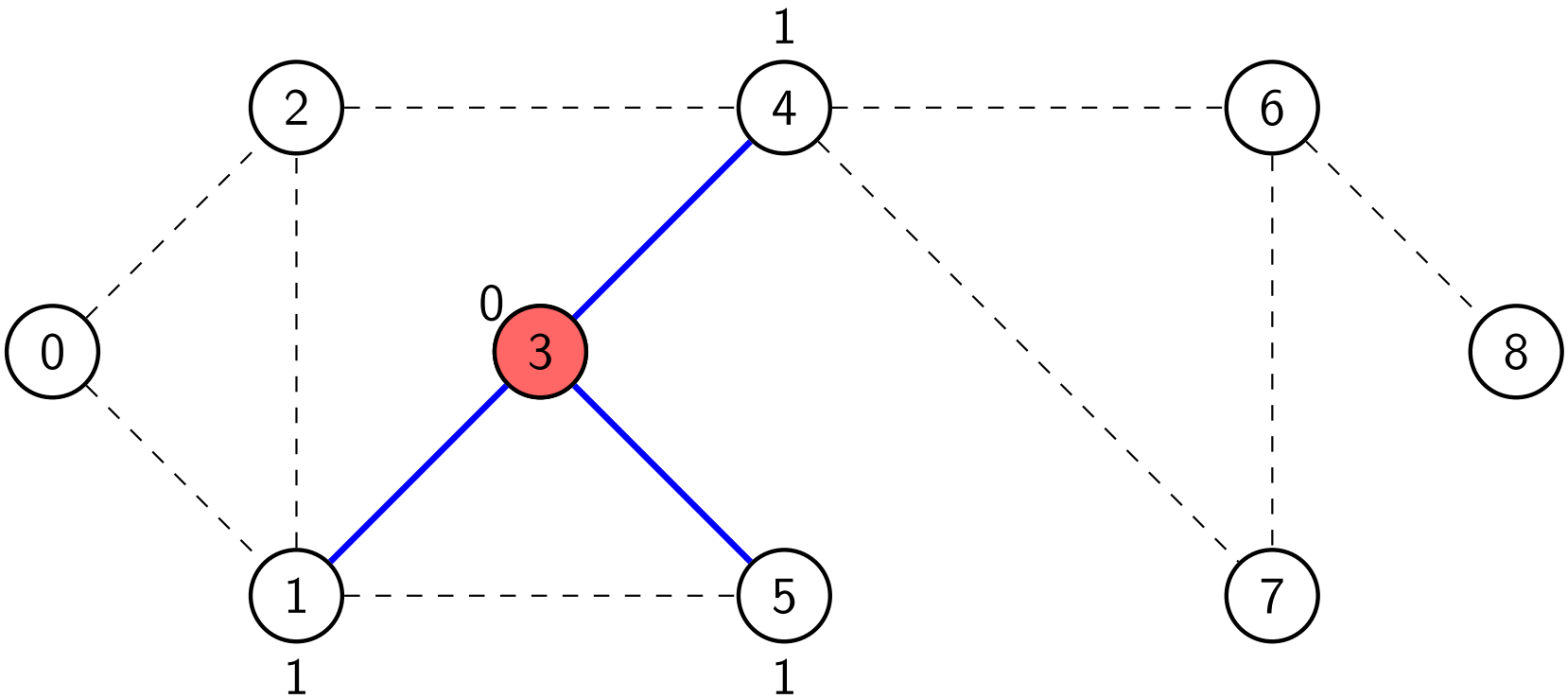
Inizia da un nodo ed imposta la sua distanza a 0.



$Q = \{\underline{3}\}$

BFS o visita in ampiezza

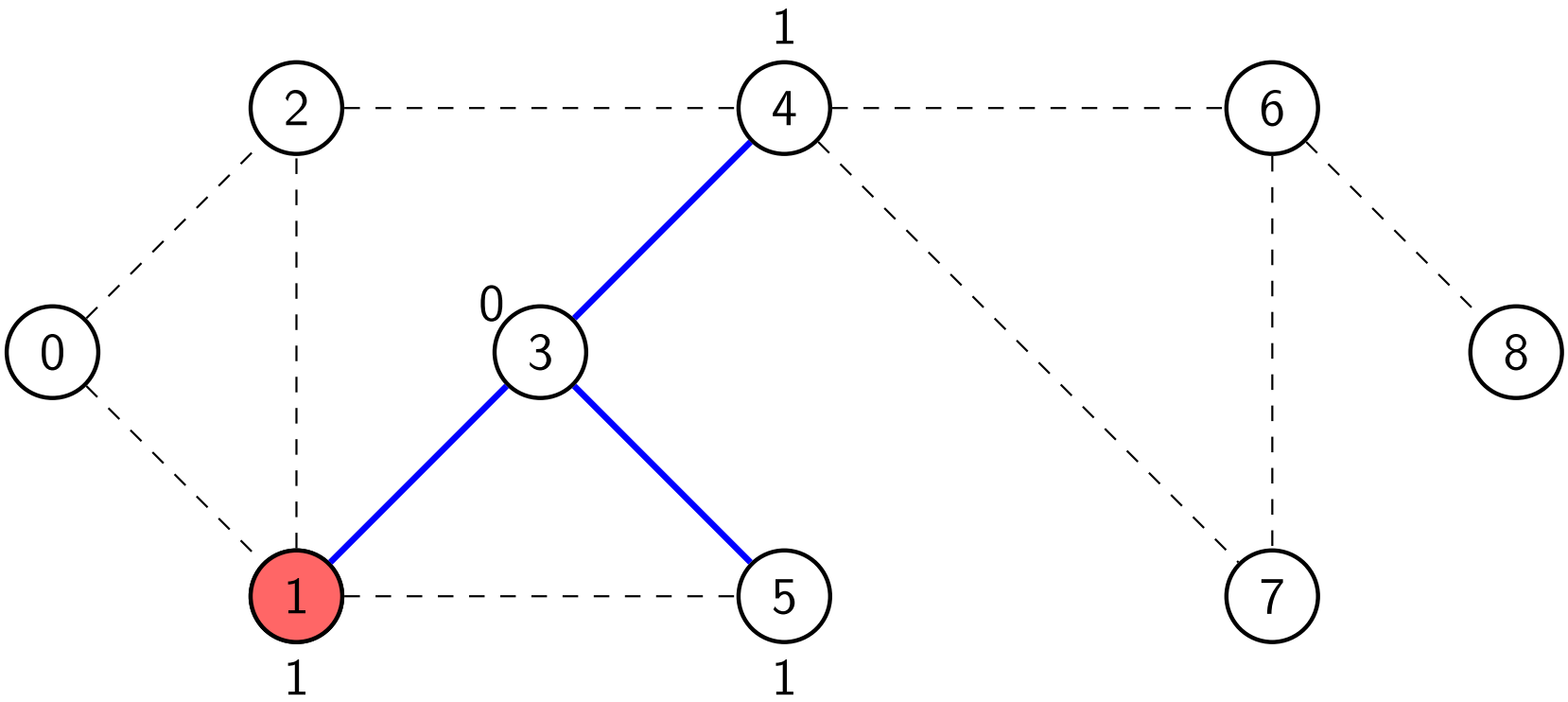
Espandi il nodo *visitando* tutti i suoi vicini, aggiornando la loro distanza e accodandoli alla coda.



$Q = \{1, 4, 5\}$

BFS o visita in ampiezza

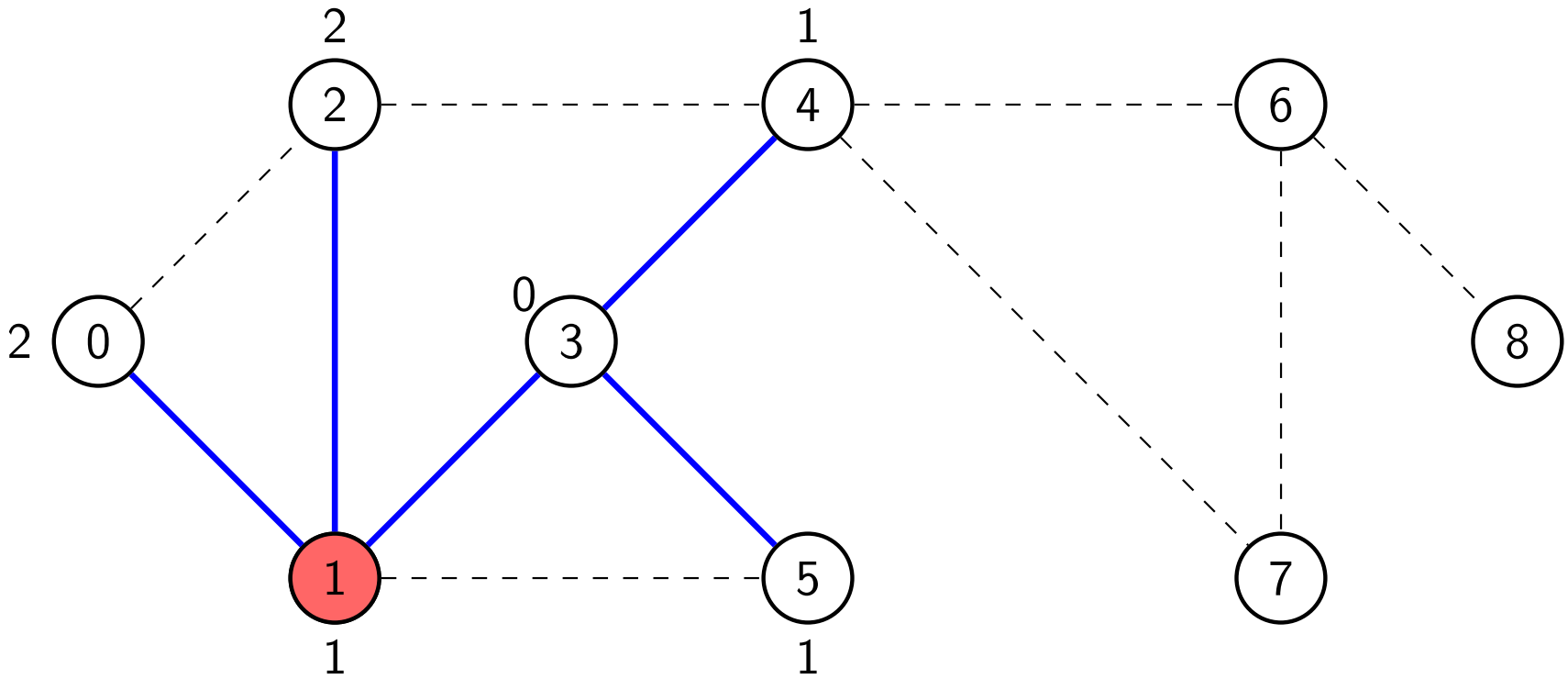
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{1}, 4, 5\}$

BFS o visita in ampiezza

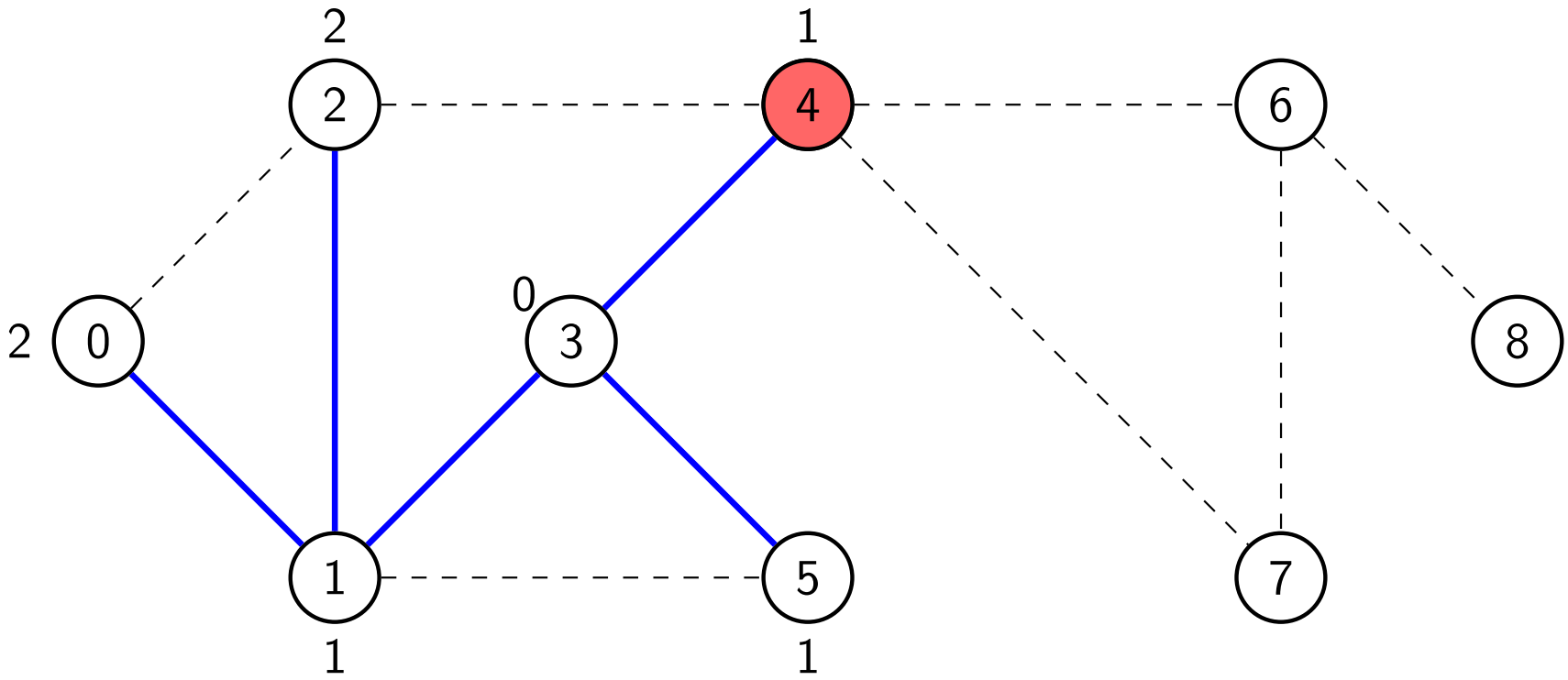
Espandi il nodo, visita i suoi vicini, aggiornando la loro distanza (se necessario!) e aggiungili alla coda (se hai aggiornato la distanza!)



$Q = \{4, 5, 0, 2\}$

BFS o visita in ampiezza

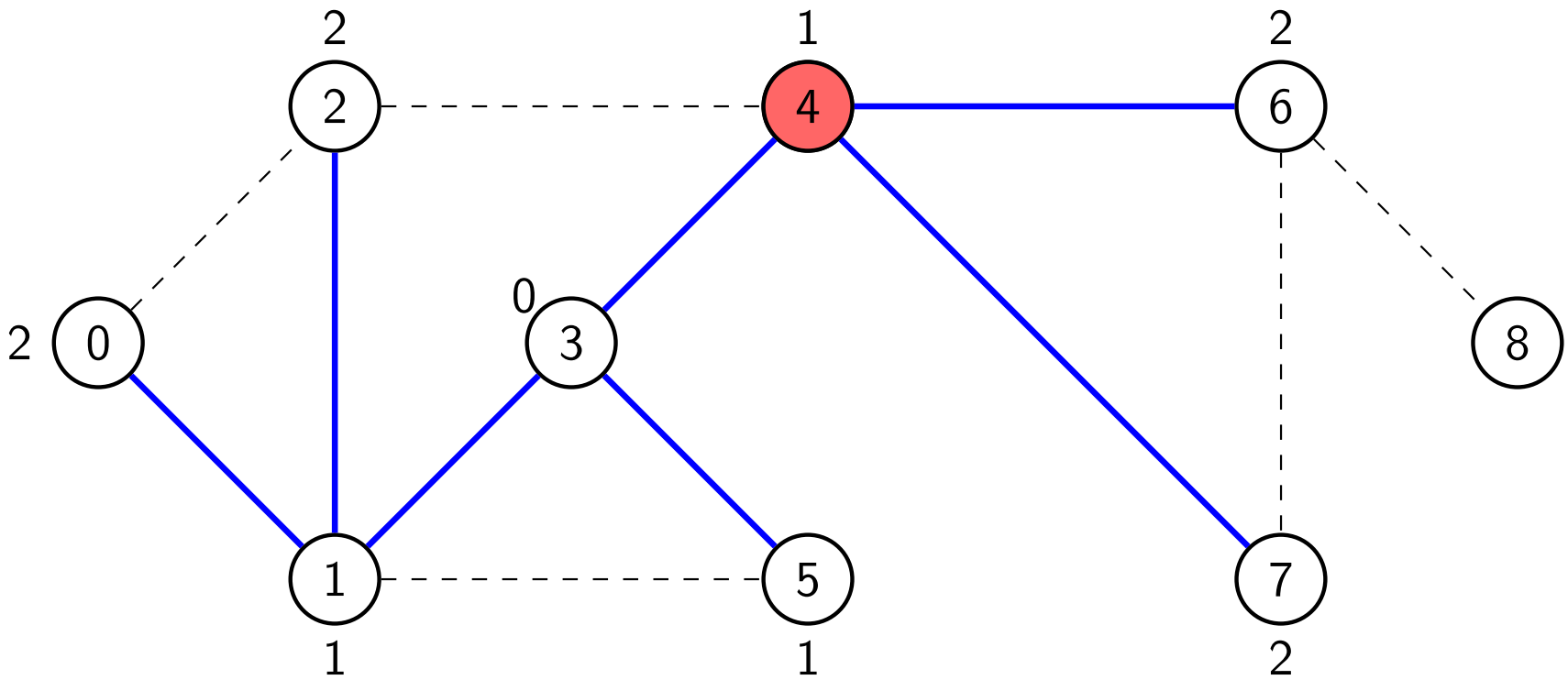
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{4}, 5, 0, 2\}$

BFS o visita in ampiezza

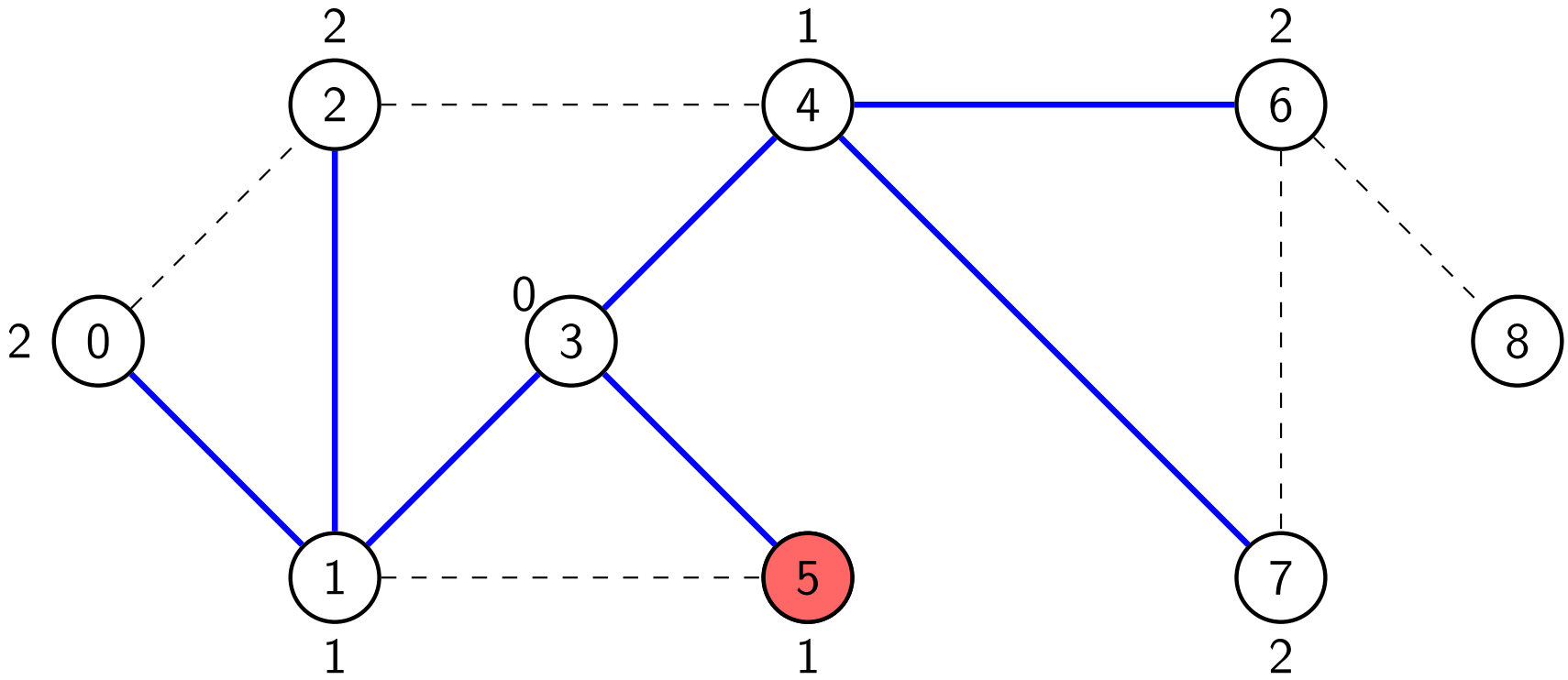
Espandi il nodo, visita i suoi vicini, aggiornando la loro distanza (se necessario!) e aggiungili alla coda (se hai aggiornato la distanza!)



$Q = \{5, 0, 2, 6, 7\}$

BFS o visita in ampiezza

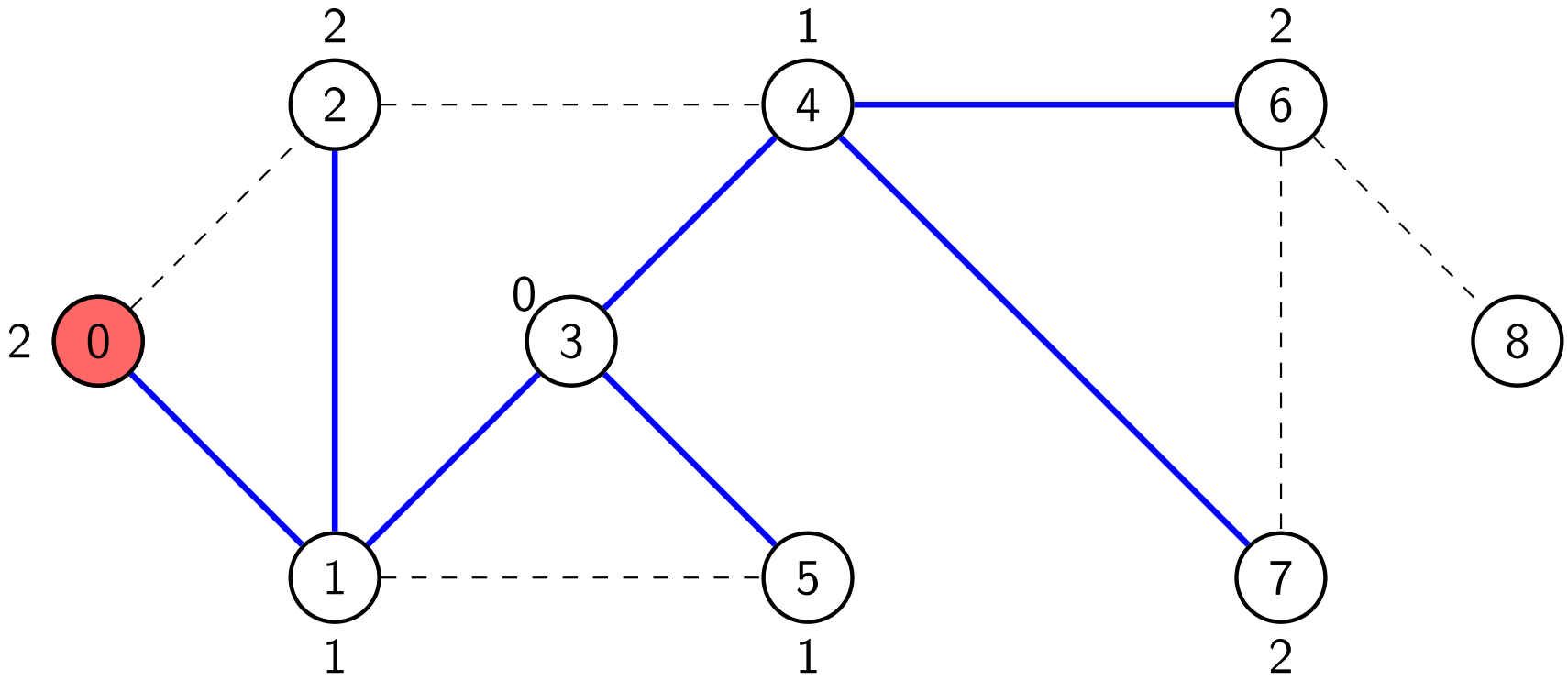
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{5}, 0, 2, 6, 7\}$

BFS o visita in ampiezza

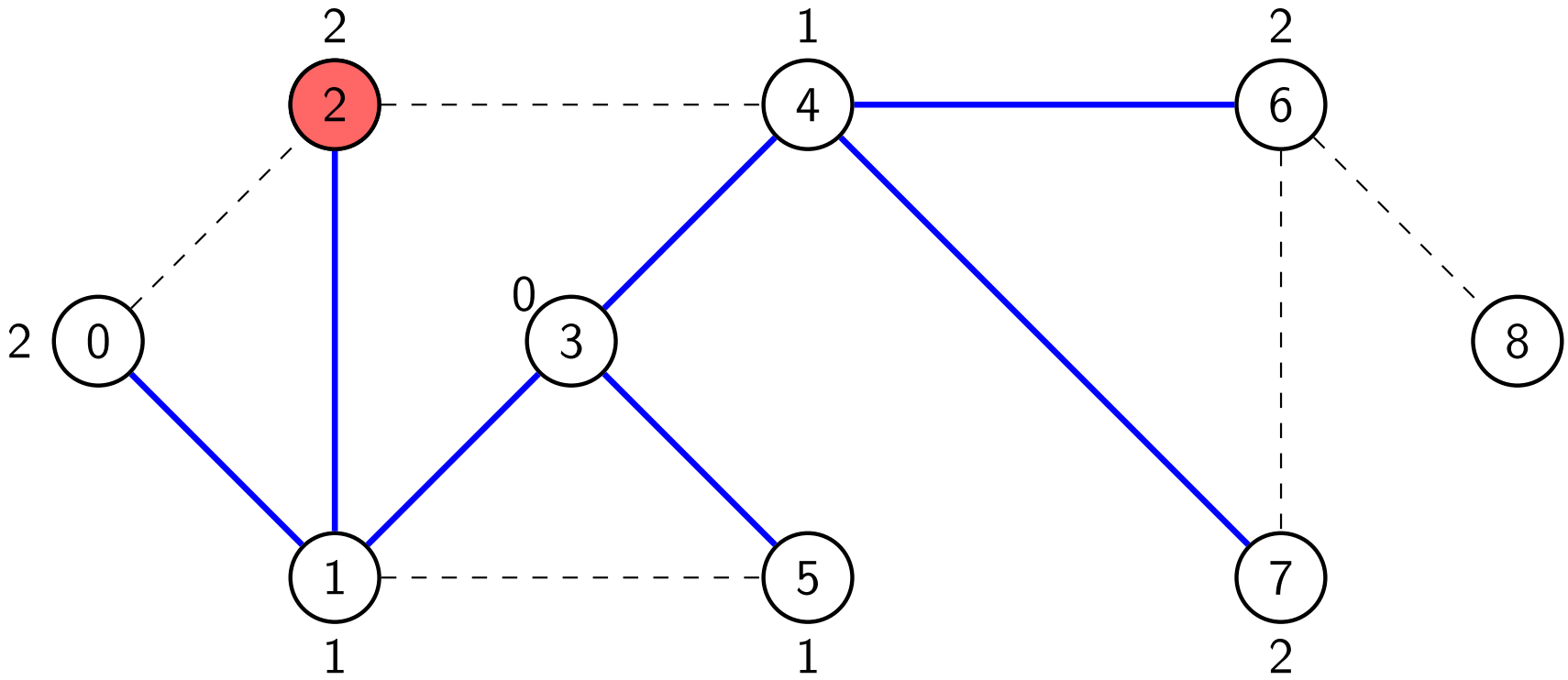
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{0}, 2, 6, 7\}$

BFS o visita in ampiezza

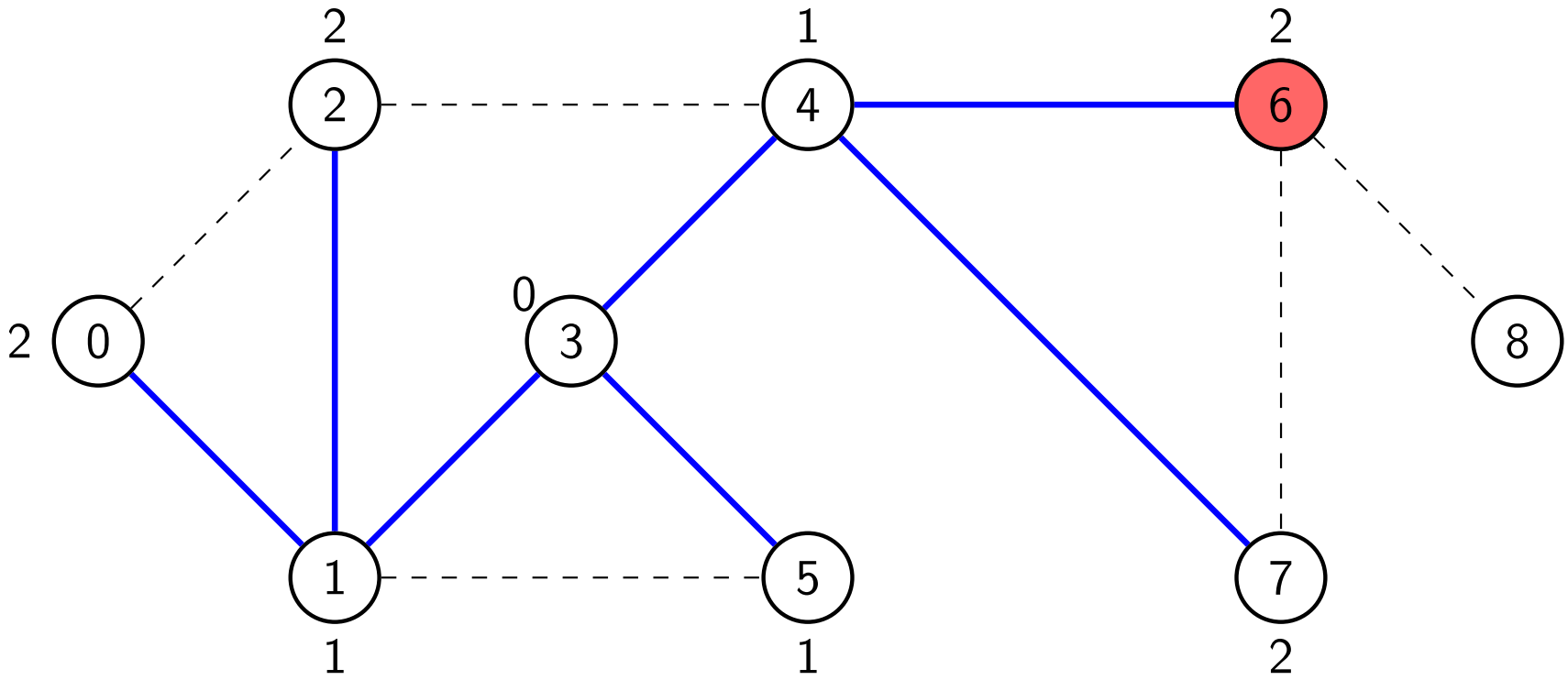
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{2}, 6, 7\}$

BFS o visita in ampiezza

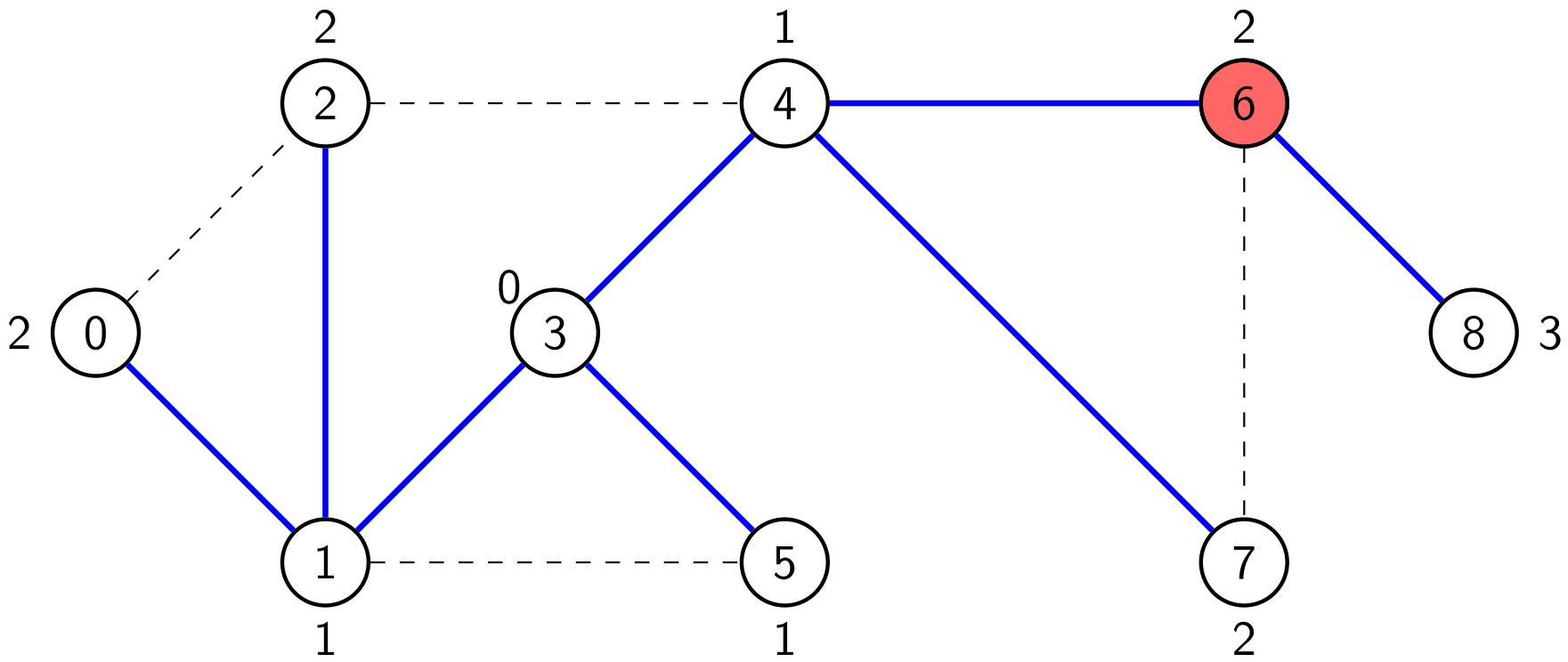
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{6}, 7\}$

BFS o visita in ampiezza

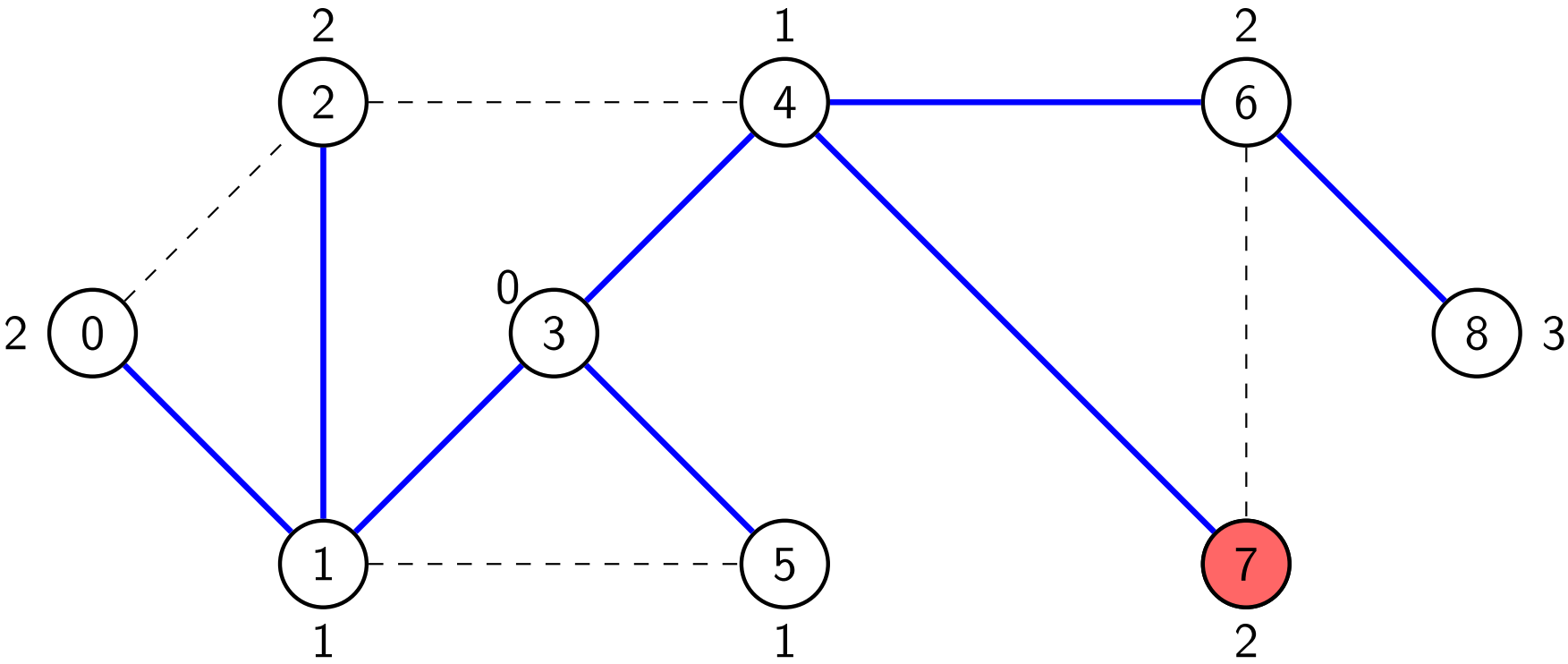
Espandi il nodo, visita i suoi vicini, aggiornando la loro distanza (se necessario!) e aggiungili alla coda (se hai aggiornato la distanza!)



$Q = \{7, 8\}$

BFS o visita in ampiezza

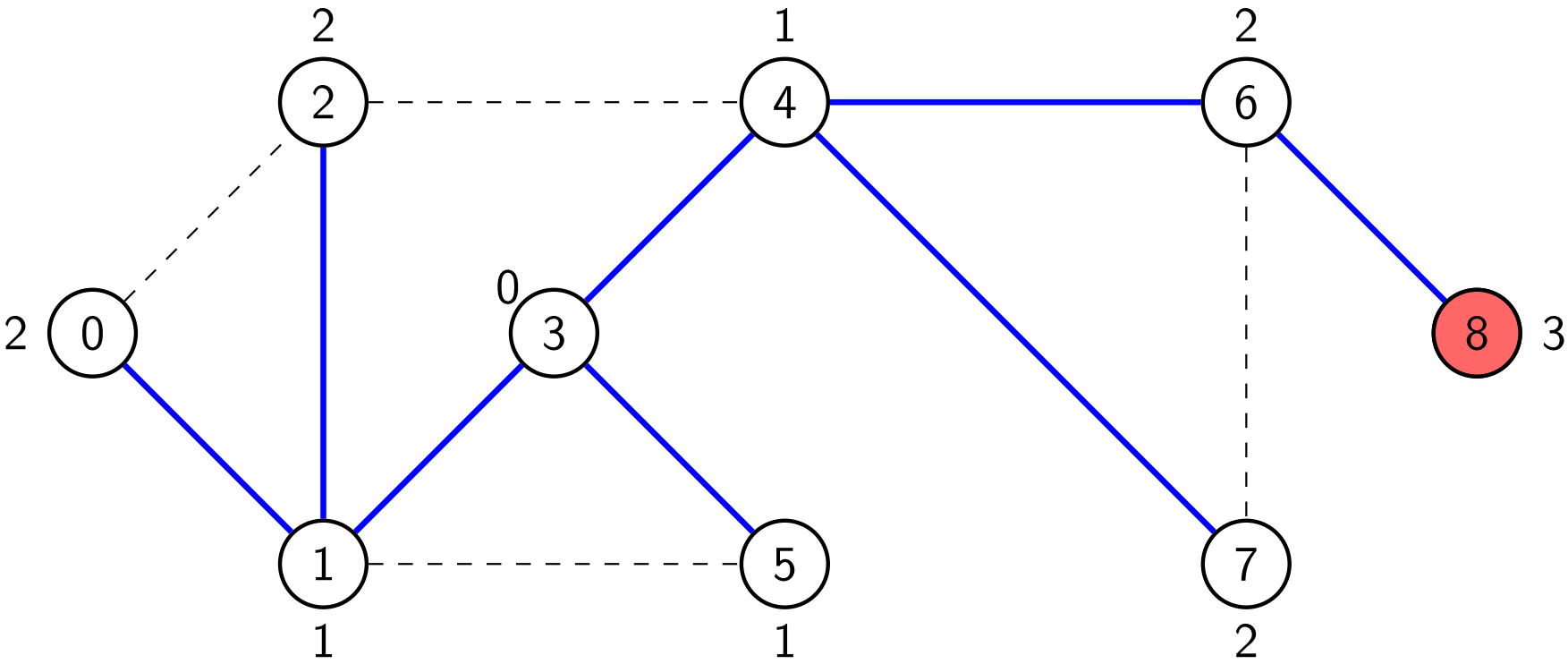
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{7}, 8\}$

BFS o visita in ampiezza

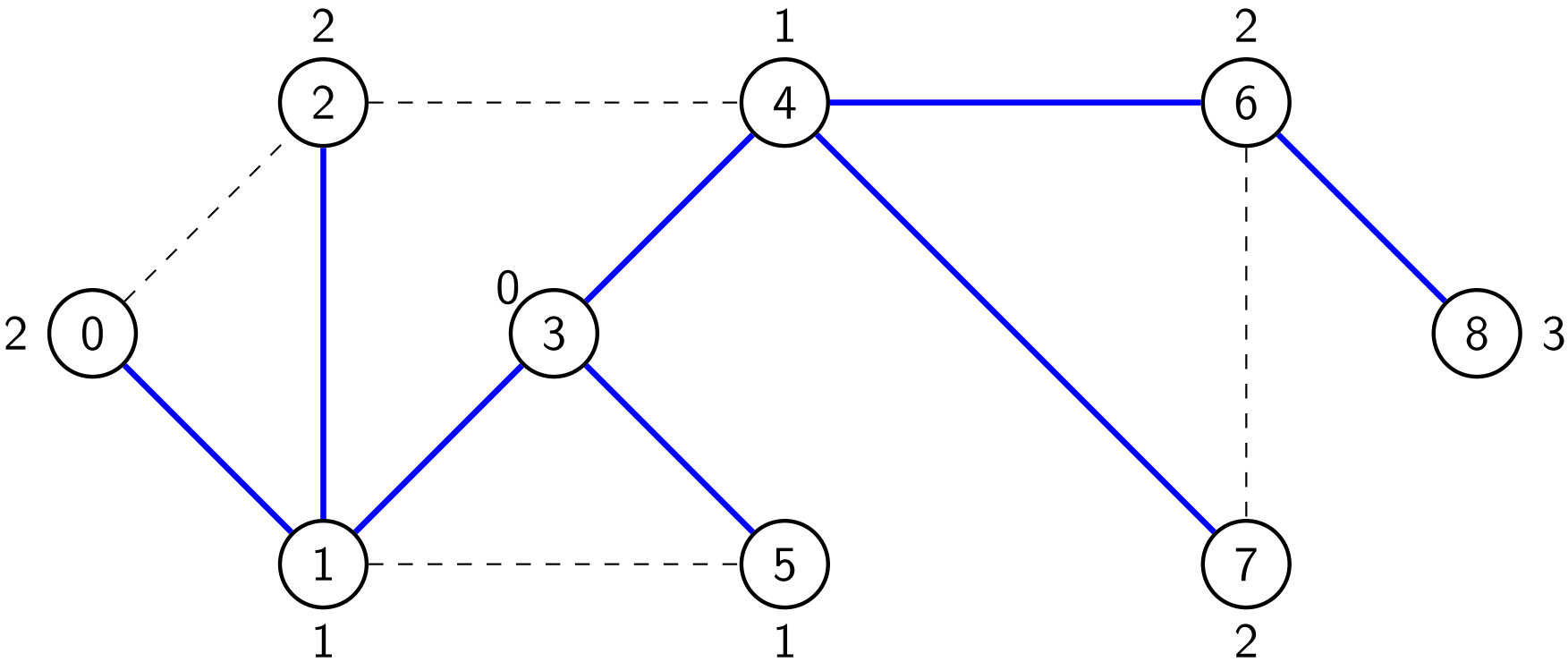
Seleziona il nodo successivo usando la coda.



$Q = \{\underline{8}\}$

BFS o visita in ampiezza

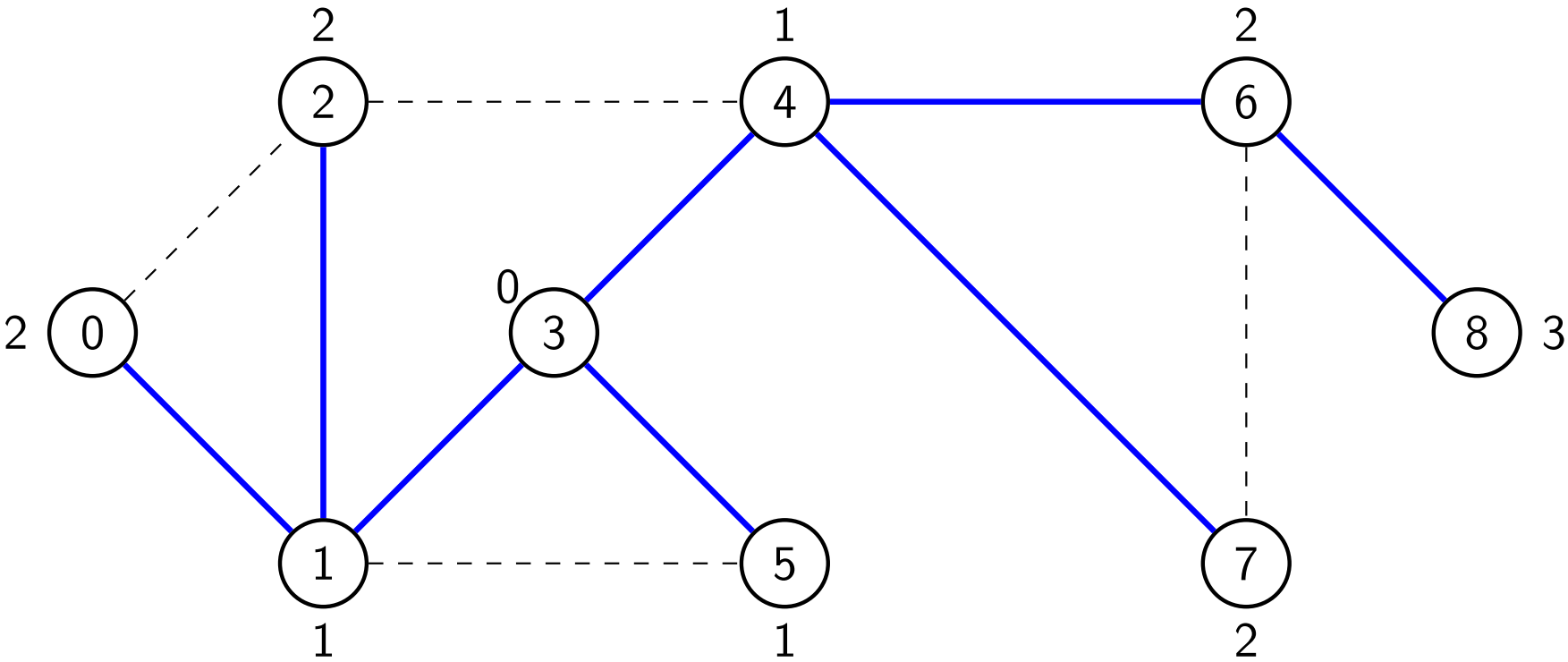
La coda è vuota, l'algoritmo termina.



$Q = \{\}$

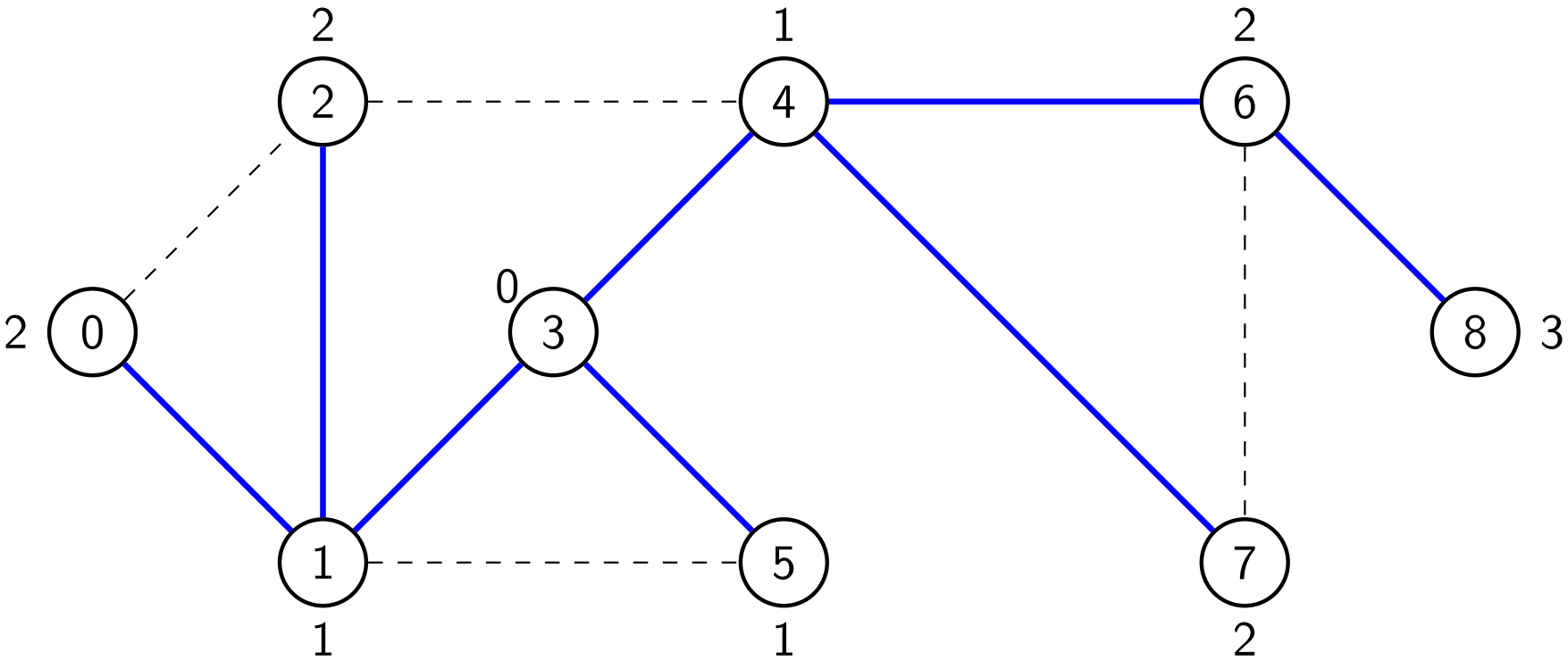
BFS o visita in ampiezza

- In quali condizioni i valori di distanza dei nodi sono *corretti*?

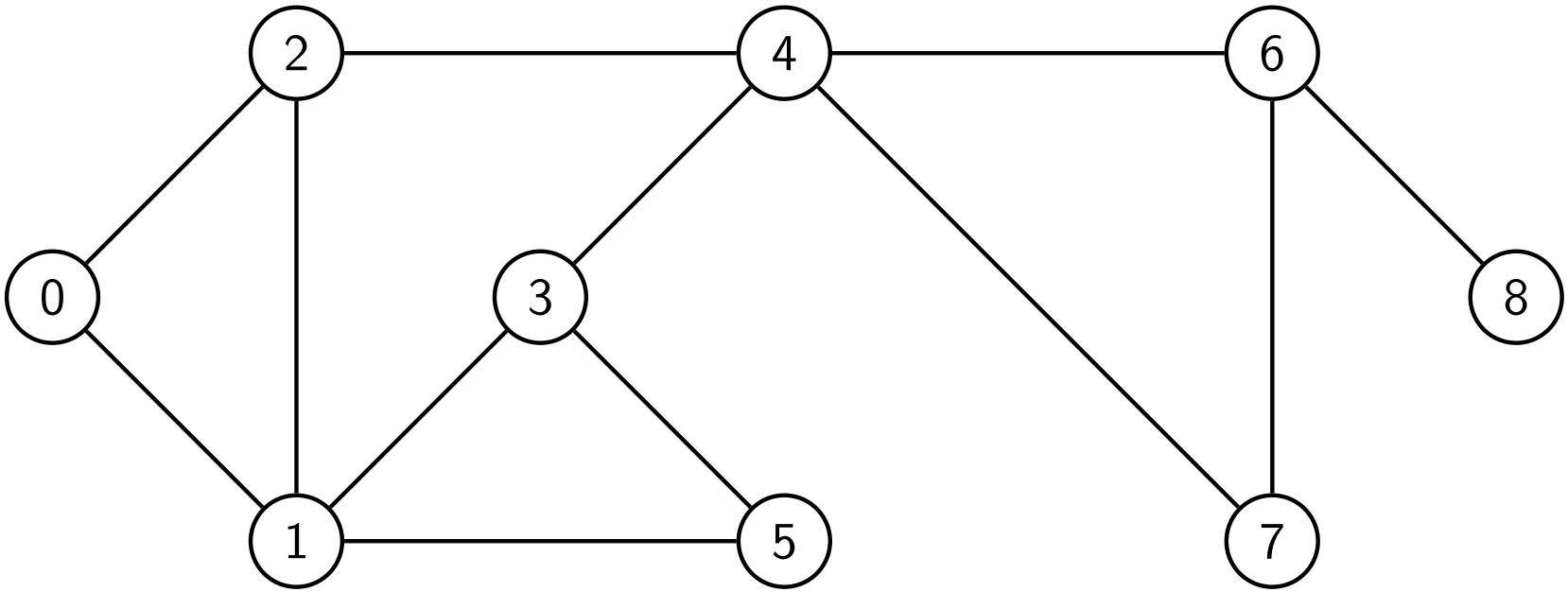


BFS o visita in ampiezza

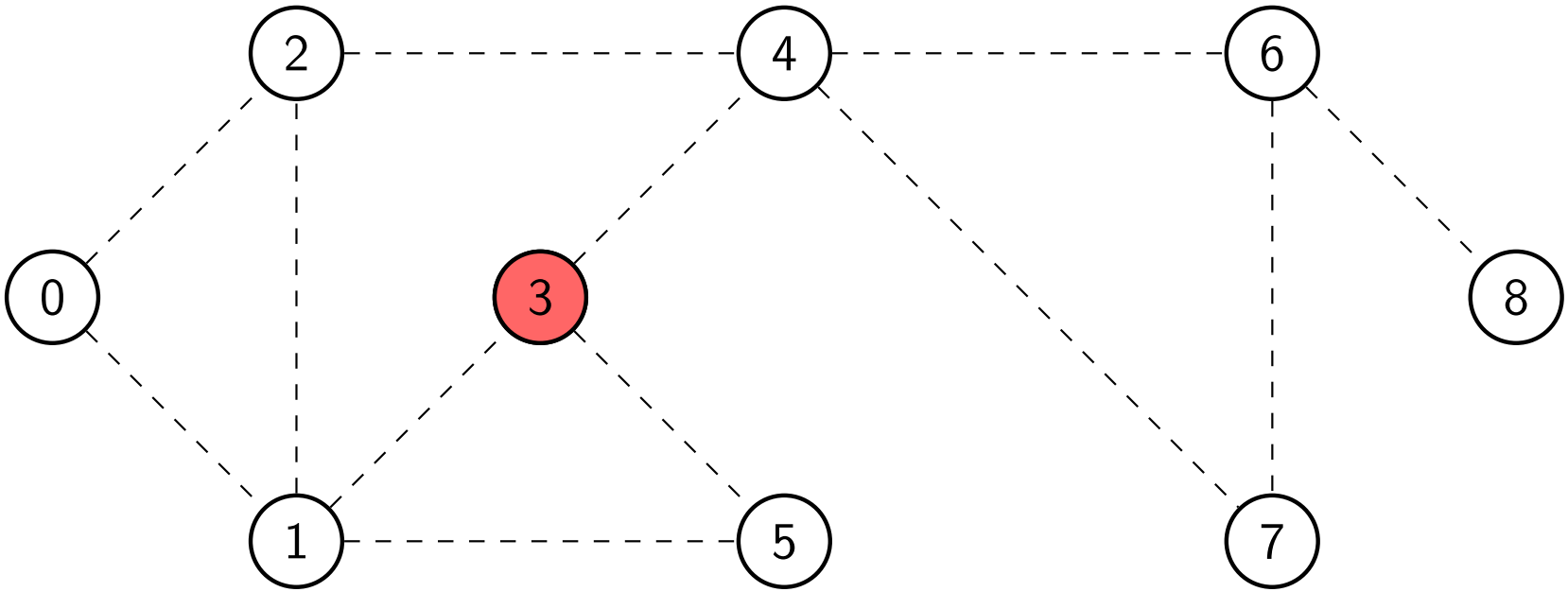
- In quali condizioni i valori di distanza dei nodi sono *corretti*?
- Qual è la complessità computazionale di questo algoritmo?



DFS o visita in profondità

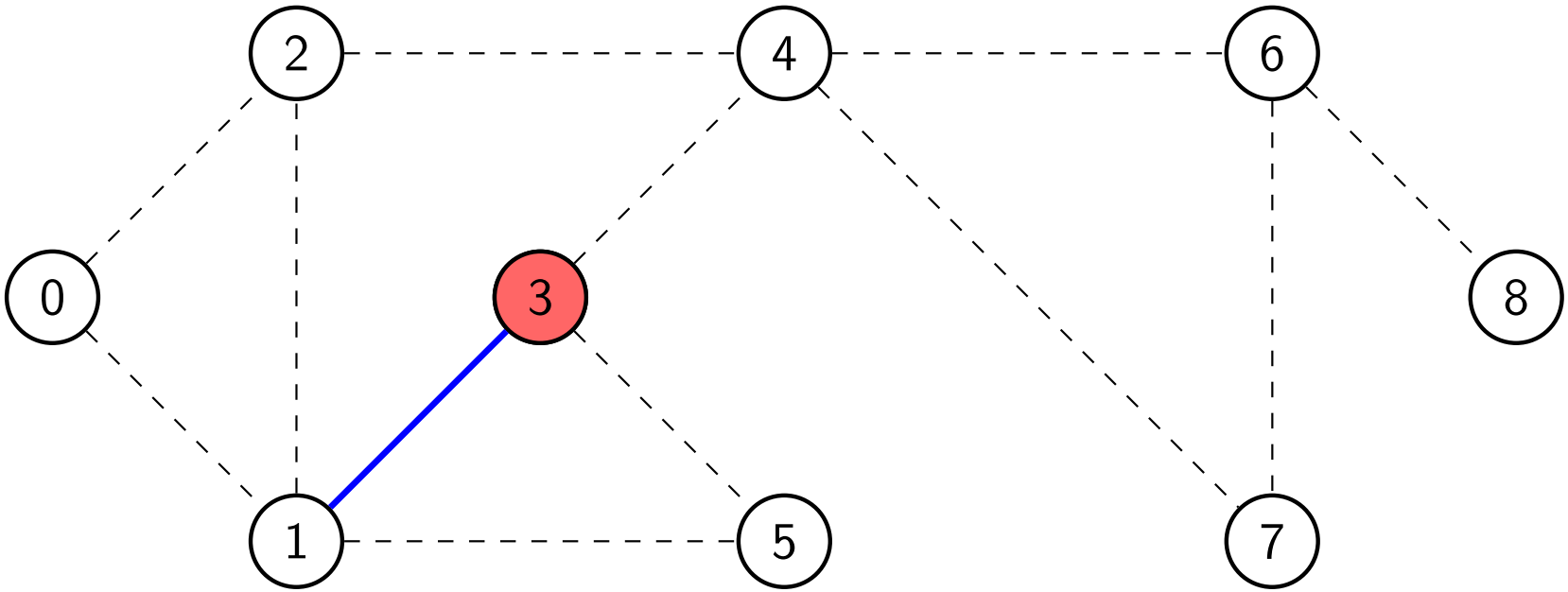


DFS o visita in profondità



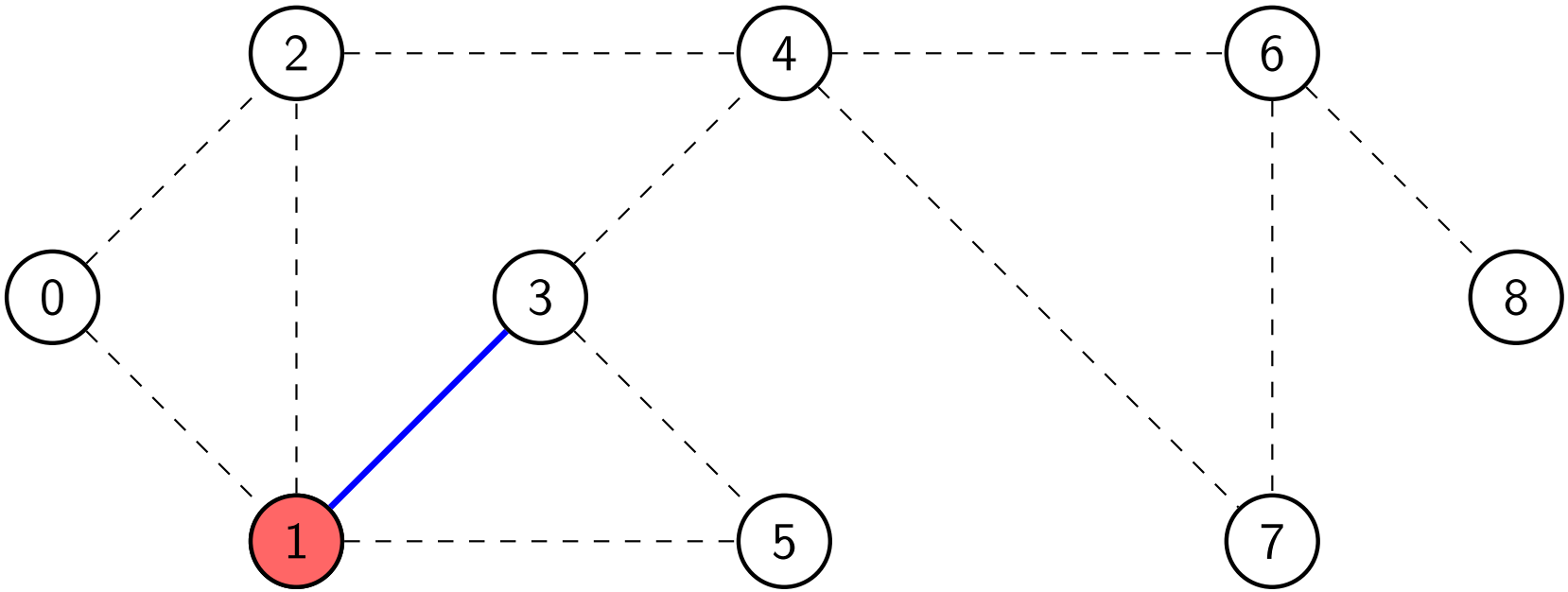
$S = \{3\}$

DFS o visita in profondità



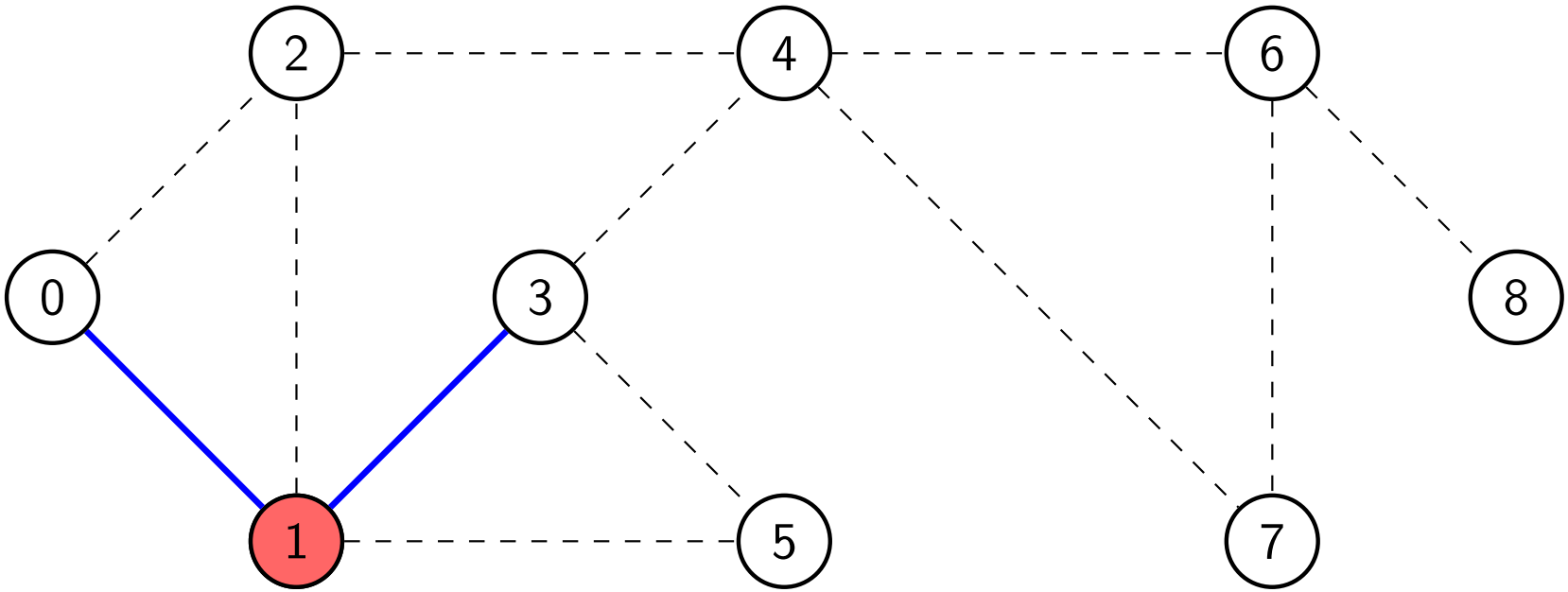
$S = \{\underline{3}\}$

DFS o visita in profondità



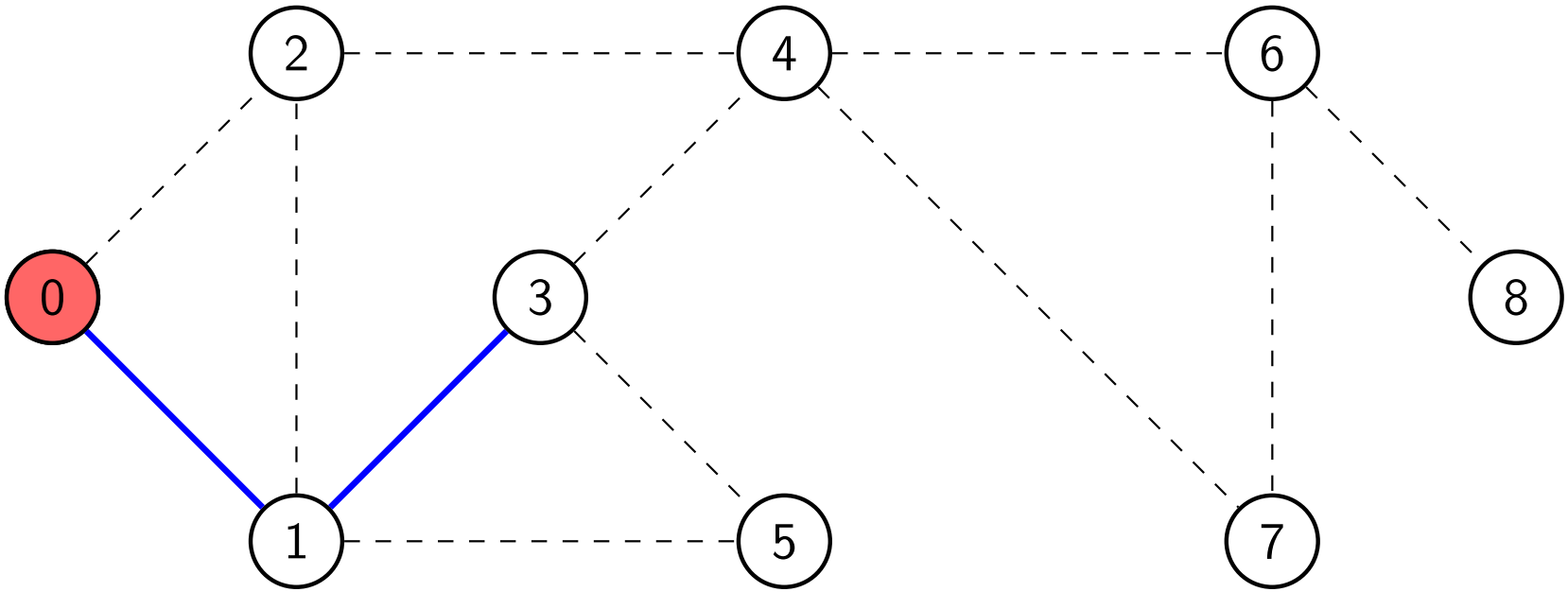
$S = \{3, 1\}$

DFS o visita in profondità



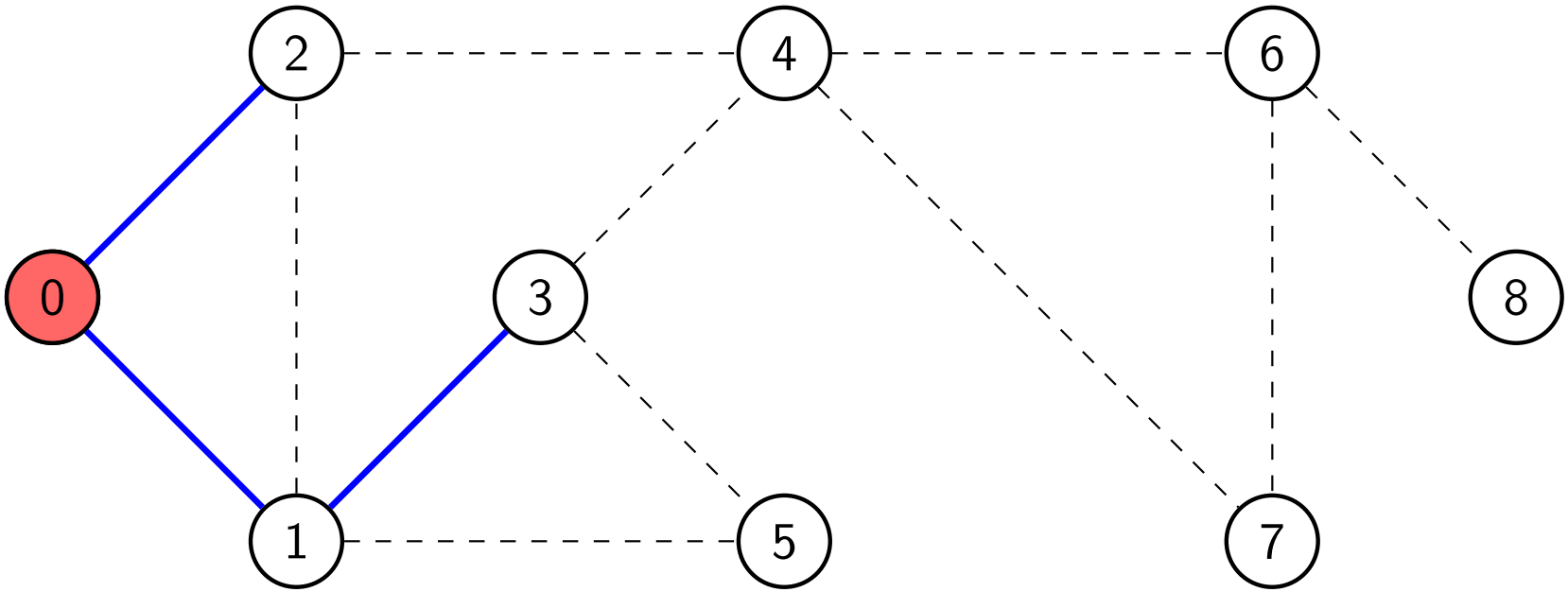
$S = \{3, \underline{1}\}$

DFS o visita in profondità



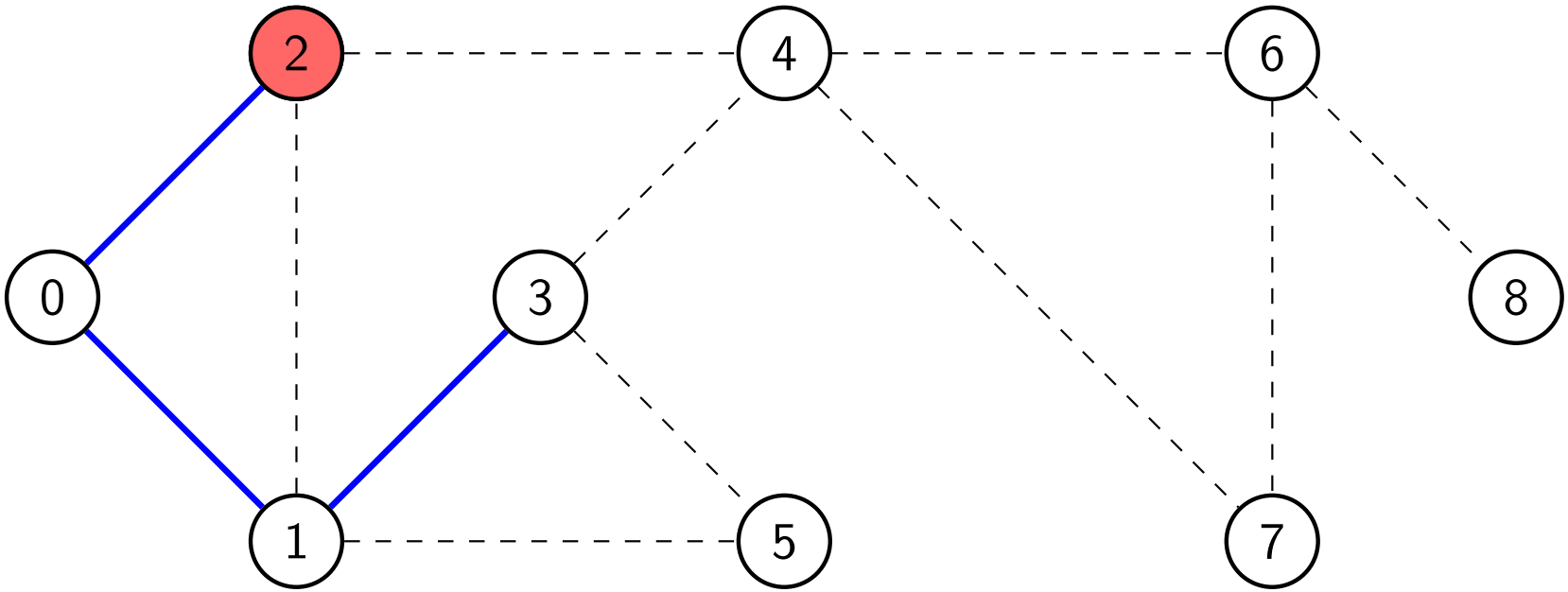
$S = \{3, 1, 0\}$

DFS o visita in profondità



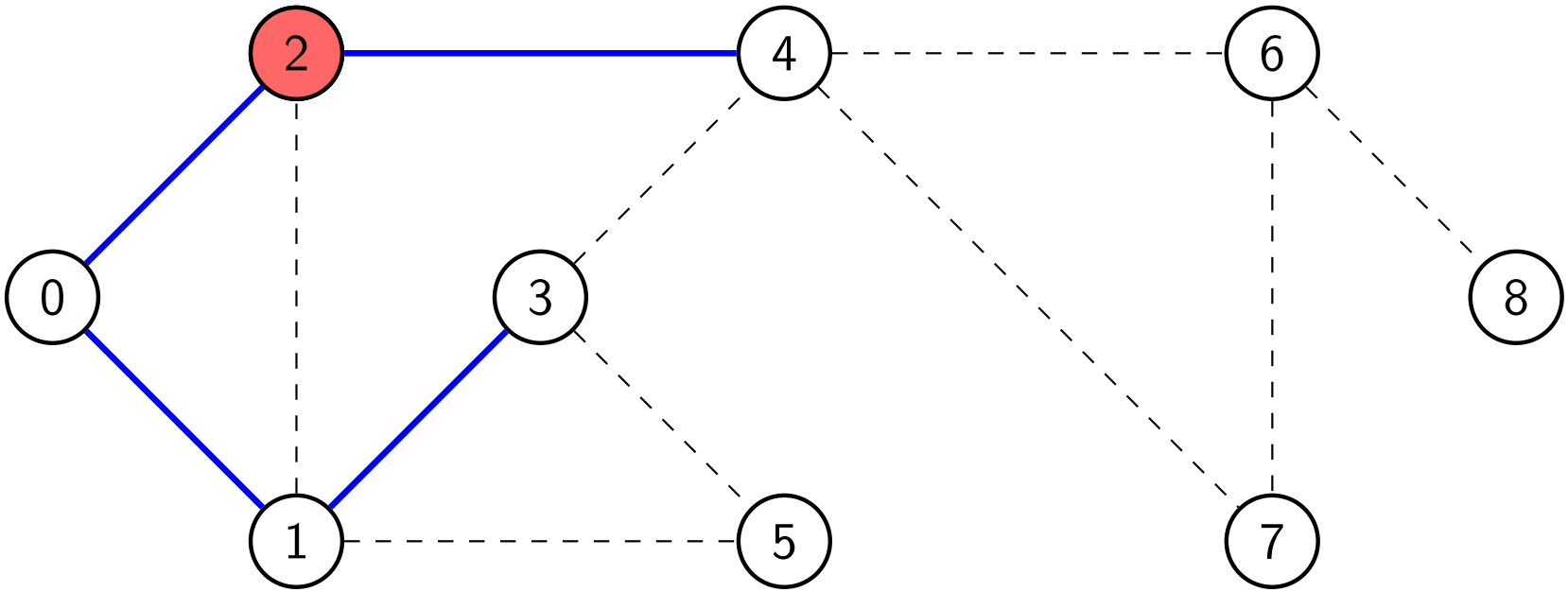
$S = \{3, 1, \underline{0}\}$

DFS o visita in profondità



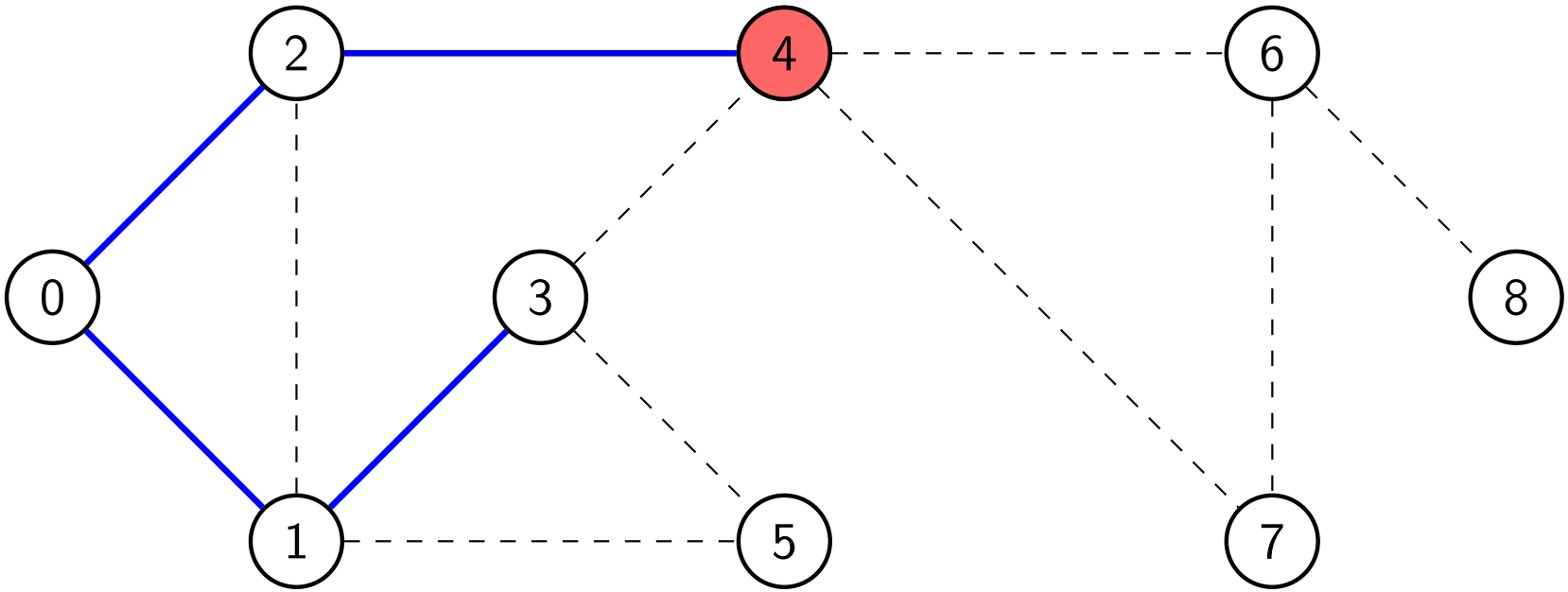
$S = \{3, 1, 0, 2\}$

DFS o visita in profondità



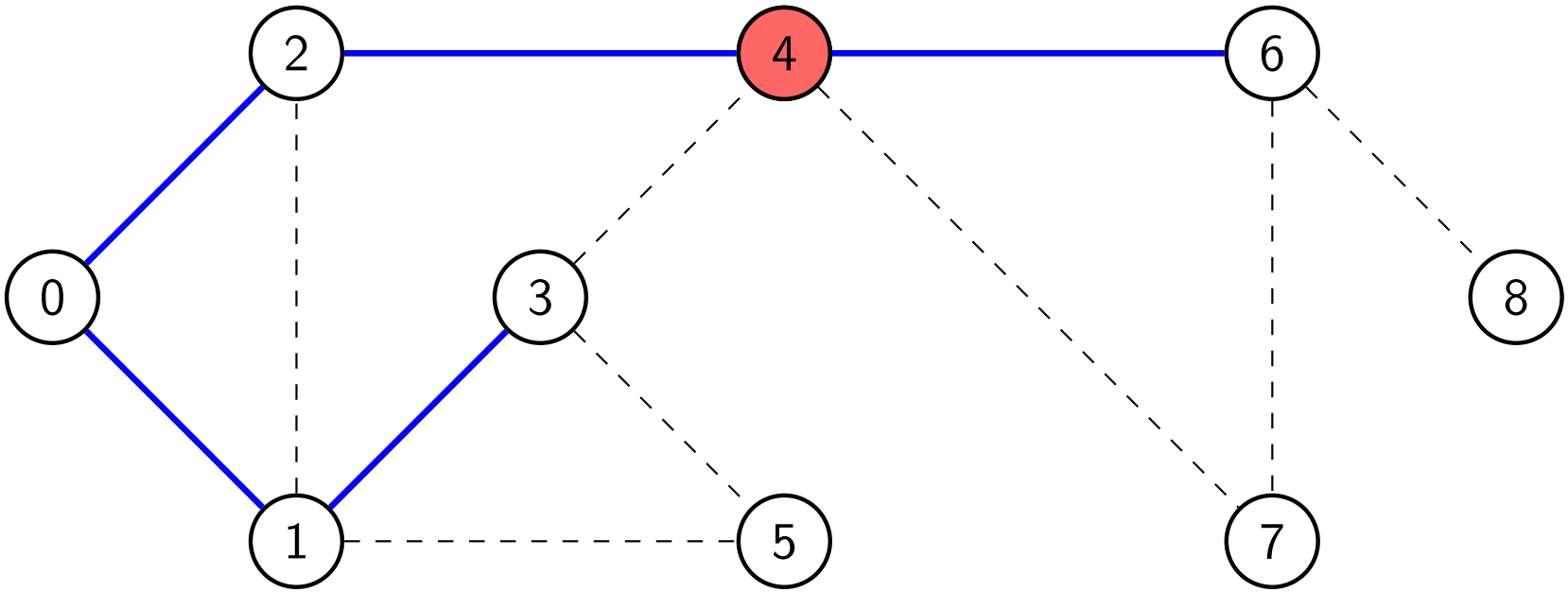
$S = \{3, 1, 0, \underline{2}\}$

DFS o visita in profondità



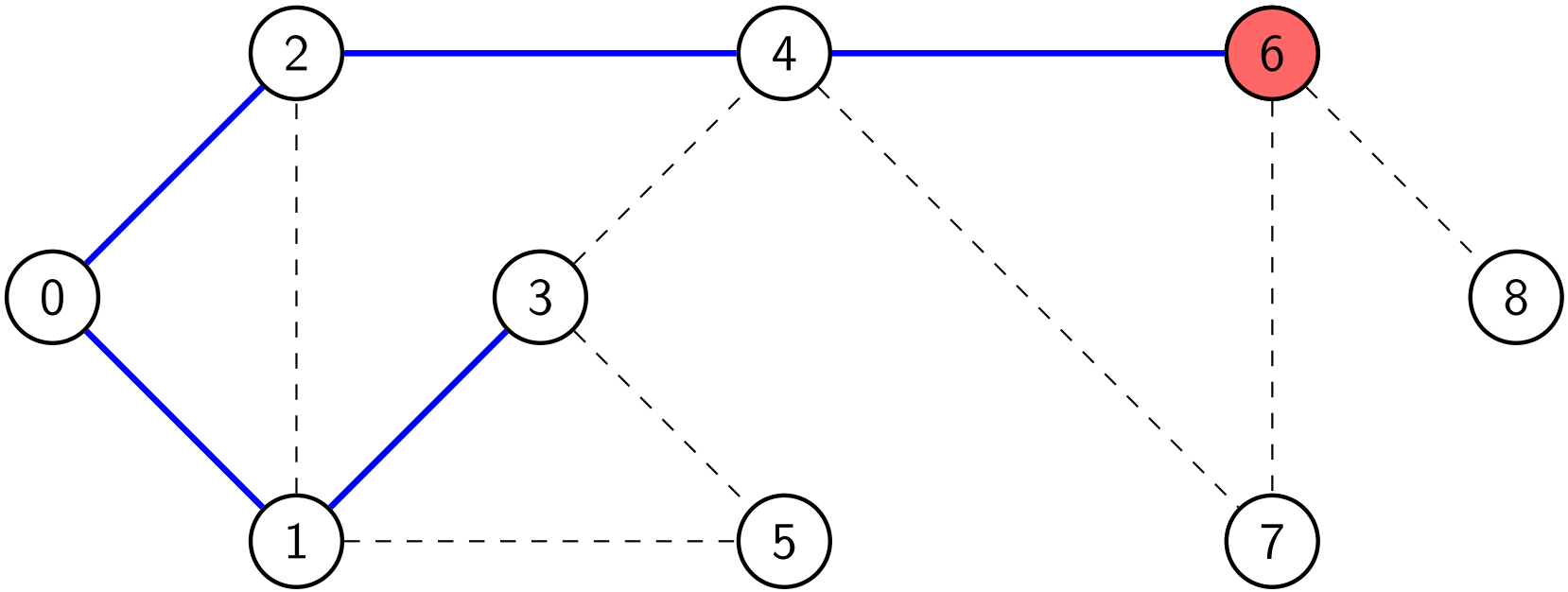
$S = \{3, 1, 0, 2, 4\}$

DFS o visita in profondità



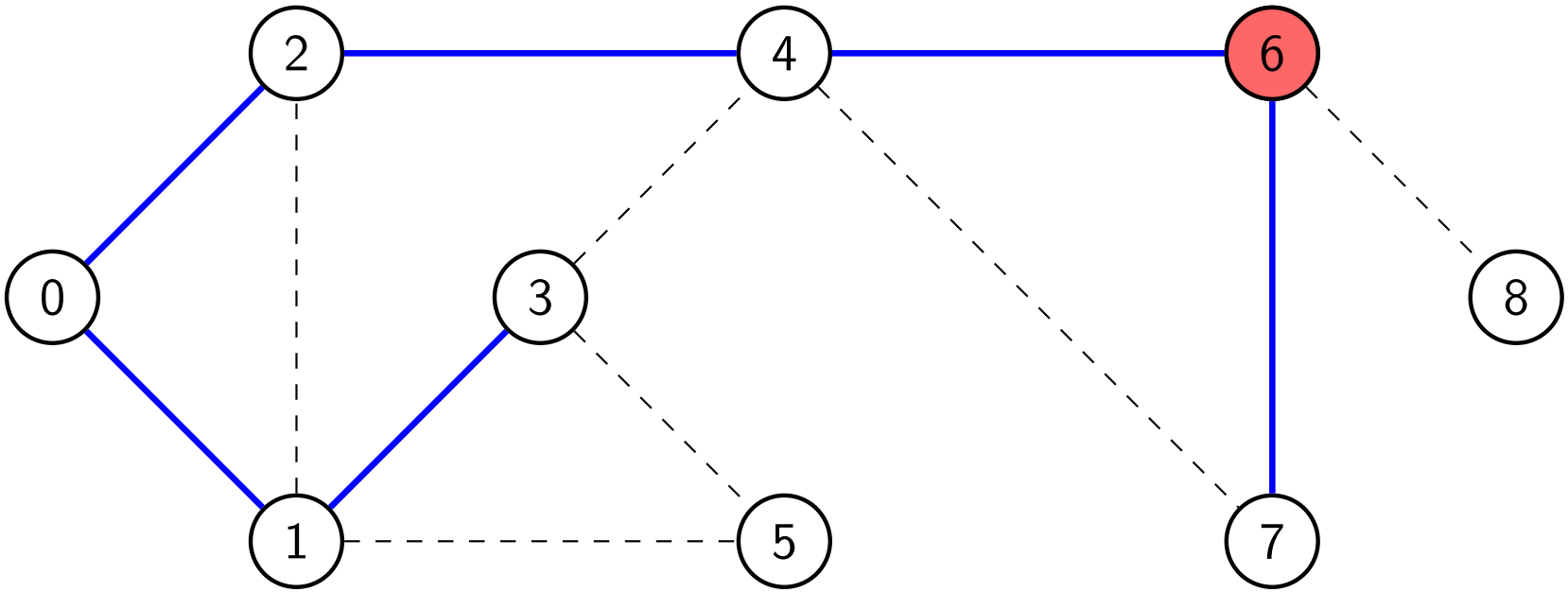
$S = \{3, 1, 0, 2, \underline{4}\}$

DFS o visita in profondità



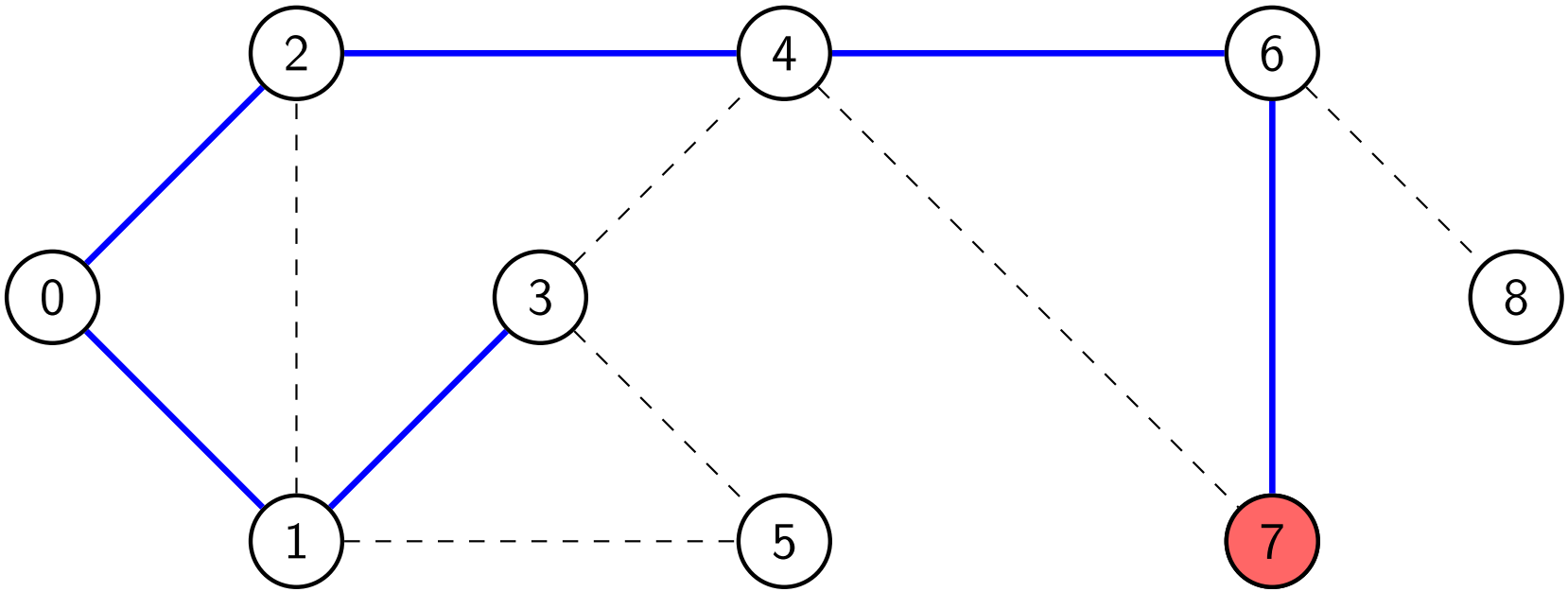
$S = \{3, 1, 0, 2, 4, 6\}$

DFS o visita in profondità



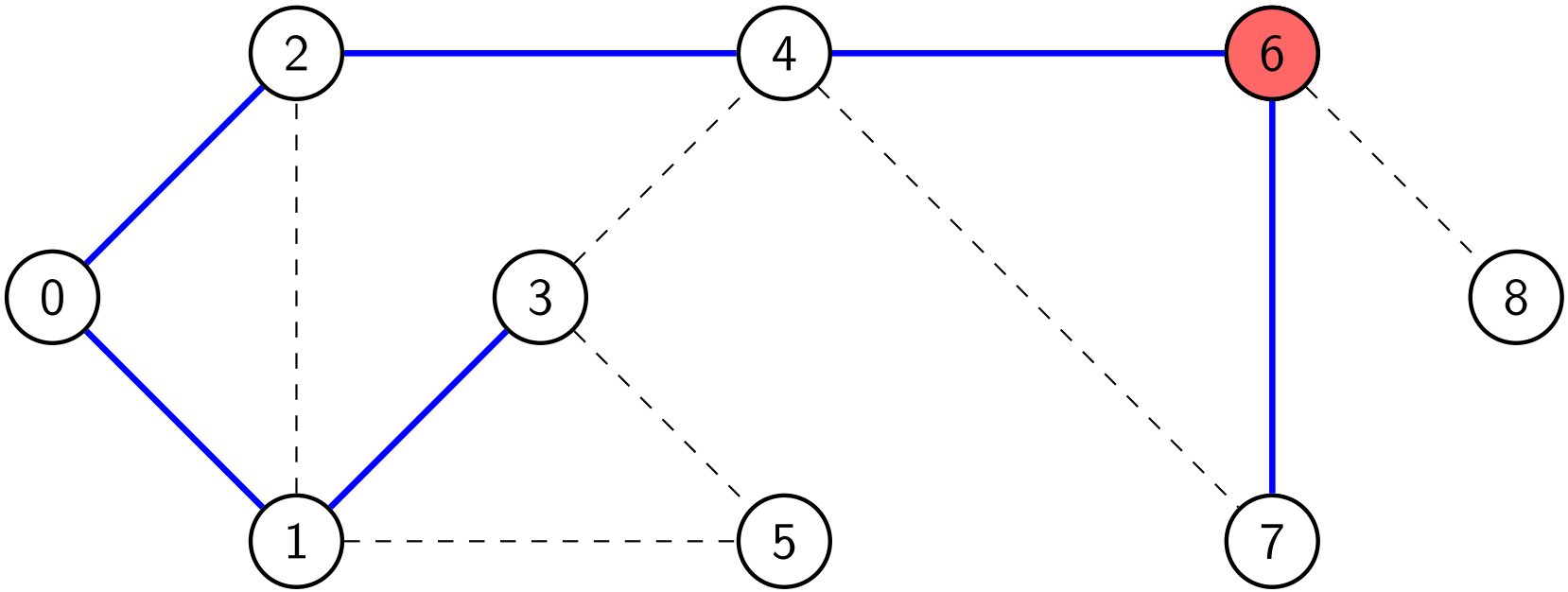
$S = \{3, 1, 0, 2, 4, \underline{6}\}$

DFS o visita in profondità



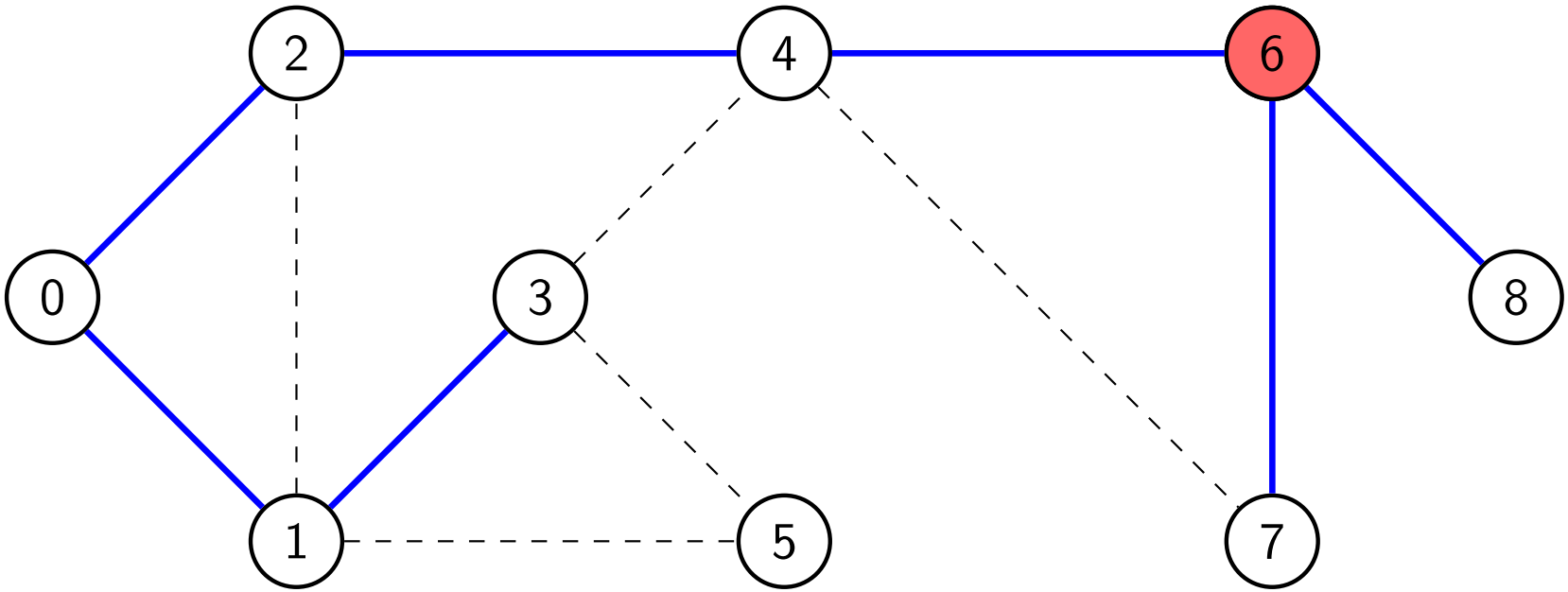
$S = \{3, 1, 0, 2, 4, 6, 7\}$

DFS o visita in profondità



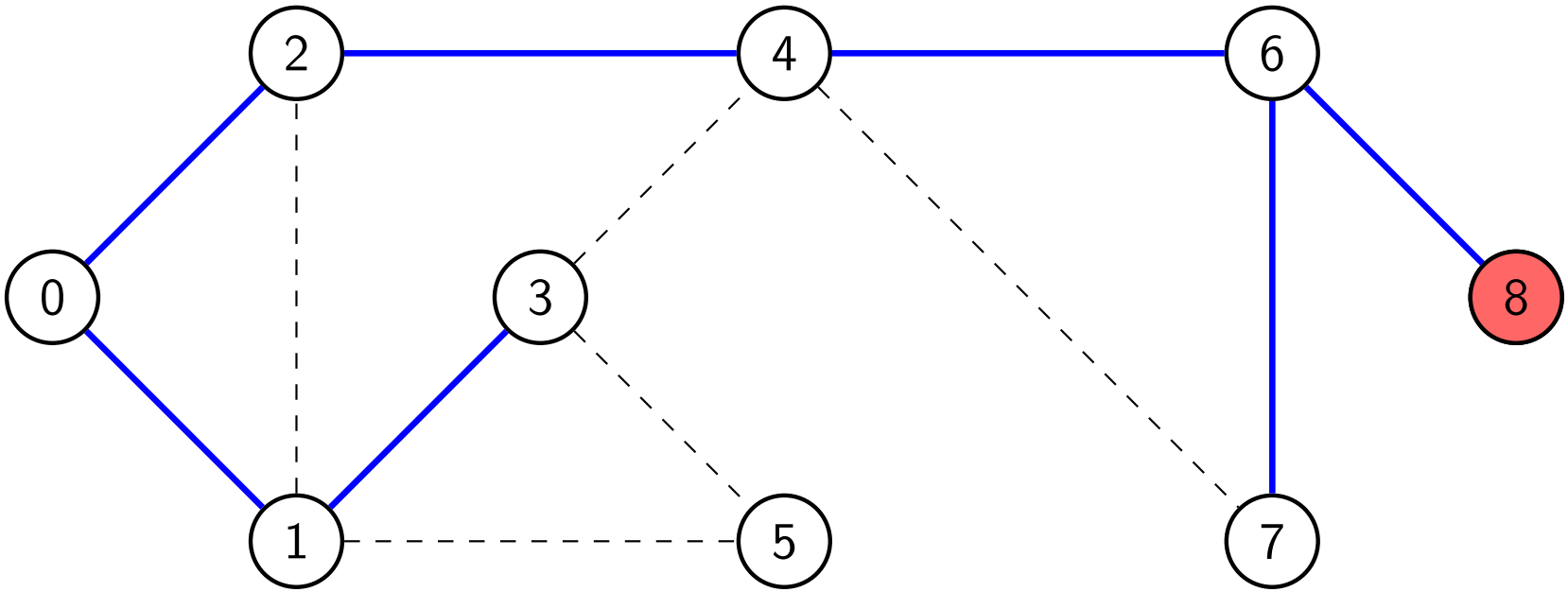
$S = \{3, 1, 0, 2, 4, 6\}$

DFS o visita in profondità



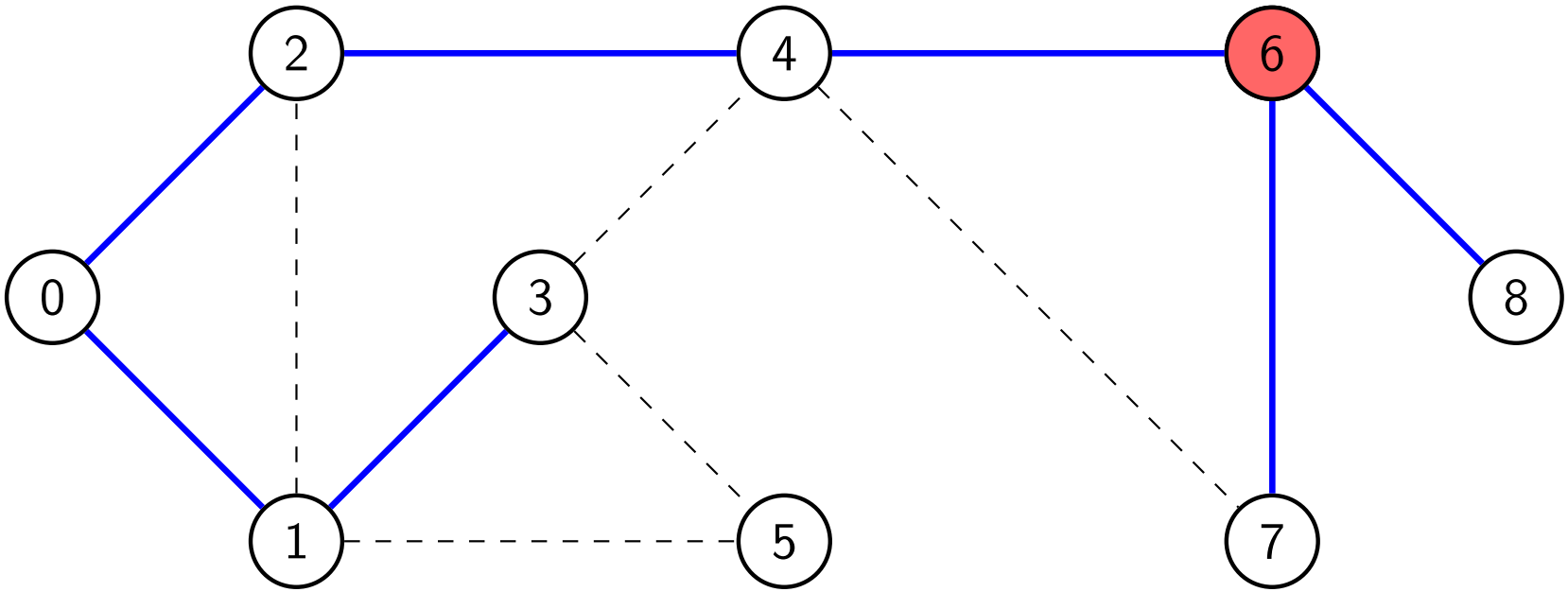
$S = \{3, 1, 0, 2, 4, \underline{6}\}$

DFS o visita in profondità



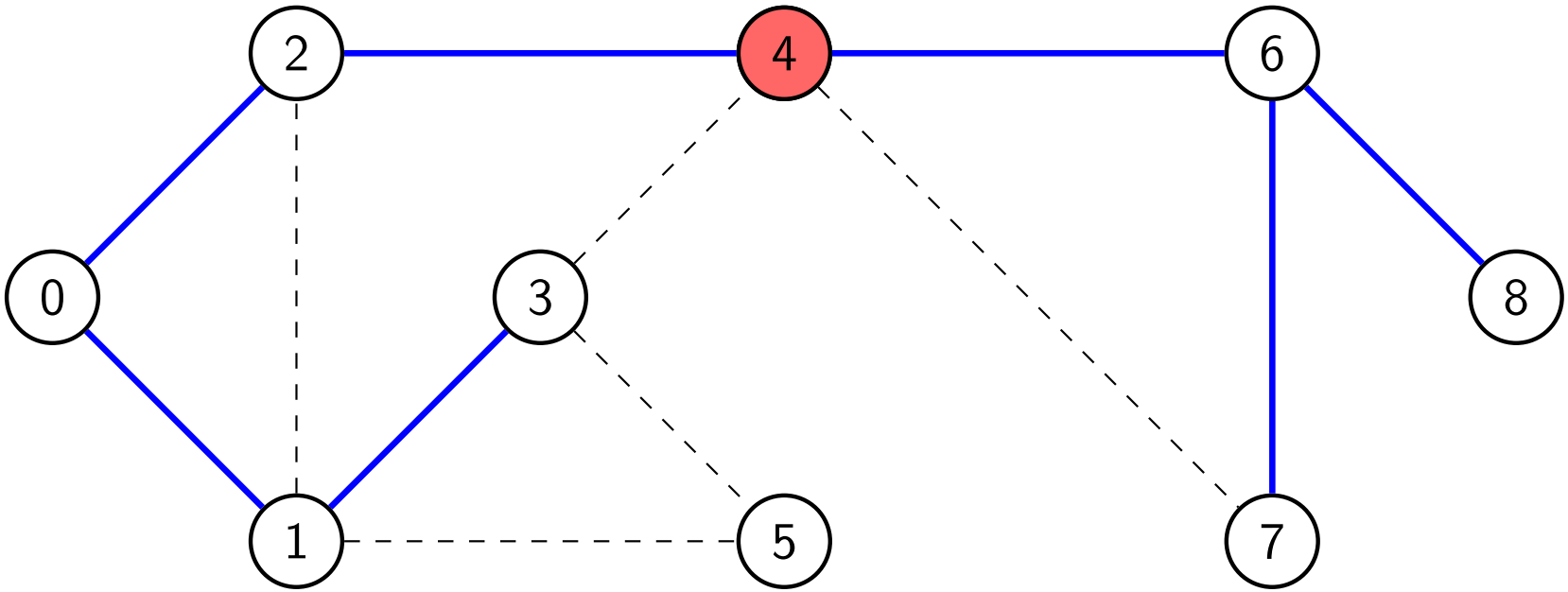
$S = \{3, 1, 0, 2, 4, 6, 8\}$

DFS o visita in profondità



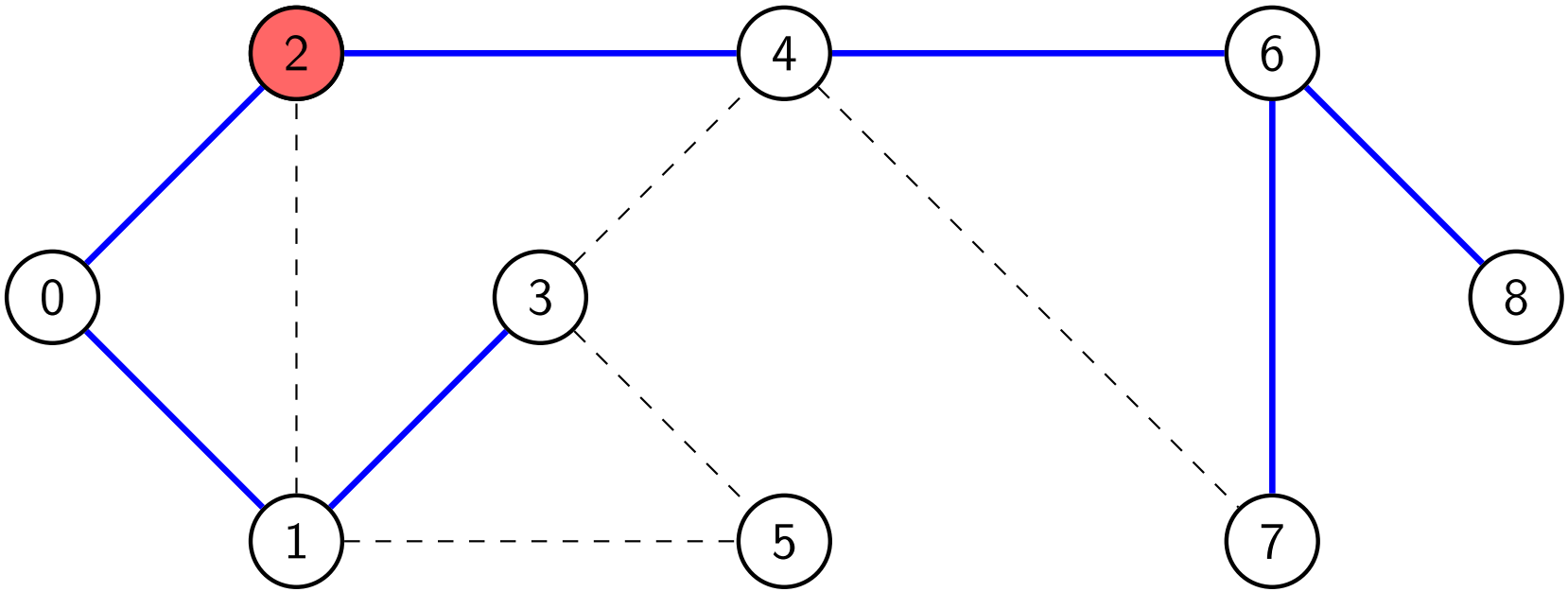
$S = \{3, 1, 0, 2, 4, 6\}$

DFS o visita in profondità



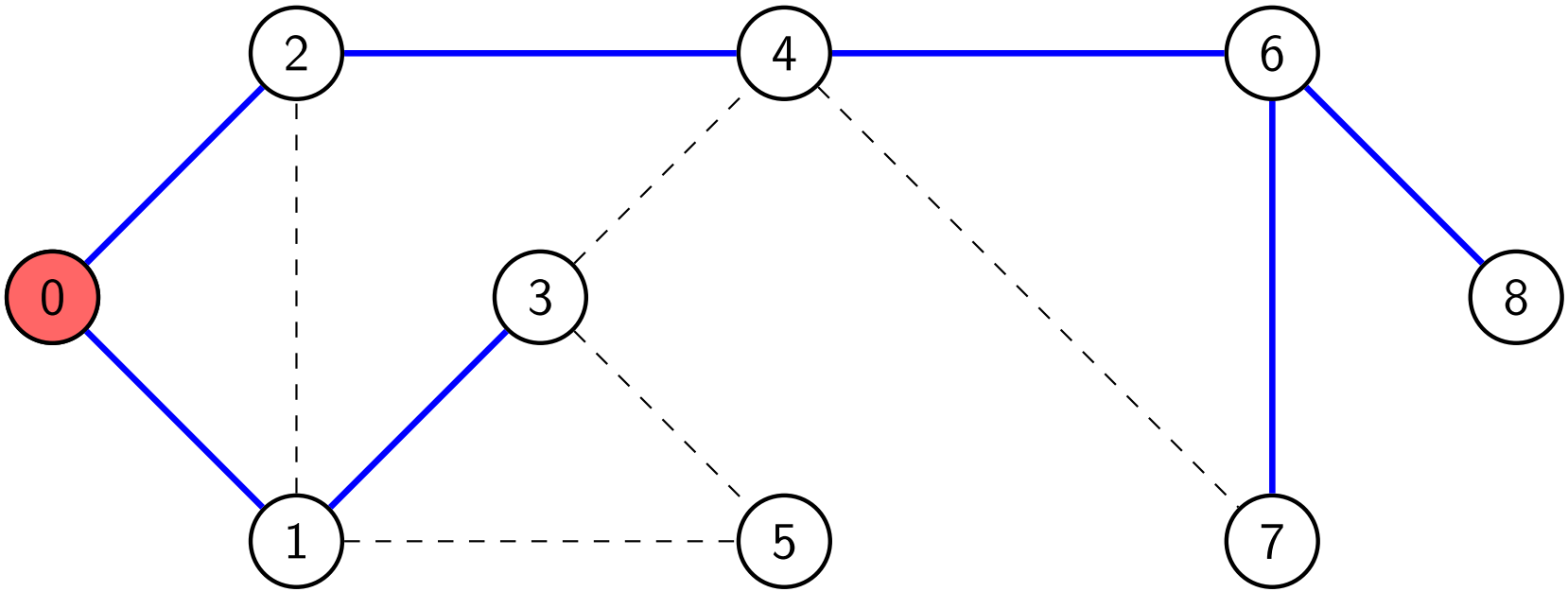
$S = \{3, 1, 0, 2, 4\}$

DFS o visita in profondità



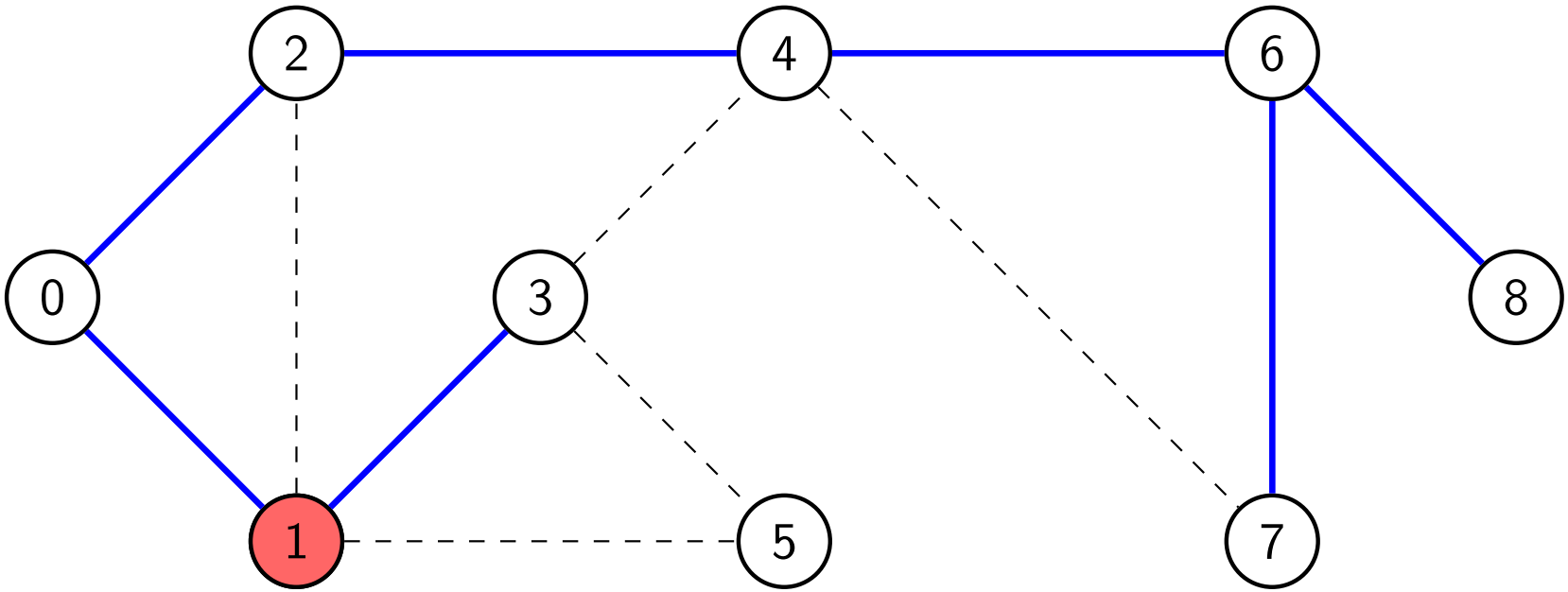
$S = \{3, 1, 0, 2\}$

DFS o visita in profondità



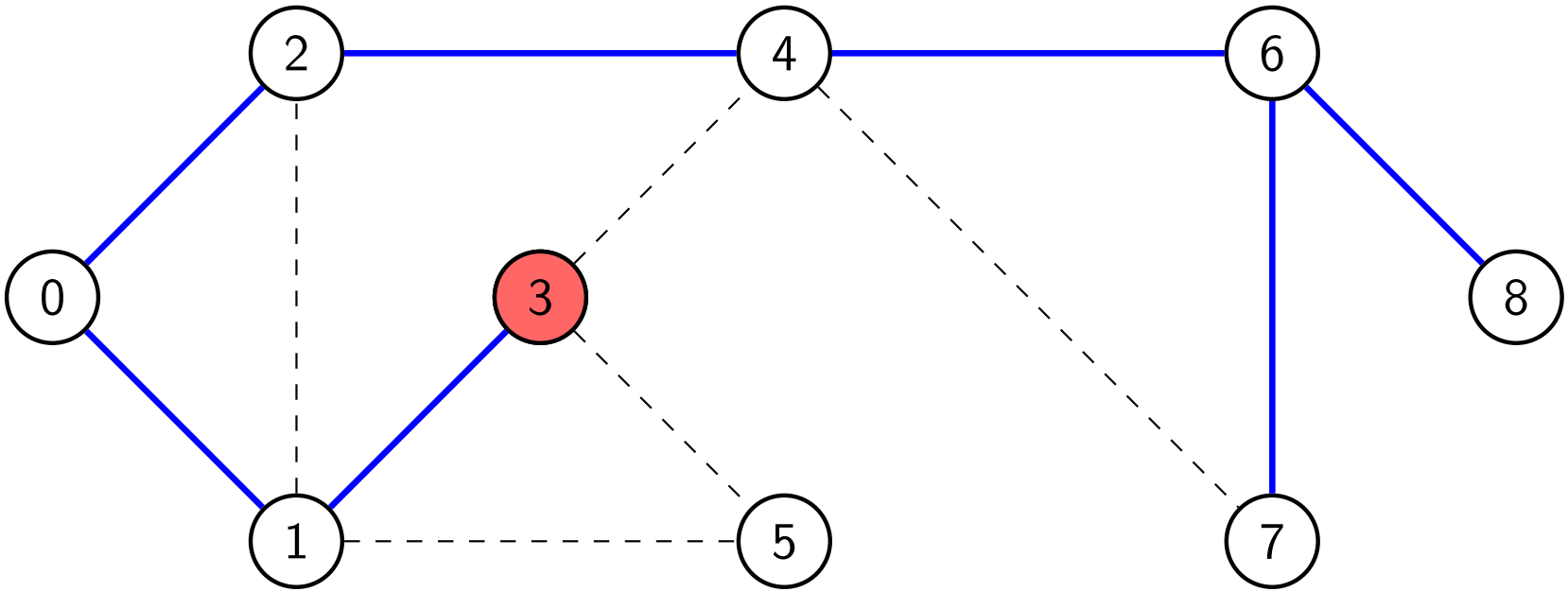
$S = \{3, 1, 0\}$

DFS o visita in profondità



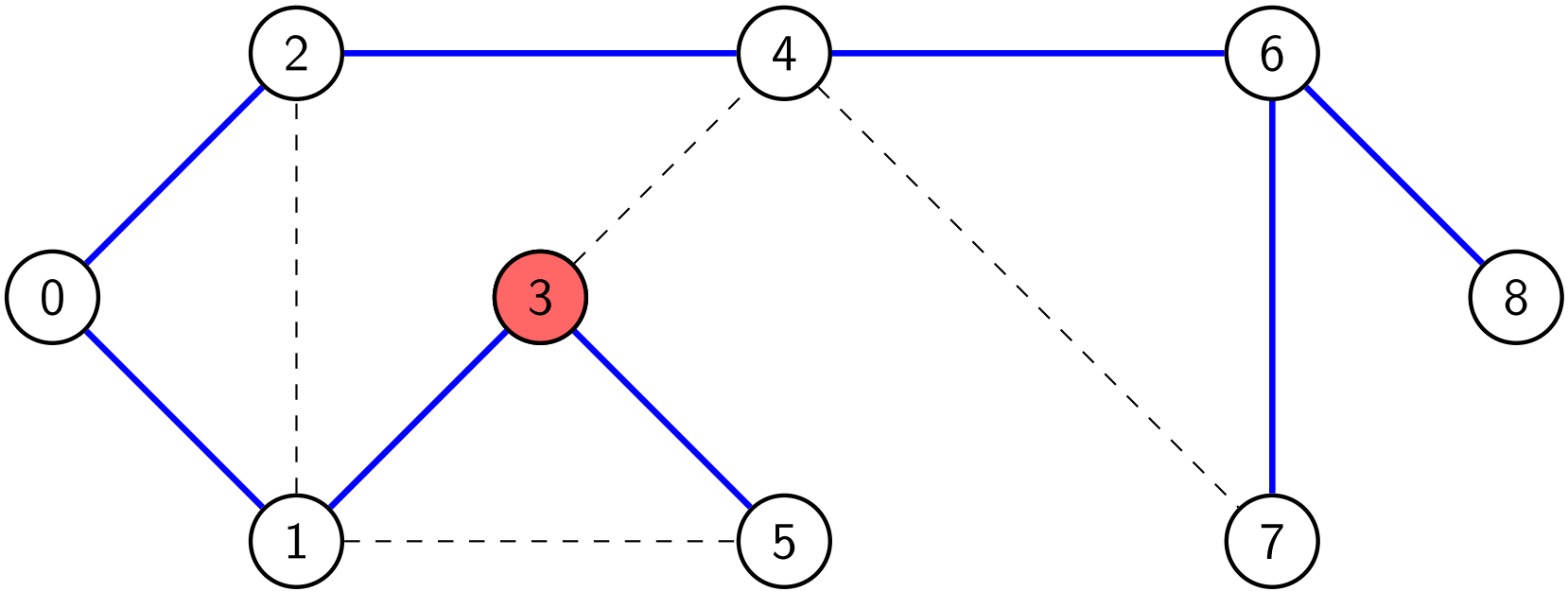
$S = \{3, 1\}$

DFS o visita in profondità



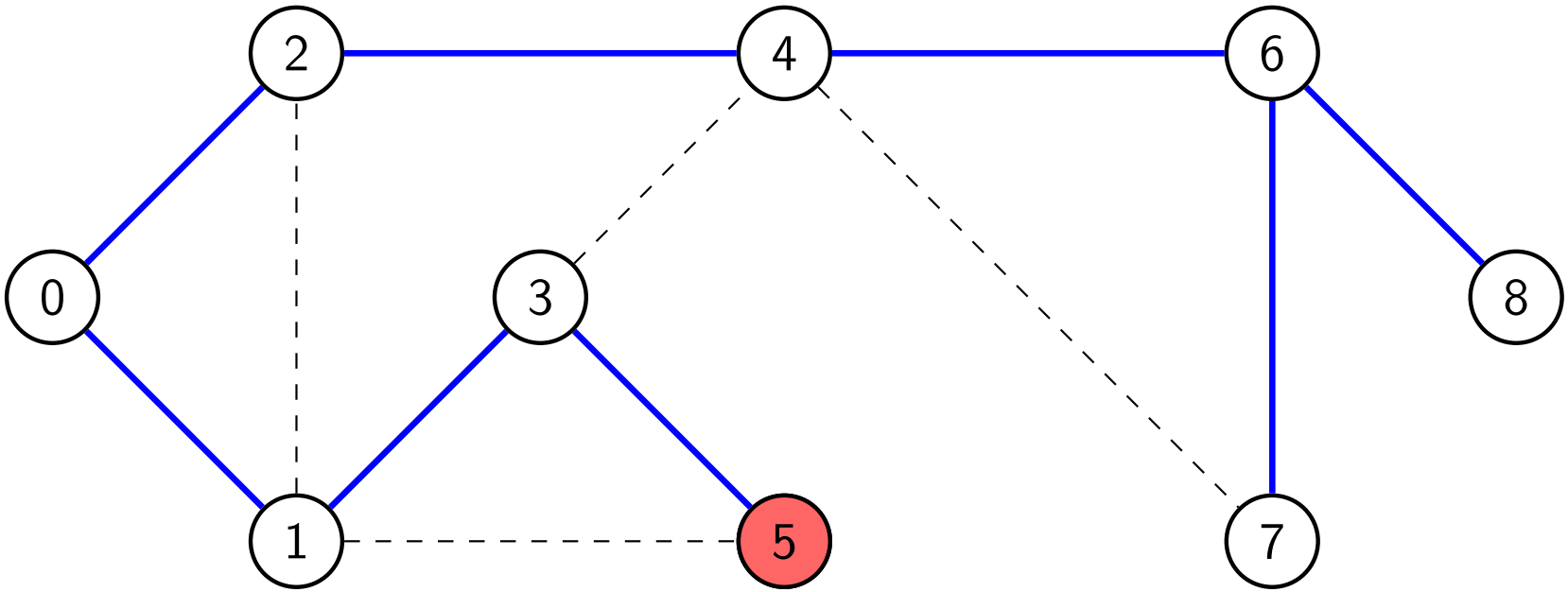
$S = \{3\}$

DFS o visita in profondità



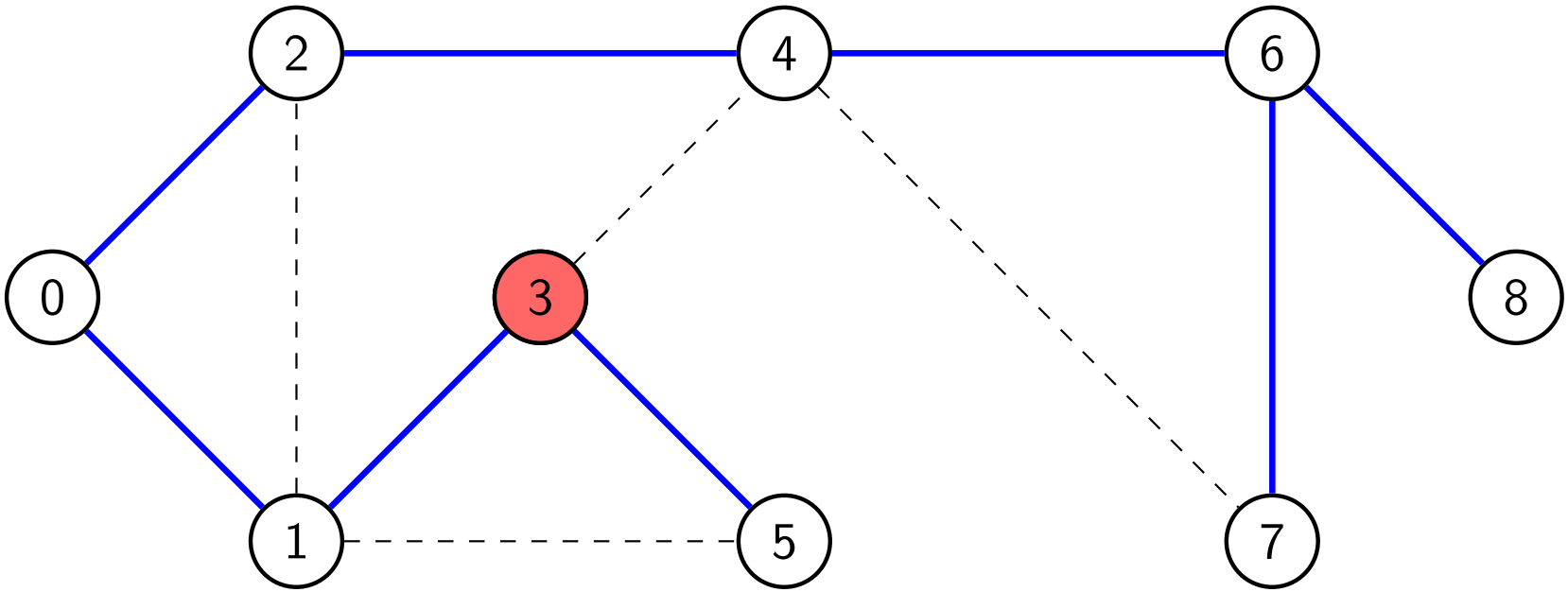
$S = \{\underline{3}\}$

DFS o visita in profondità



$S = \{3, 5\}$

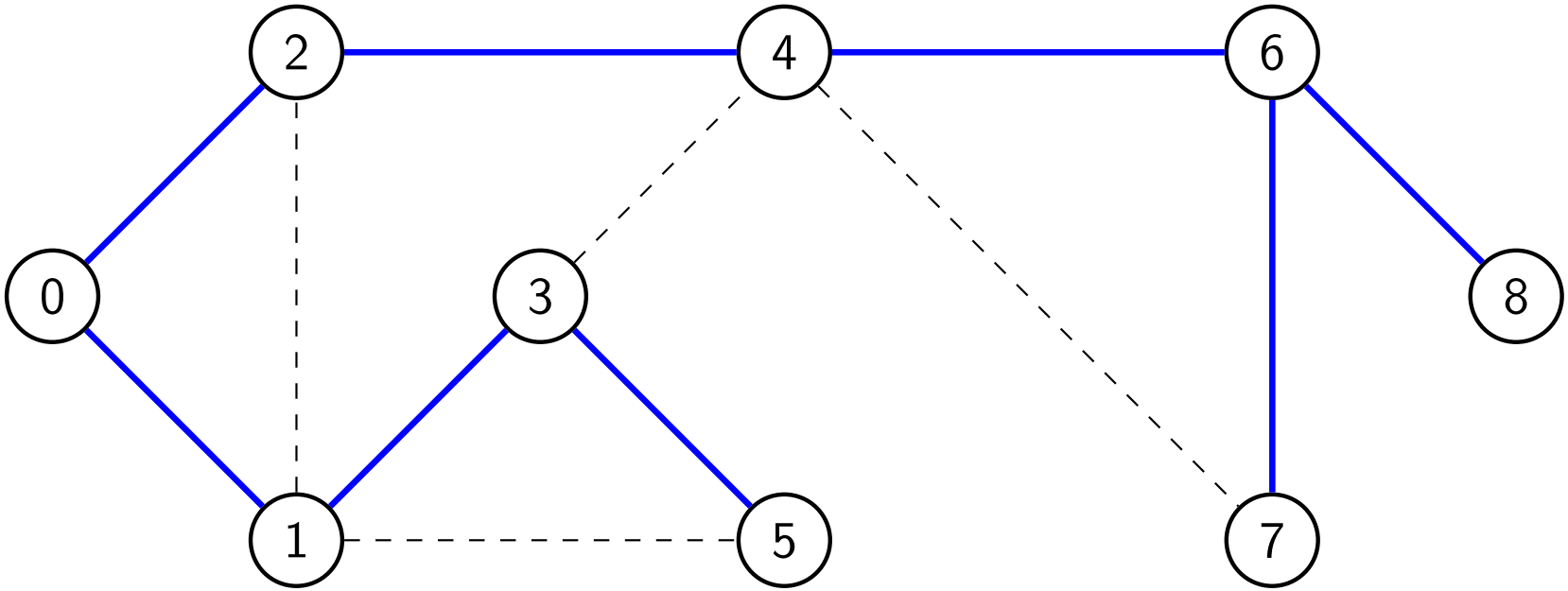
DFS o visita in profondità



$S = \{3\}$

DFS o visita in profondità

La pila è vuota, l'algoritmo termina.



$S = \{\}$

Esercizio

Trovare le componenti connesse di un grafo.

Due nodi sono nella stessa componente connessa se e solo se esiste un cammino che li collega.

Esercizio

Dire se un grafo dato è bipartito ed eventualmente mostrarne la bipartizione.

Un grafo è bipartito se è possibile dividere i nodi in due gruppi e tutti gli archi collegano nodi di un gruppo con nodi dell'altro.

Problema

Siamo interessati a trovare il percorso più breve fra due vertici in un grafo con archi pesati.

Problema

Siamo interessati a trovare il percorso più breve fra due vertici in un grafo con archi pesati.

In realtà tutti gli algoritmi che studieremo ci daranno un risultato leggermente più generale, cioè troveremo i percorsi più brevi da un dato vertice a *tutti* gli altri vertici del grafo.

Problema

Siamo interessati a trovare il percorso più breve fra due vertici in un grafo con archi pesati.

In realtà tutti gli algoritmi che studieremo ci daranno un risultato leggermente più generale, cioè troveremo i percorsi più brevi da un dato vertice a *tutti* gli altri vertici del grafo.

Osservazione

Possiamo assumere che il grafo sia connesso: dal nodo in questione è possibile raggiungere ogni altro nodo.

Struttura dei cammini minimi

Fissiamo un vertice v e per ogni altro vertice w segniamo sul grafo un cammino minimo che collega v a w.

Allora si possono scegliere tali cammini minimi in modo che il sottografo risultante sia un albero, detto albero dei cammini minimi.

Algoritmo di Dijkstra

L'algoritmo di Dijkstra risolve il problema dei cammini minimi da un vertice nel caso in cui il grafo abbia pesi degli archi non negativi.

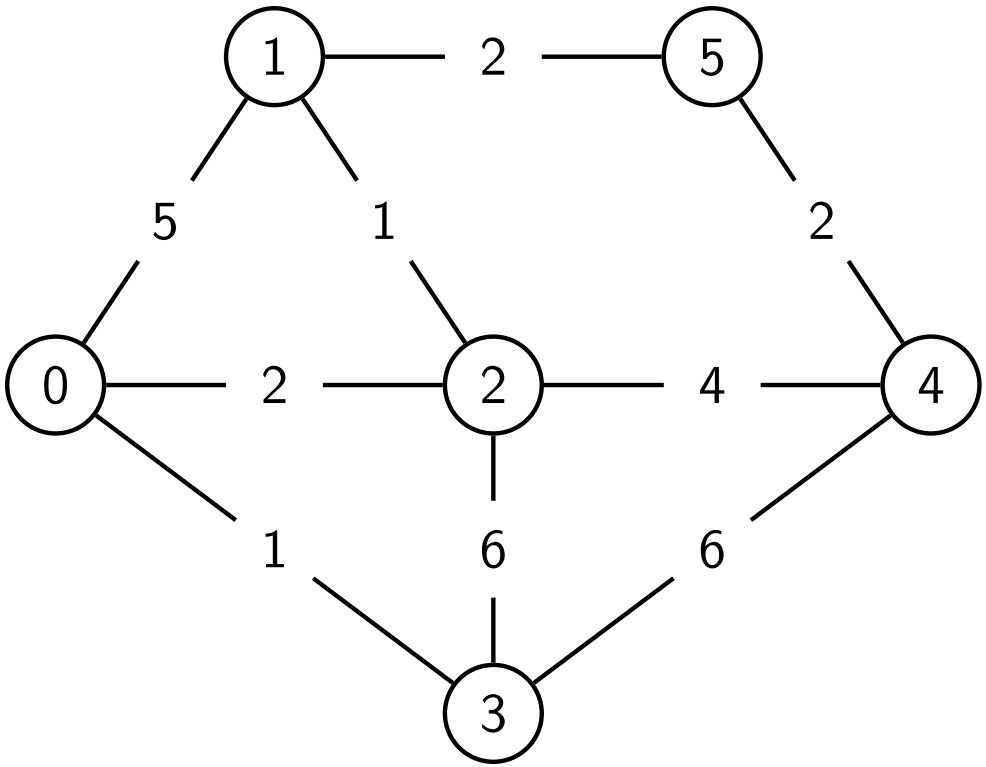
Osservazione

Se il grafo ha archi con pesi negativi il problema potrebbe diventare non ben definito: possono esistere cicli di lunghezza negativa!

Algoritmo di Dijkstra

Algoritmo

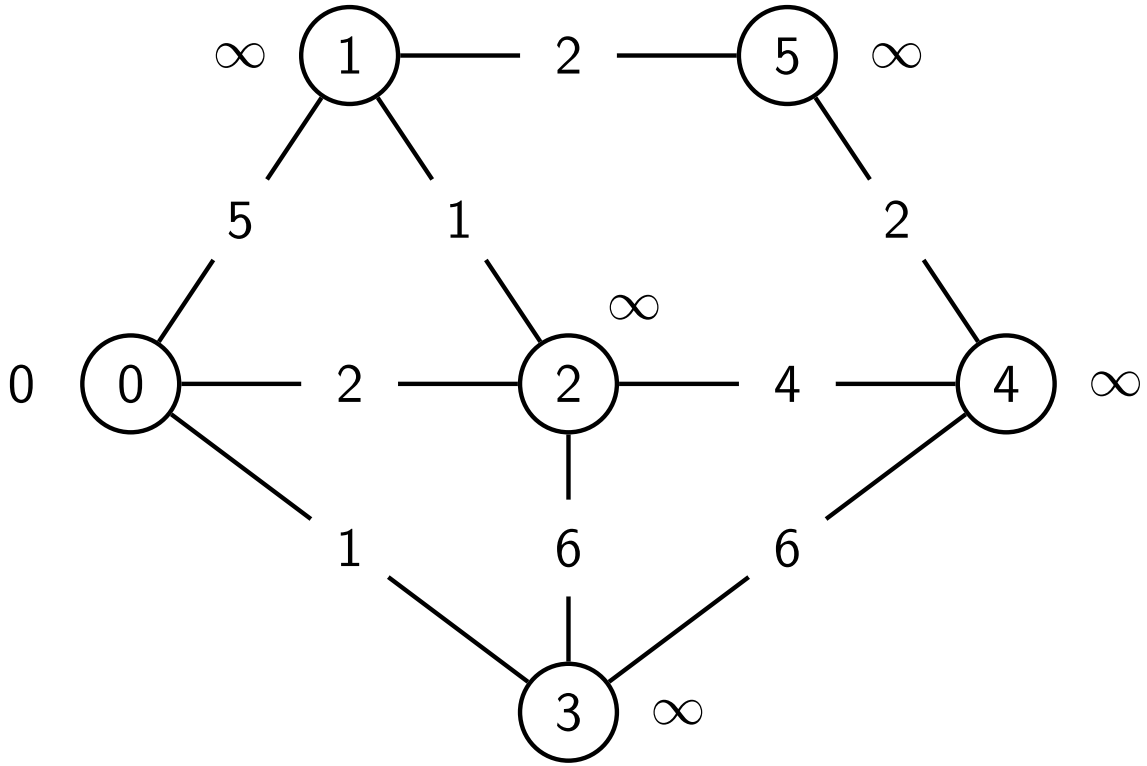
Vogliamo calcolare la distanza minima dal nodo 0 verso tutti gli altri nodi.



Algoritmo di Dijkstra

Algoritmo

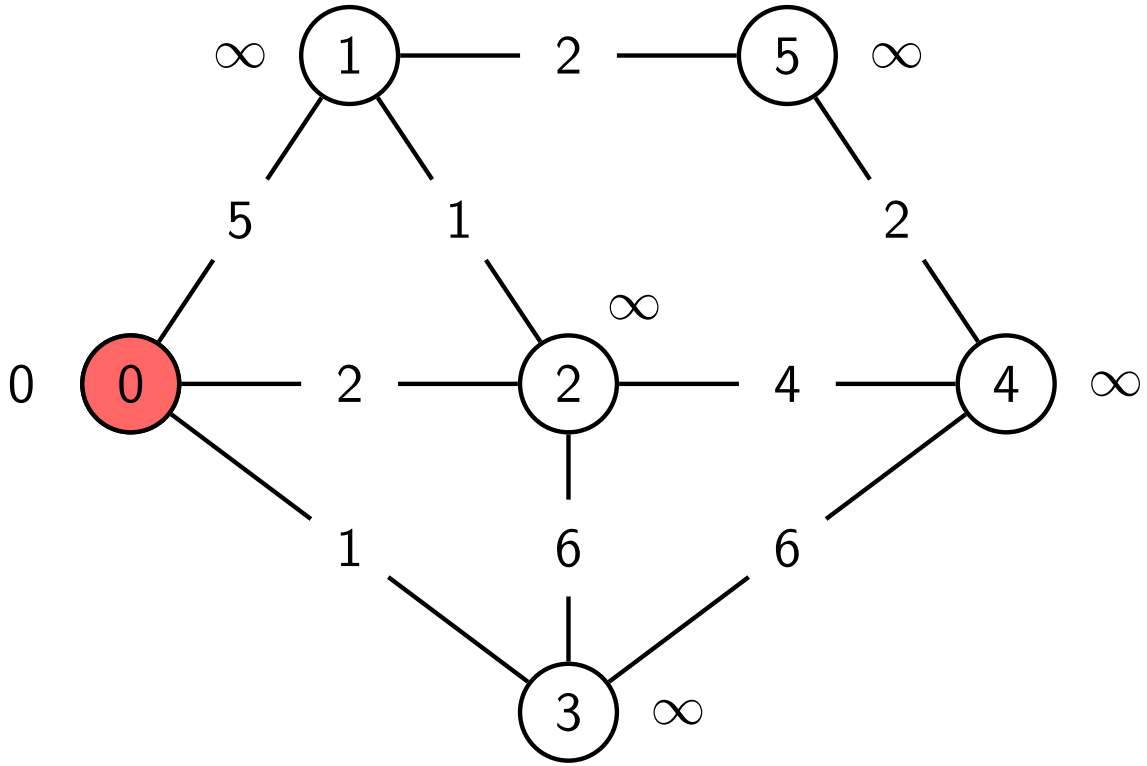
Inizializzazione: la distanza di tutti i nodi è ∞, tranne per la sorgente.



Algoritmo di Dijkstra

Algoritmo

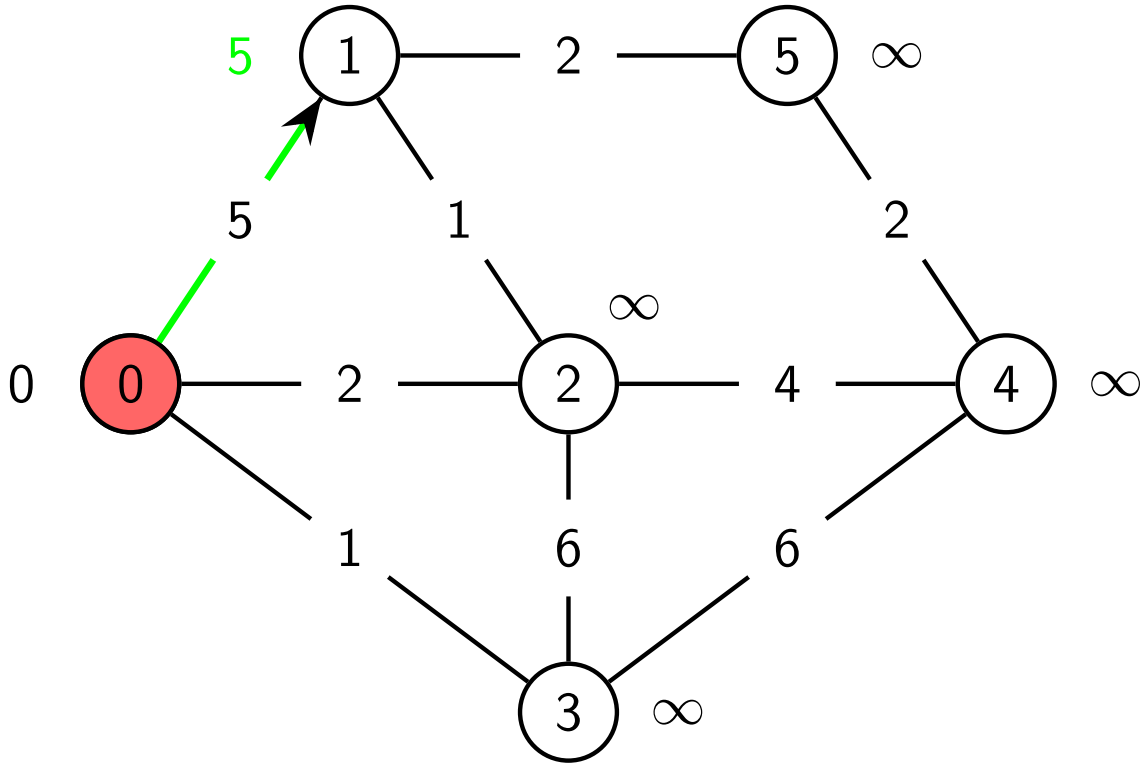
Selezioniamo il nodo non espanso con distanza minima.



Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

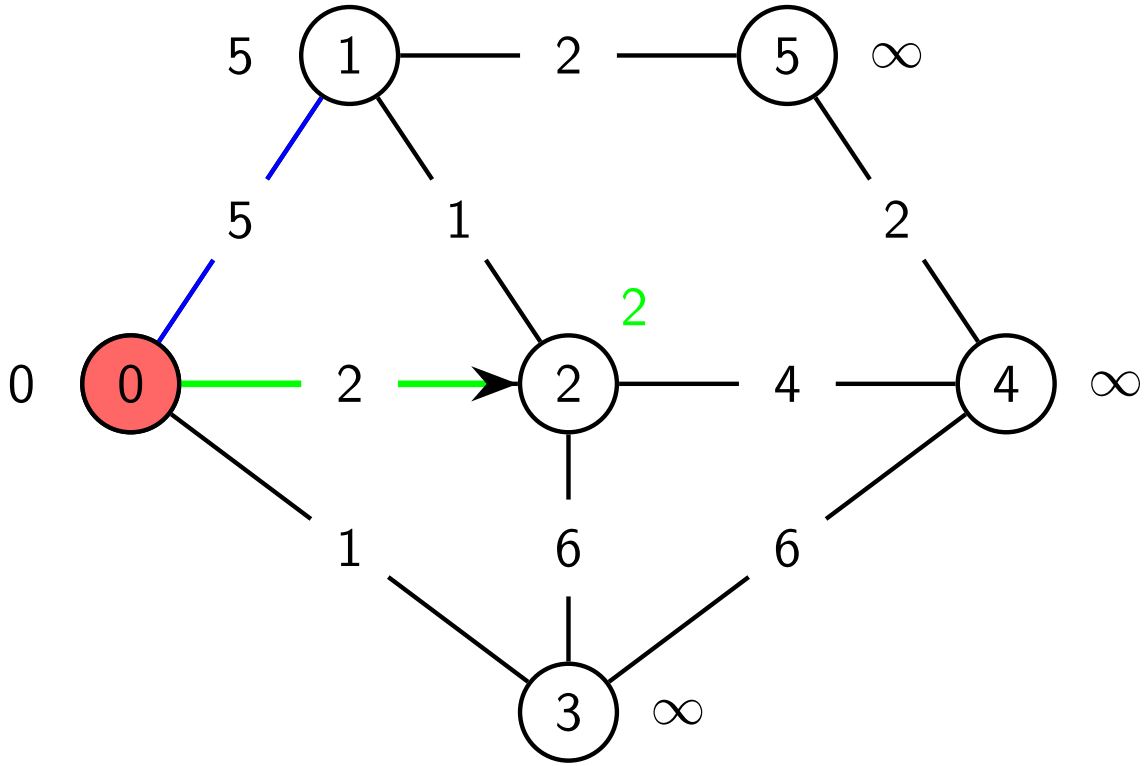


$0 + 5 < \infty$

Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

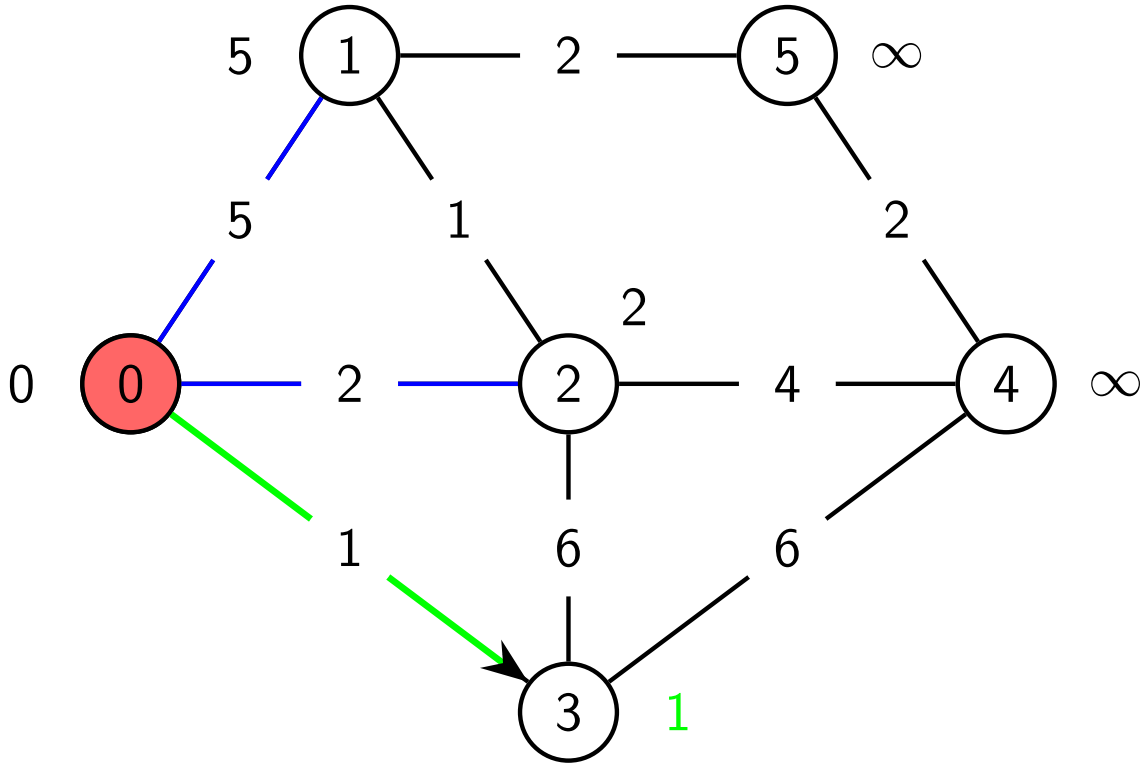


$0 + 2 < \infty$

Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

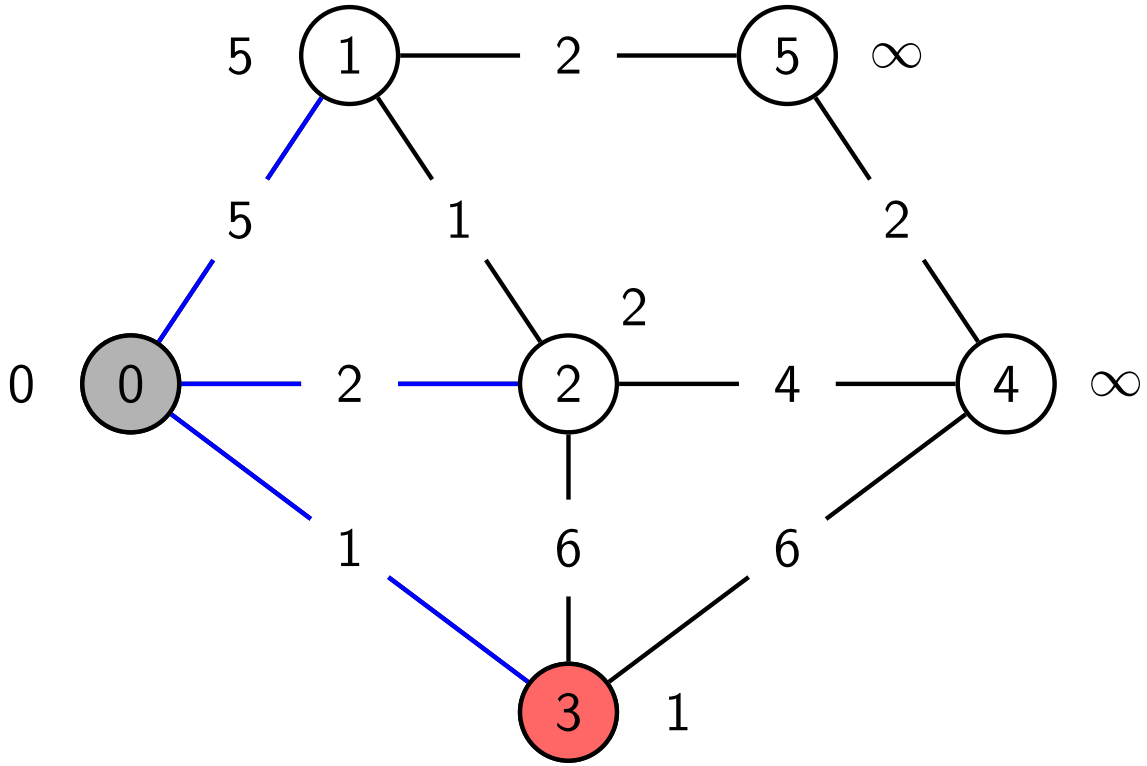


$0 + 1 < \infty$

Algoritmo di Dijkstra

Algoritmo

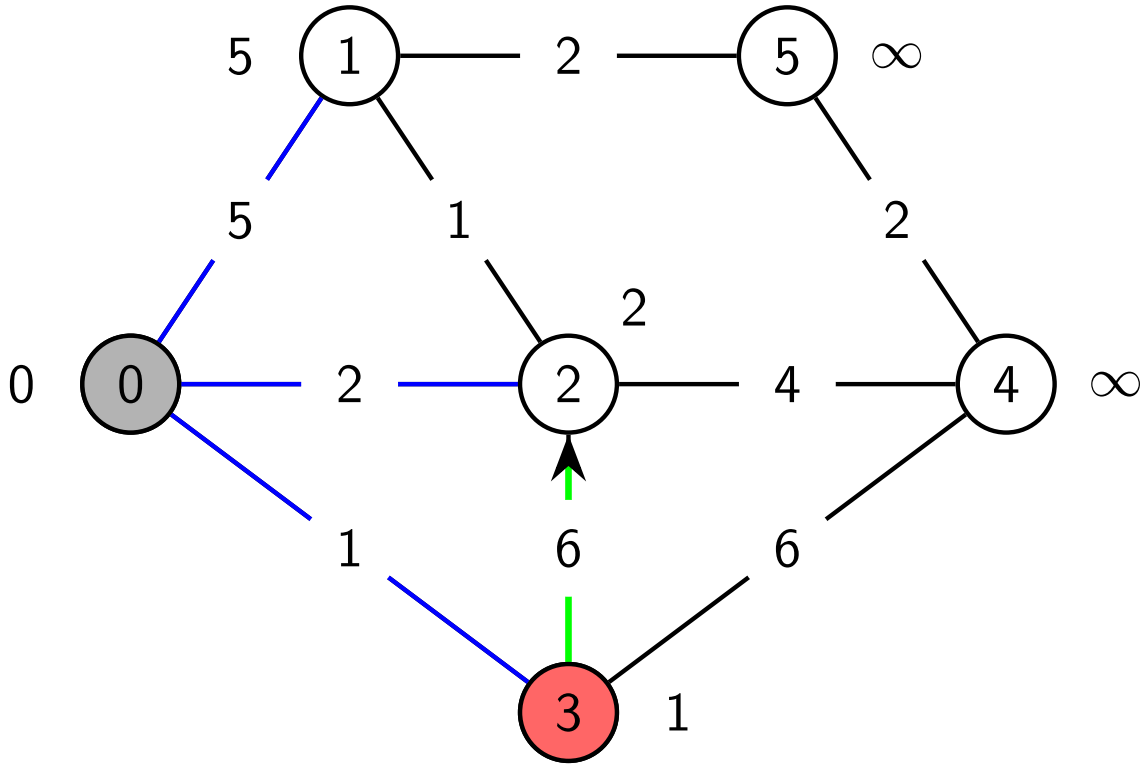
Selezioniamo il nodo non espanso con distanza minima.



Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

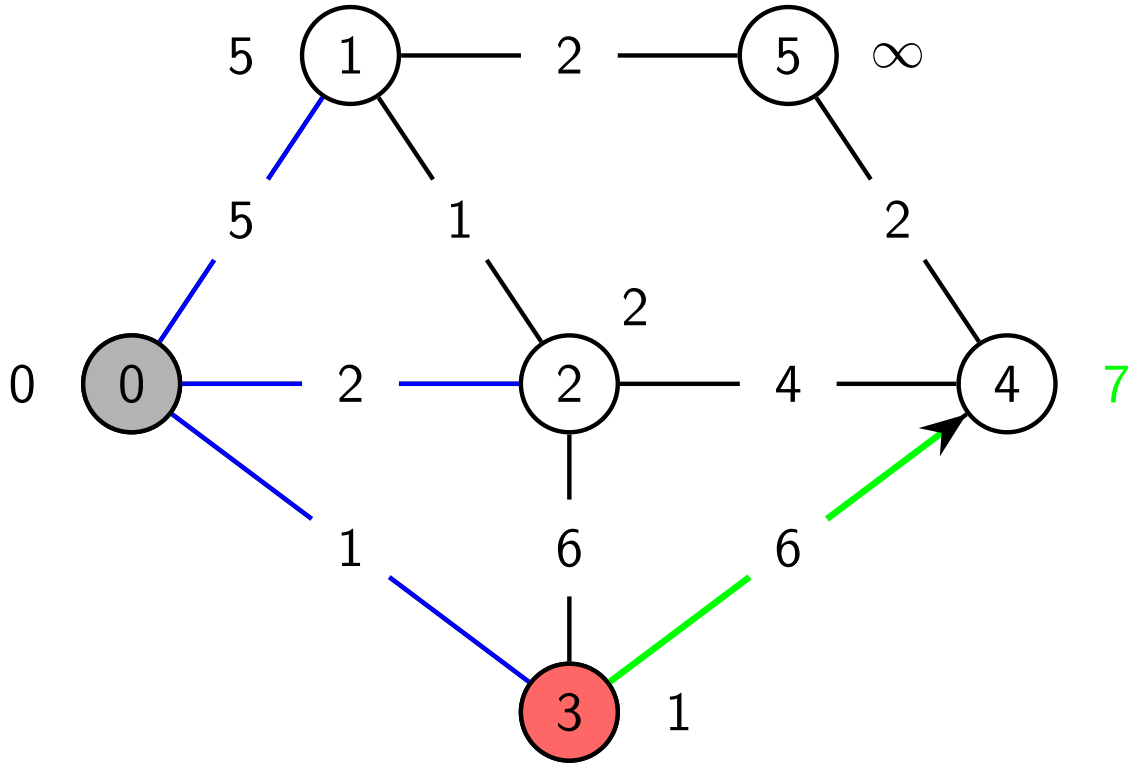


$1 + 6 \not< 2$

Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

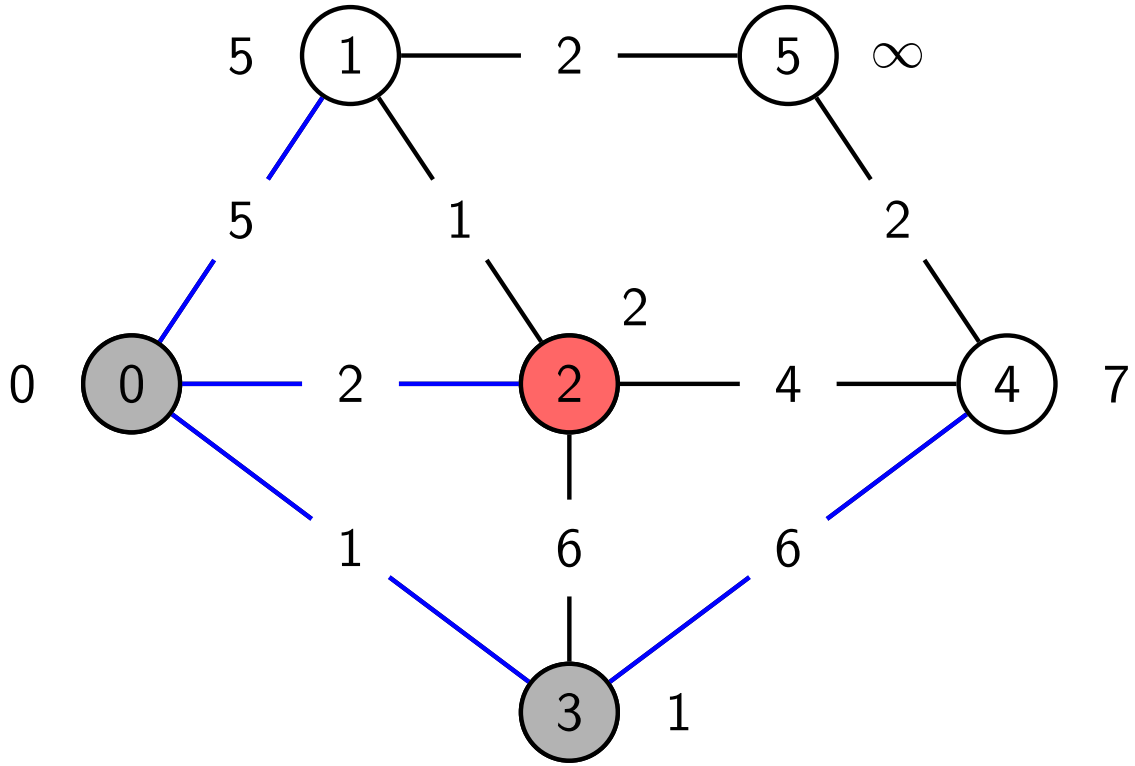


$1 + 6 < \infty$

Algoritmo di Dijkstra

Algoritmo

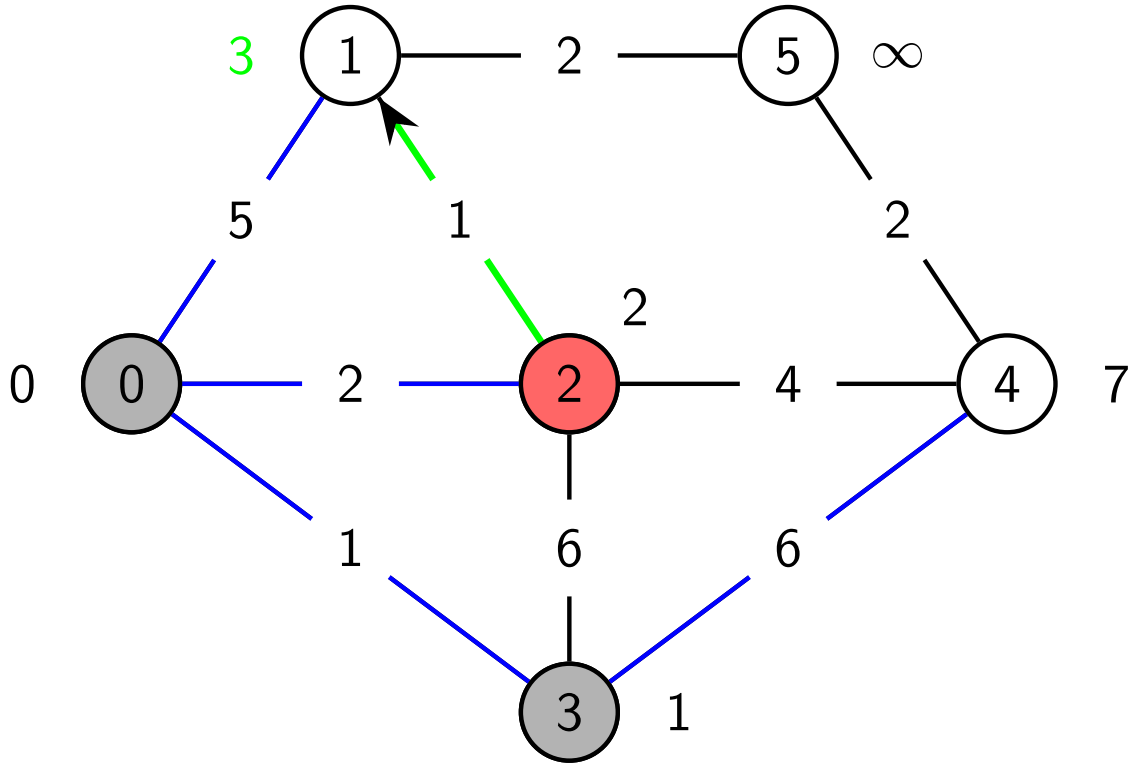
Selezioniamo il nodo non espanso con distanza minima.



Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

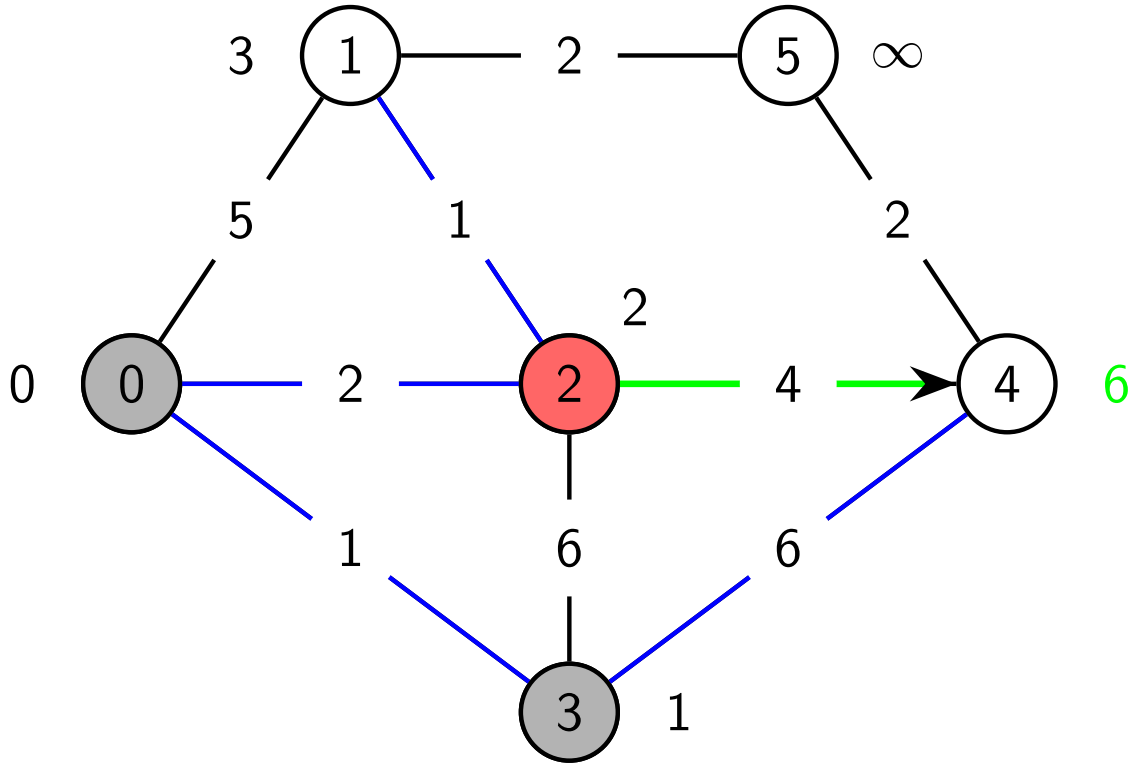


$2 + 1 < 5$

Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

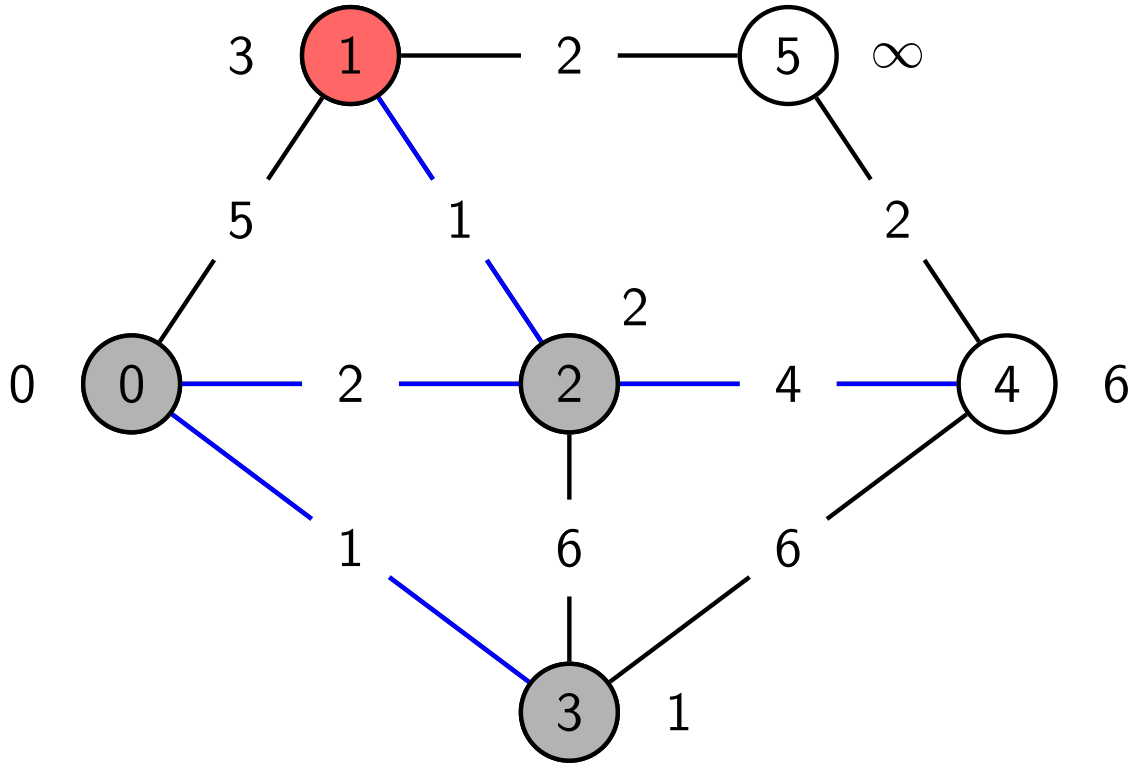


$2 + 4 < 6$

Algoritmo di Dijkstra

Algoritmo

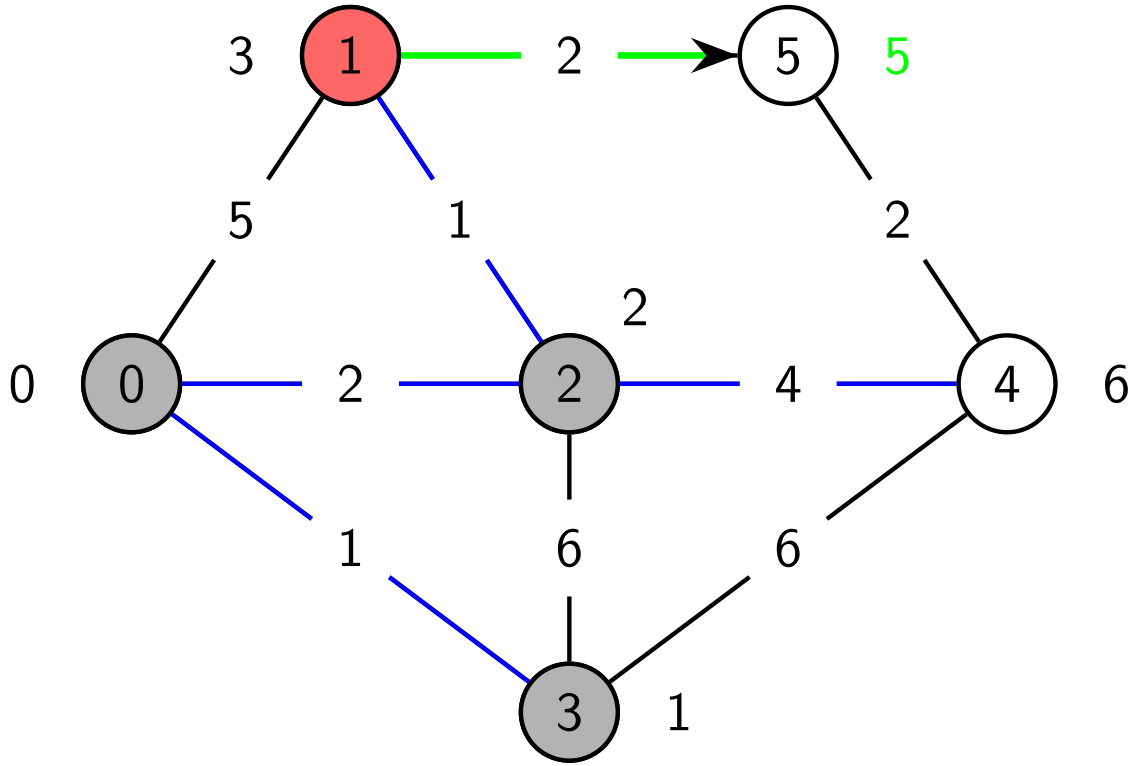
Selezioniamo il nodo non espanso con distanza minima.



Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

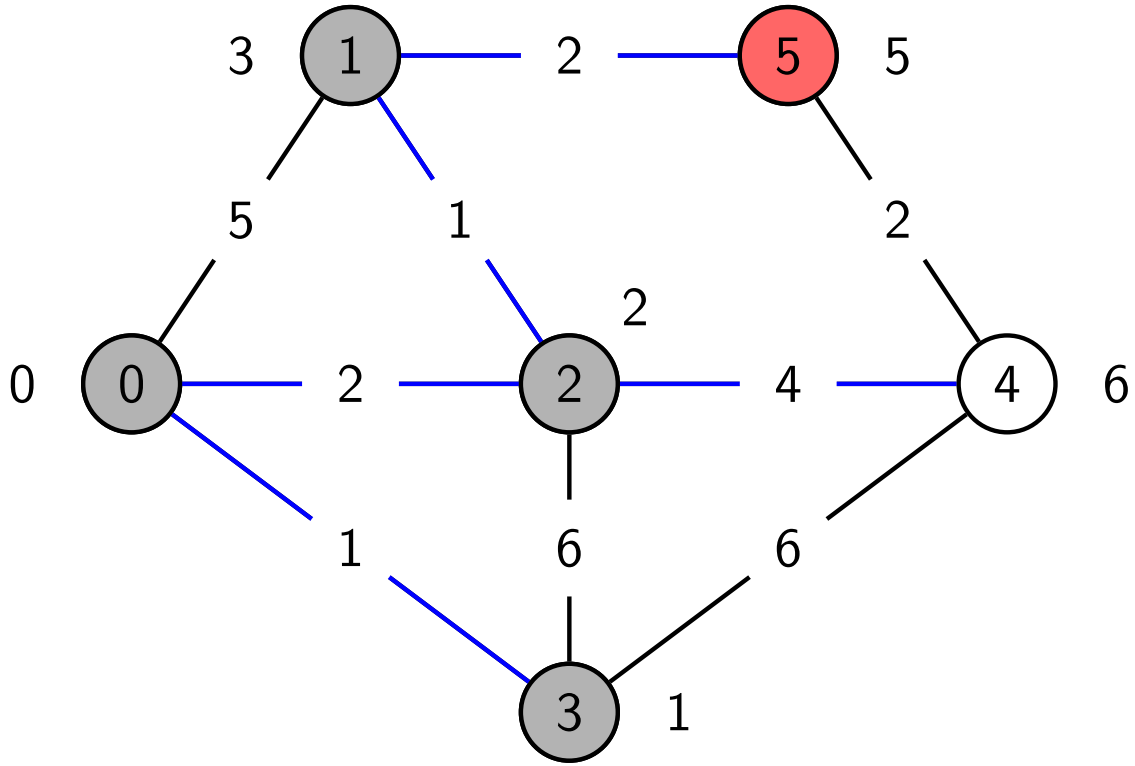


$3 + 2 < \infty$

Algoritmo di Dijkstra

Algoritmo

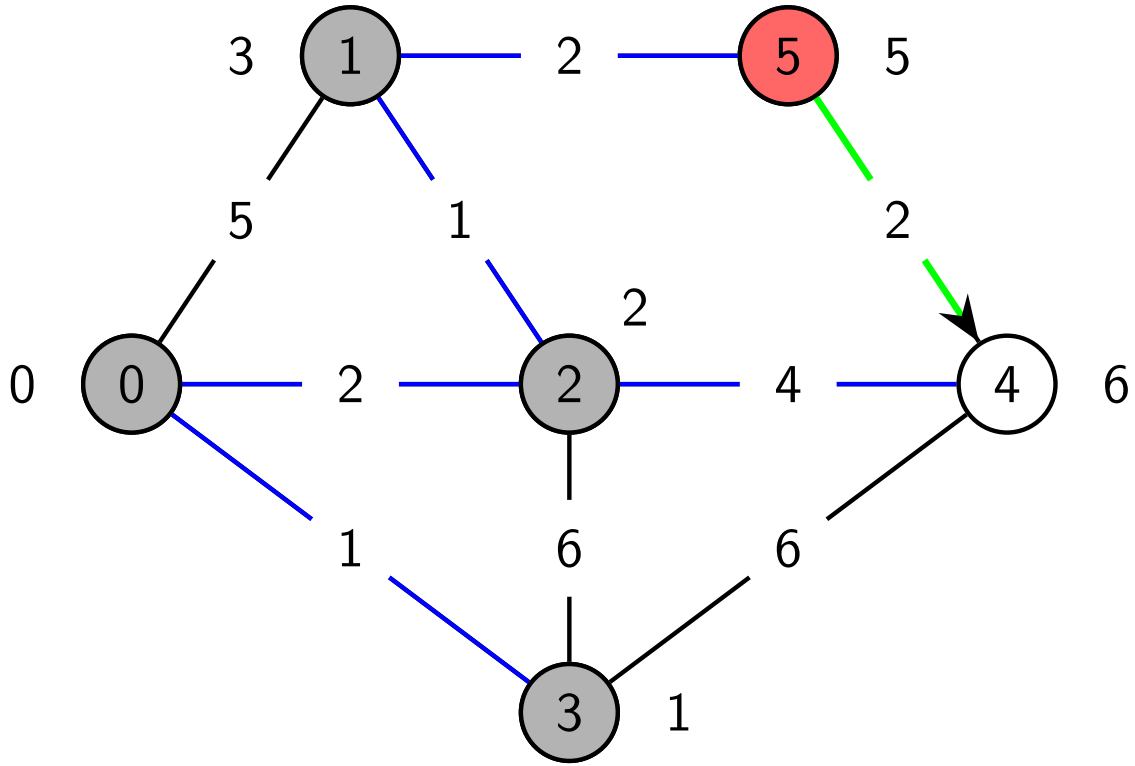
Selezioniamo il nodo non espanso con distanza minima.



Algoritmo di Dijkstra

Algoritmo

Espandiamo il nodo visitando i suoi vicini e aggiornando (se necessario!) la loro distanza.

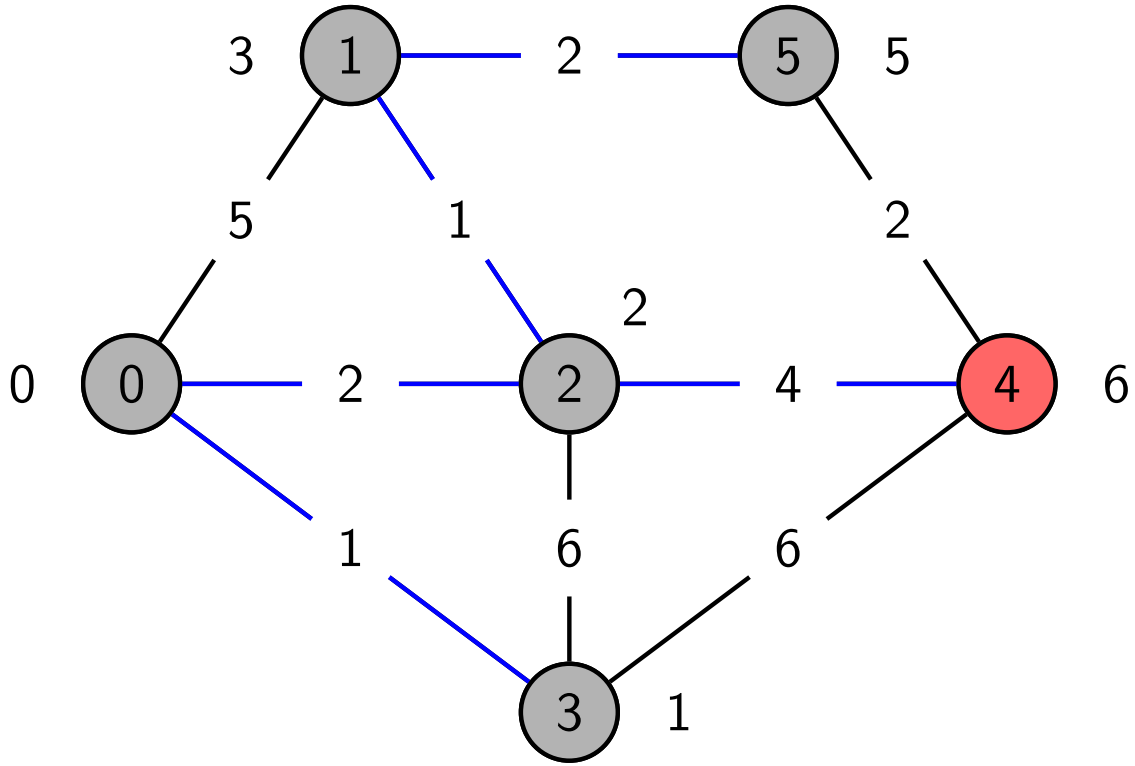


$5 + 2 \not< 6$

Algoritmo di Dijkstra

Algoritmo

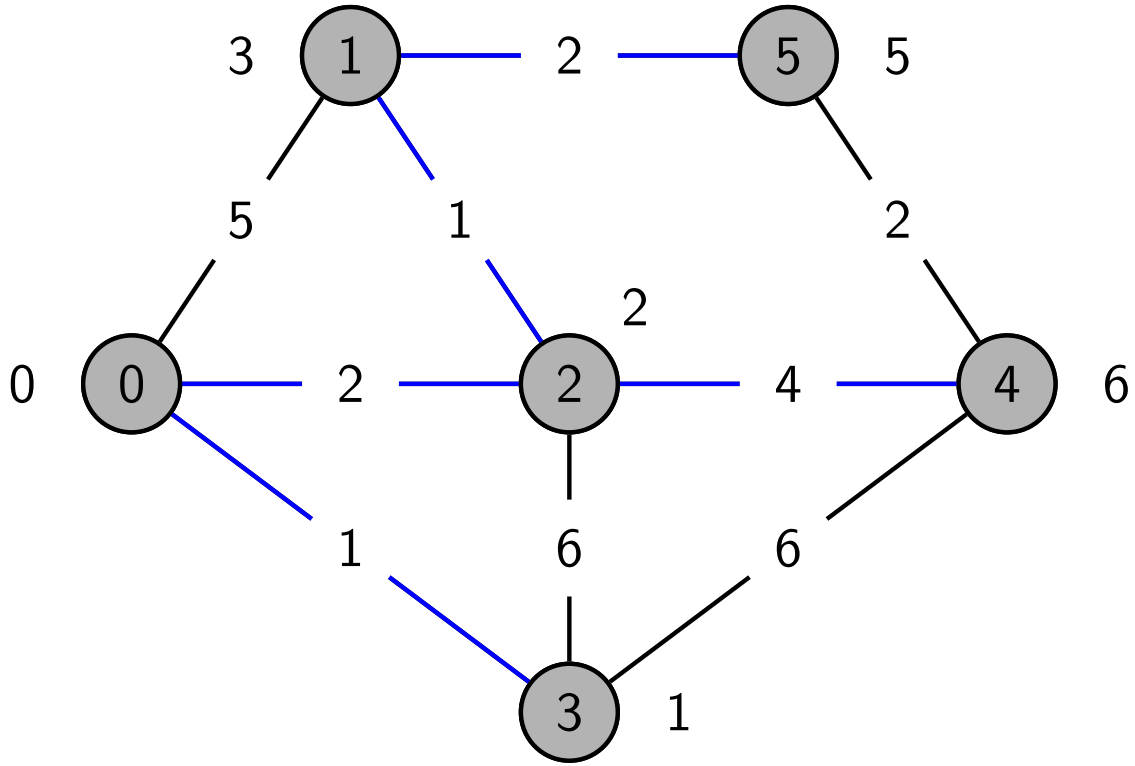
Selezioniamo il nodo non espanso con distanza minima.



Algoritmo di Dijkstra

Algoritmo

Tutti i nodi sono stati espansi, l'algoritmo termina.



Algoritmo di Dijkstra

Complessità

L'algoritmo effettua N step: ogni nodo viene espanso esattamente una volta.
All'interno di uno step si deve:

- Selezionare il nodo non visitato con distanza minima
- Visitare tutti i suoi vicini

Algoritmo di Dijkstra

Complessità

L'algoritmo effettua N step: ogni nodo viene espanso esattamente una volta.
All'interno di uno step si deve:

- Selezionare il nodo non visitato con distanza minima
- Visitare tutti i suoi vicini

Ogni arco viene processato esattamente una volta, ogni nodo viene processato esattamente una volta.
La complessità dell'algoritmo dipende da come viene trovato il vertice non ancora processato con distanza minore. Abbiamo quindi due casi:

Algoritmo di Dijkstra

Complessità

L'algoritmo effettua N step: ogni nodo viene espanso esattamente una volta.
All'interno di uno step si deve:

- Selezionare il nodo non visitato con distanza minima
- Visitare tutti i suoi vicini

Ogni arco viene processato esattamente una volta, ogni nodo viene processato esattamente una volta.
La complessità dell'algoritmo dipende da come viene trovato il vertice non ancora processato con distanza minore. Abbiamo quindi due casi:

- se si scorrono tutti i vertici, la complessità è $\mathcal{O}(N^2 + M) = \mathcal{O}(N^2)$;

Algoritmo di Dijkstra

Complessità

L'algoritmo effettua N step: ogni nodo viene espanso esattamente una volta.
All'interno di uno step si deve:

- Selezionare il nodo non visitato con distanza minima
- Visitare tutti i suoi vicini

Ogni arco viene processato esattamente una volta, ogni nodo viene processato esattamente una volta.
La complessità dell'algoritmo dipende da come viene trovato il vertice non ancora processato con distanza minore. Abbiamo quindi due casi:

- se si scorrono tutti i vertici, la complessità è $\mathcal{O}(N^2 + M) = \mathcal{O}(N^2)$;
- se si utilizza una priority queue, la complessità è invece $\mathcal{O}(M \log M)$.

Algoritmo di Bellman-Ford

L'algoritmo di Bellman-Ford ha un costo computazionalmente maggiore di quello di Dijkstra, ma risolve il problema anche nel caso in cui ci sono dei pesi degli archi negativi e può essere sfruttato per trovare cicli di peso negativo.

Algoritmo di Bellman-Ford

L'algoritmo di Bellman-Ford ha un costo computazionalmente maggiore di quello di Dijkstra, ma risolve il problema anche nel caso in cui ci sono dei pesi degli archi negativi e può essere sfruttato per trovare cicli di peso negativo.

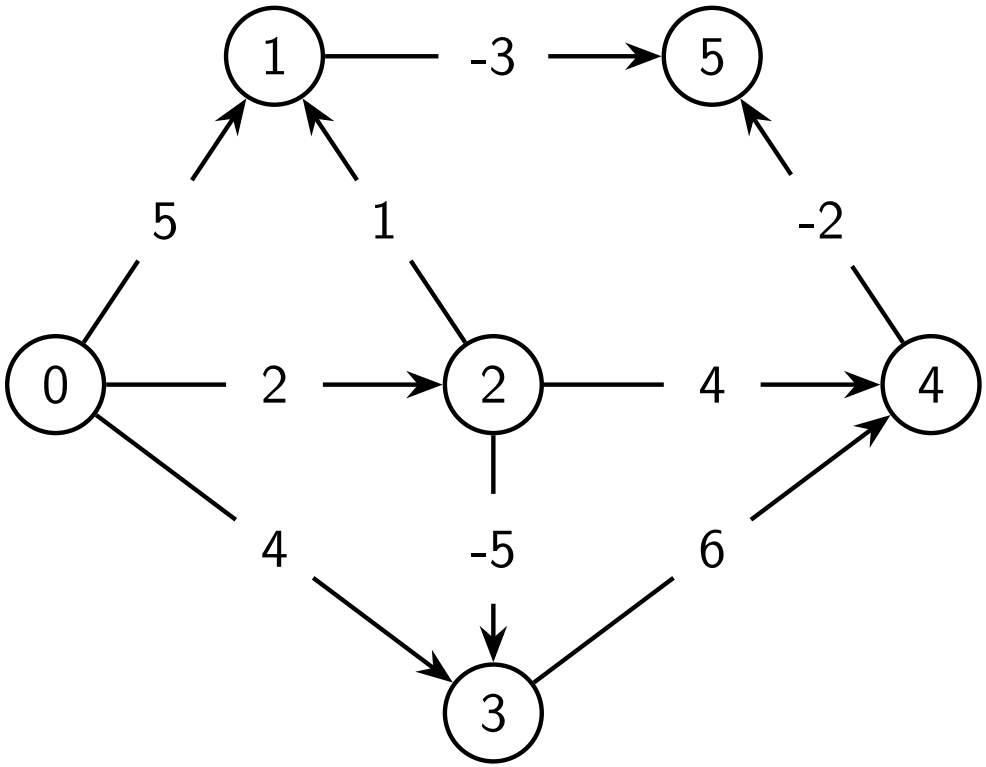
Osservazione

Se un grafo non orientato ha pesi negativi in alcuni archi, allora avrà sempre "cicli" di peso negativo.
Quindi ci interessiamo del caso orientato.

Algoritmo di Bellman-Ford

Algoritmo

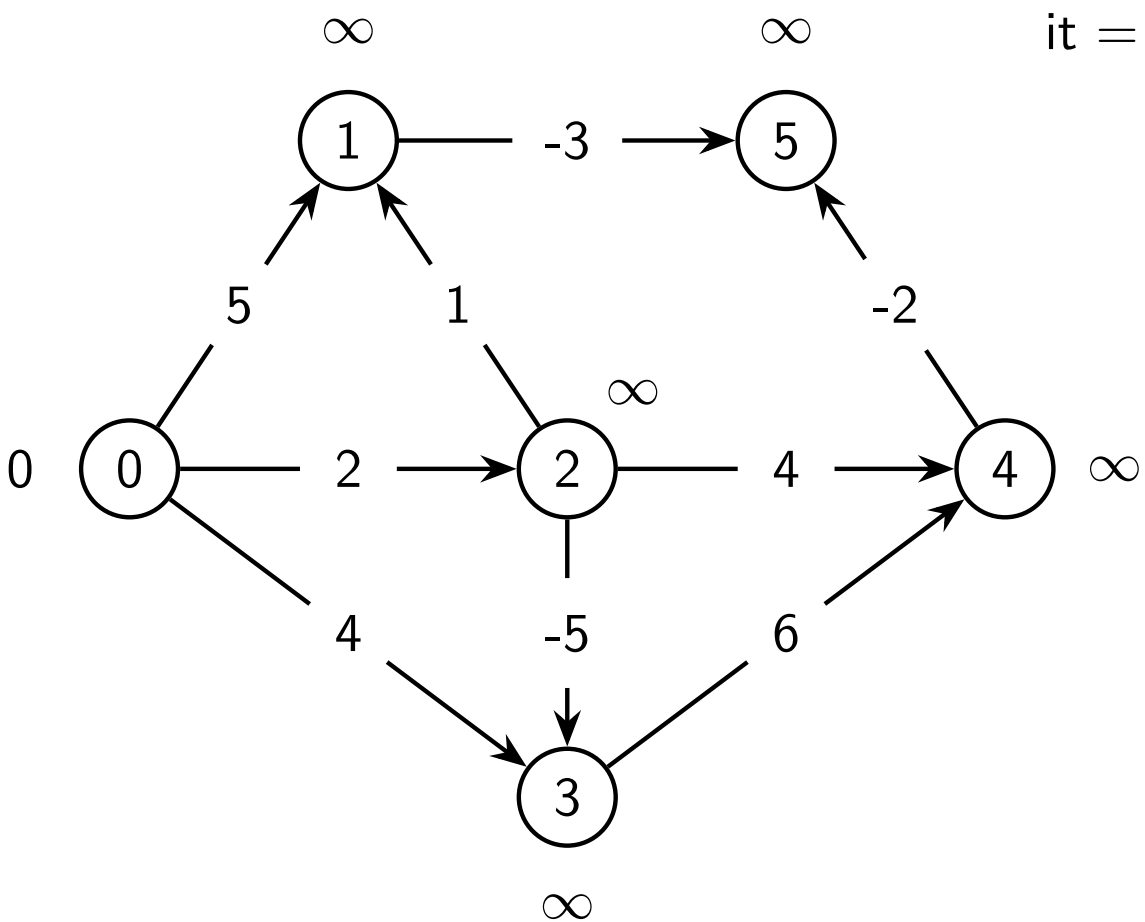
Vogliamo calcolare la distanza minima dal nodo 0 verso tutti gli altri nodi.



Algoritmo di Bellman-Ford

Algoritmo

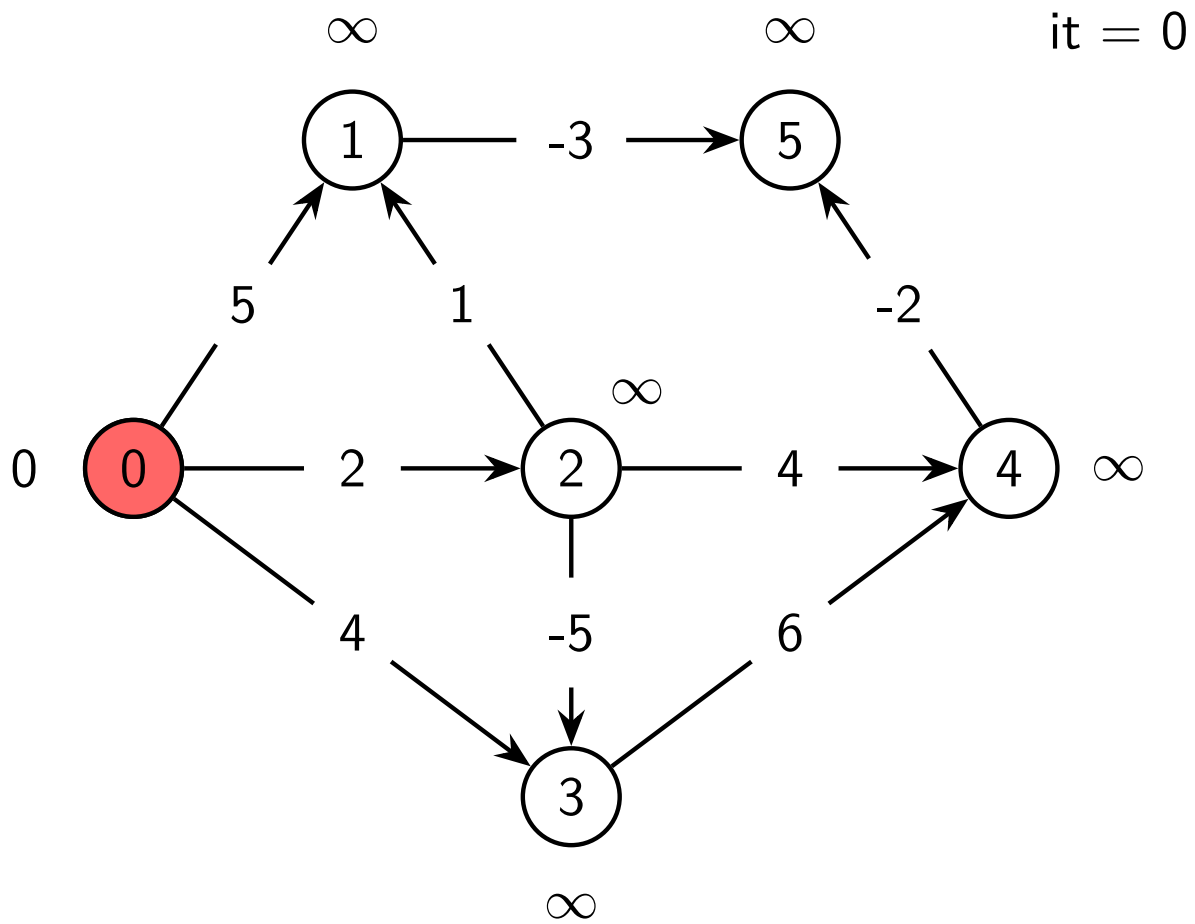
Inizializzazione: la distanza di tutti i nodi è ∞, tranne per la sorgente.



Algoritmo di Bellman-Ford

Algoritmo

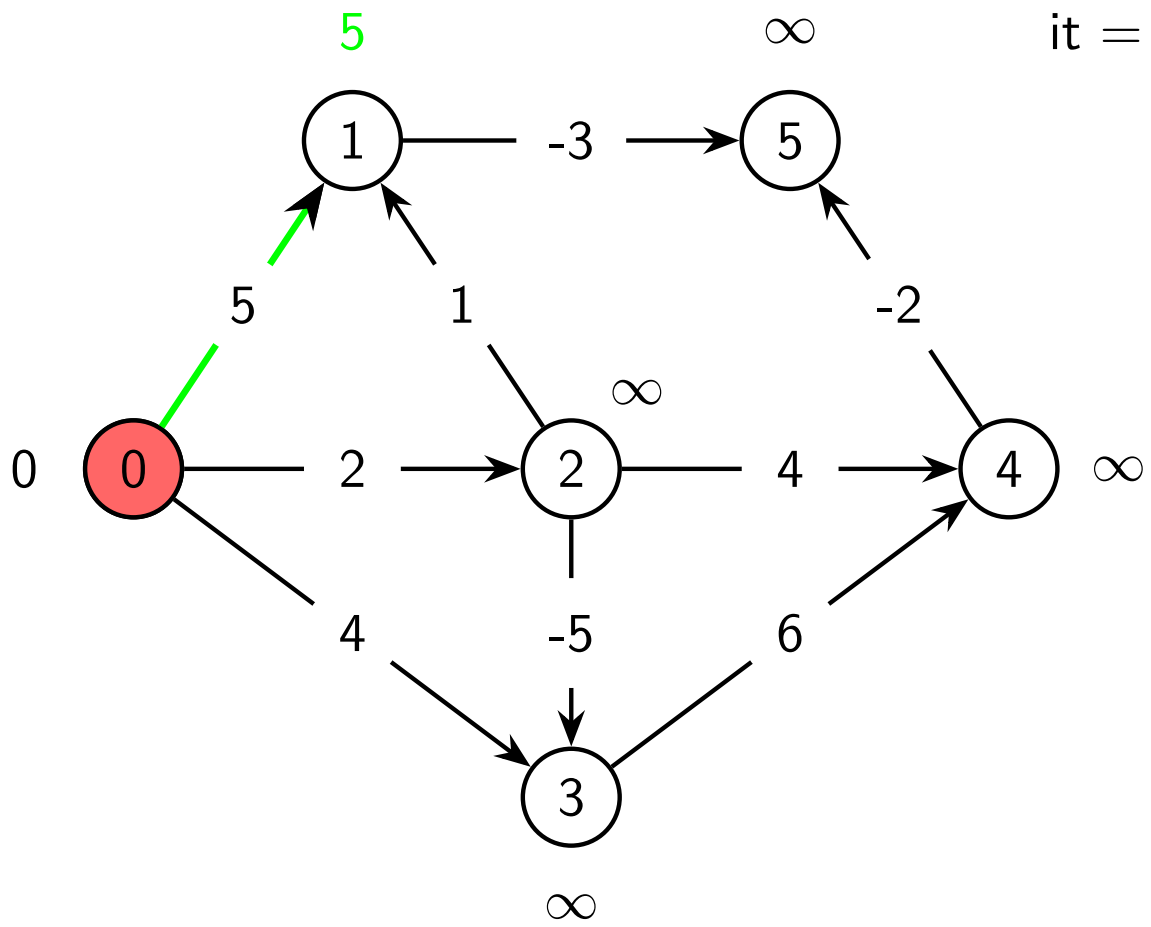
Processiamo in un qualunque ordine tutti i nodi



Algoritmo di Bellman-Ford

Algoritmo

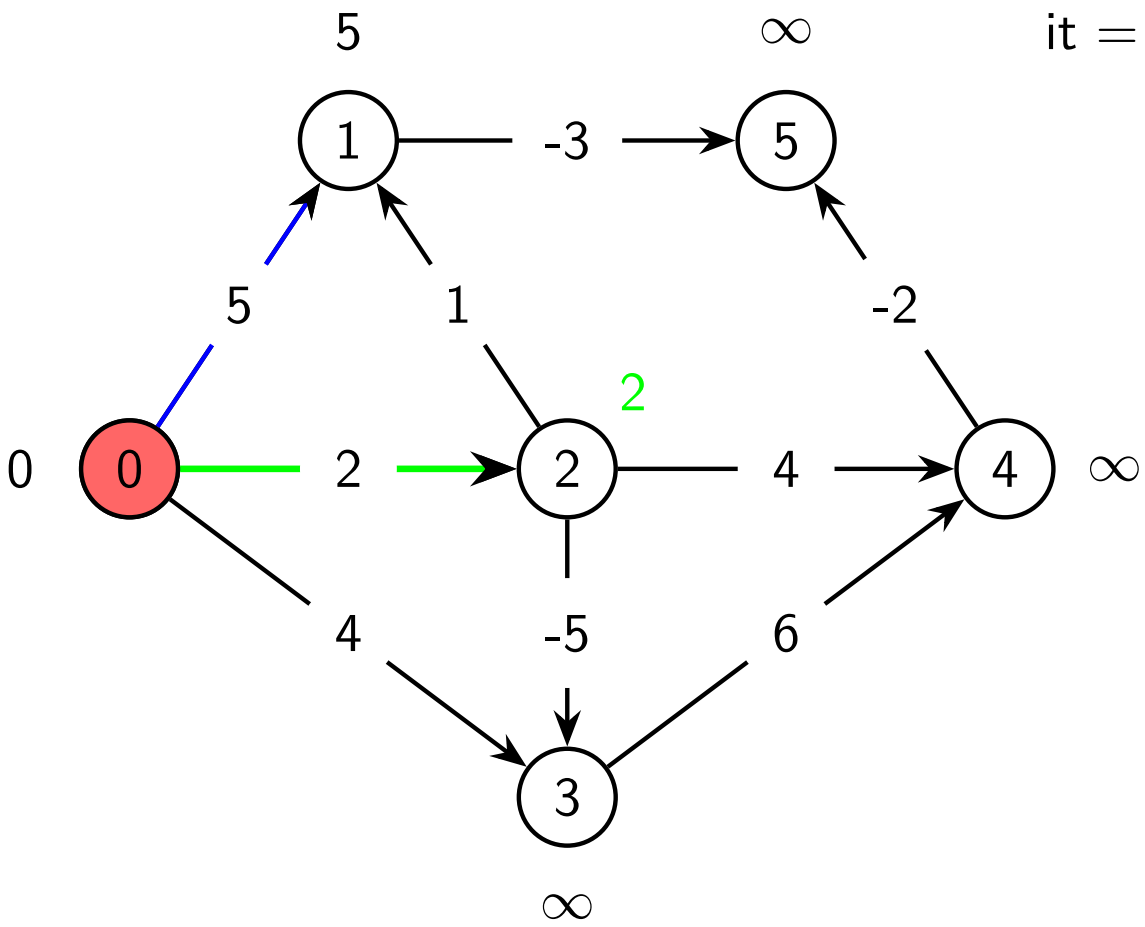
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

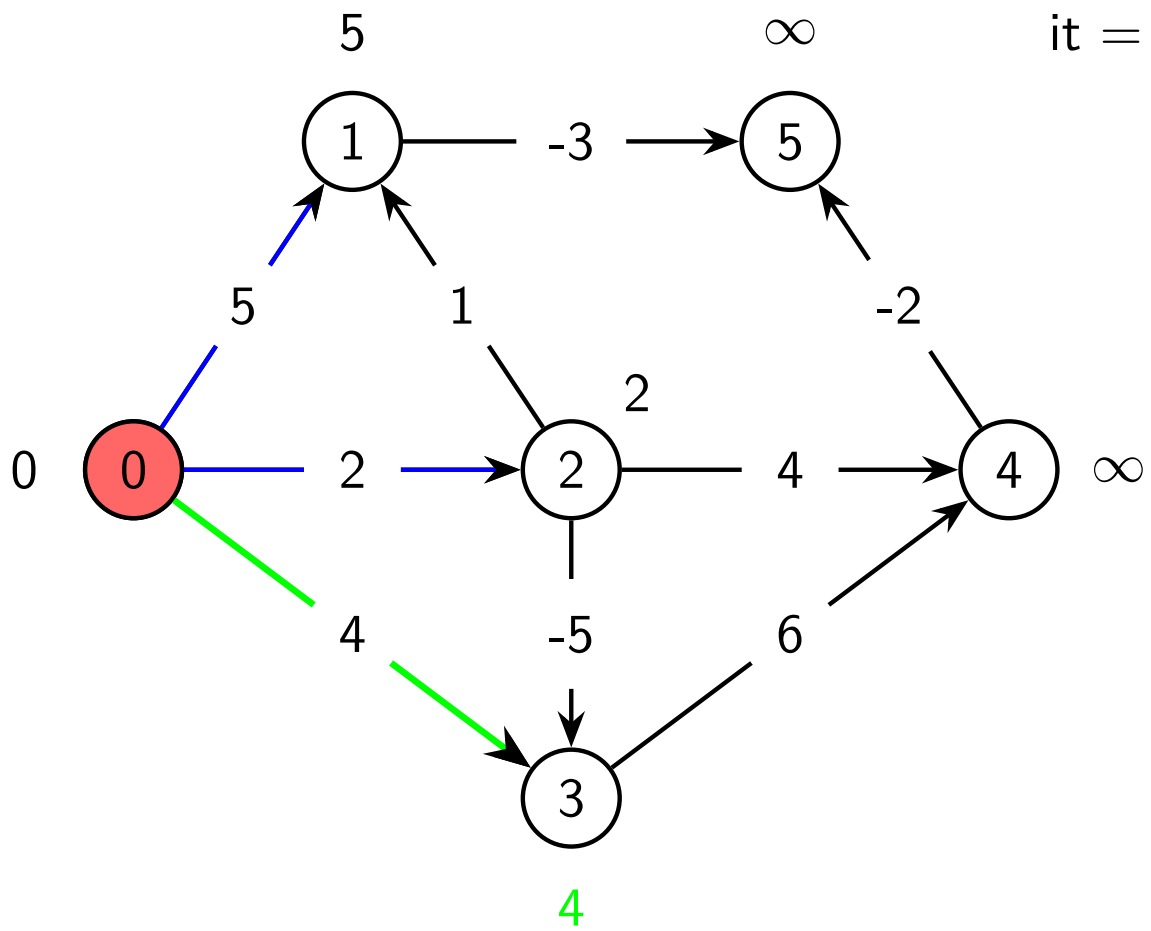
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

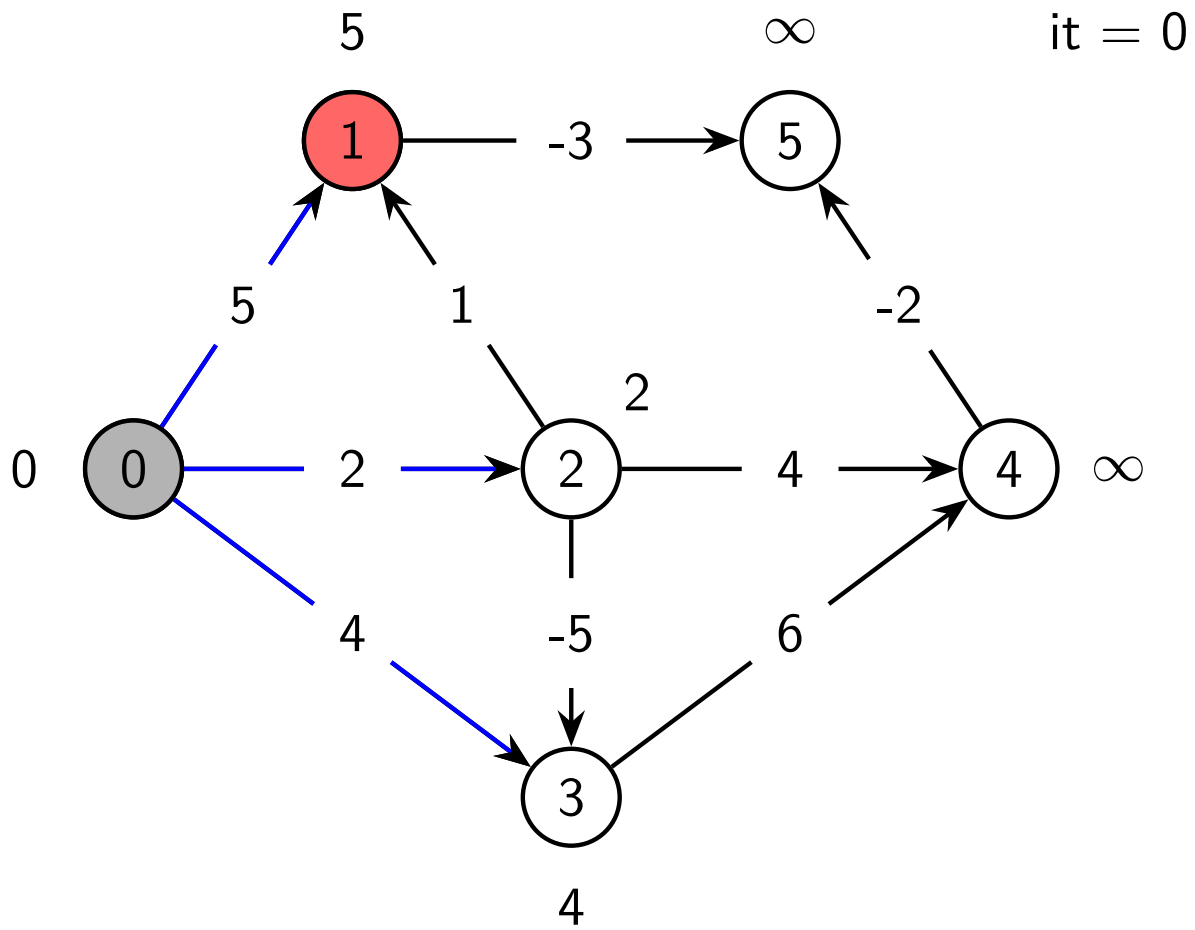
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

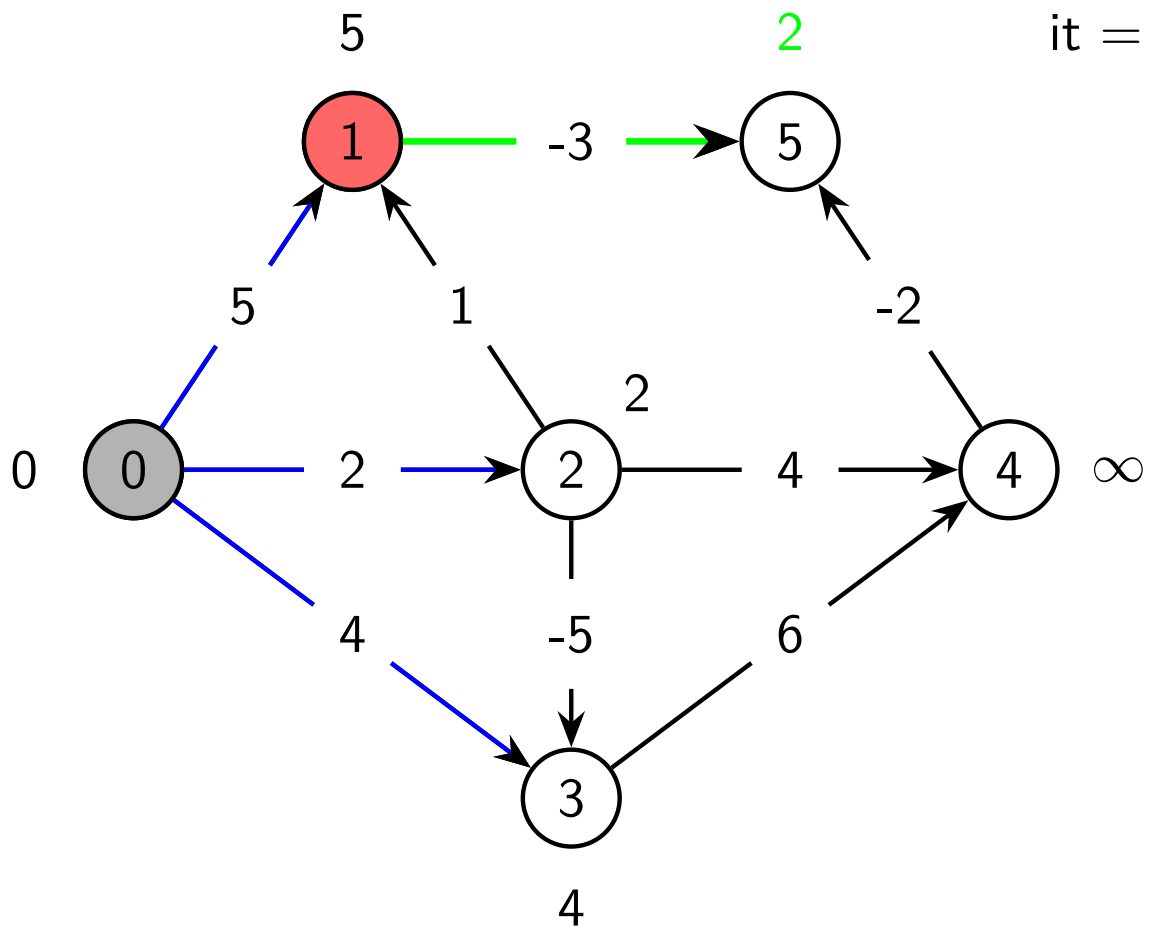
Processiamo in un qualunque ordine tutti i nodi



Algoritmo di Bellman-Ford

Algoritmo

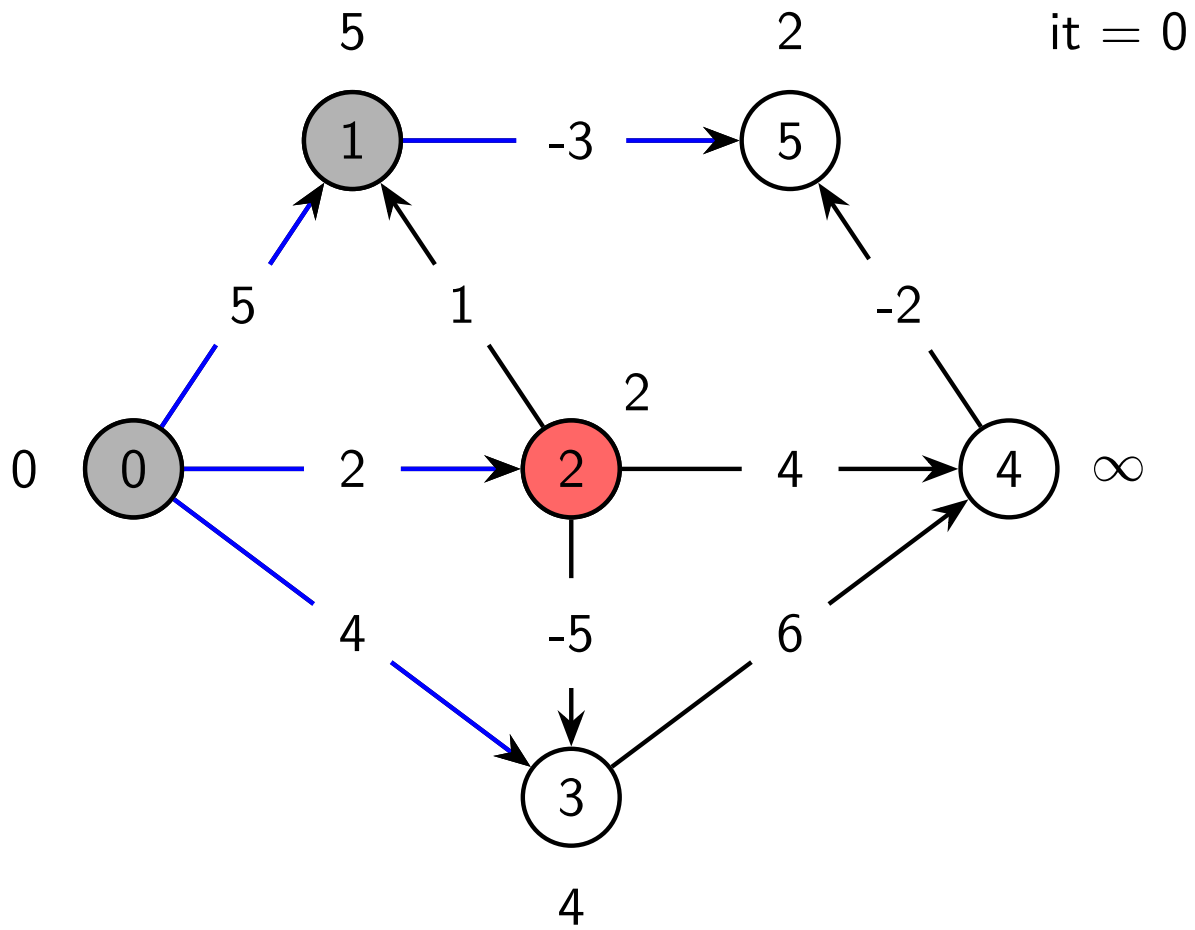
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

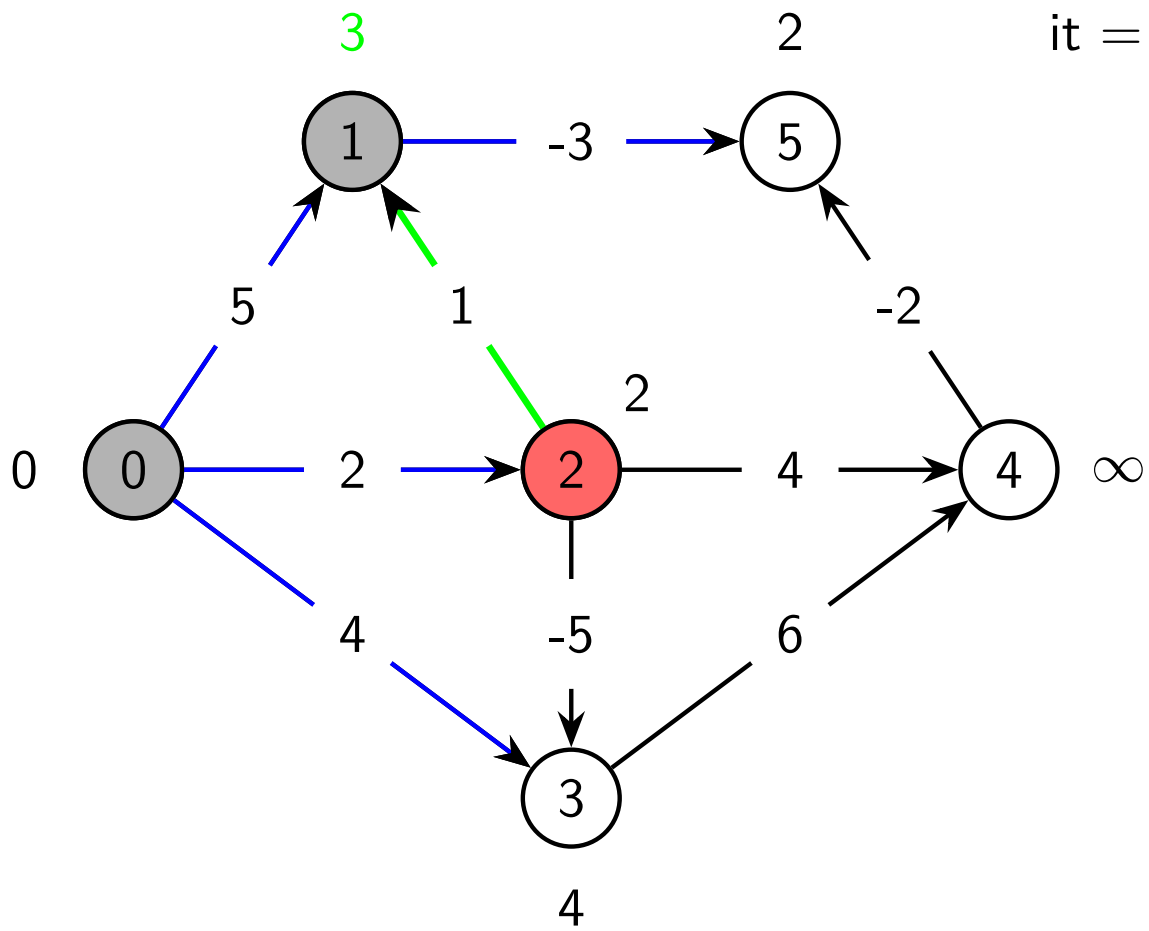
Processiamo in un qualunque ordine tutti i nodi



Algoritmo di Bellman-Ford

Algoritmo

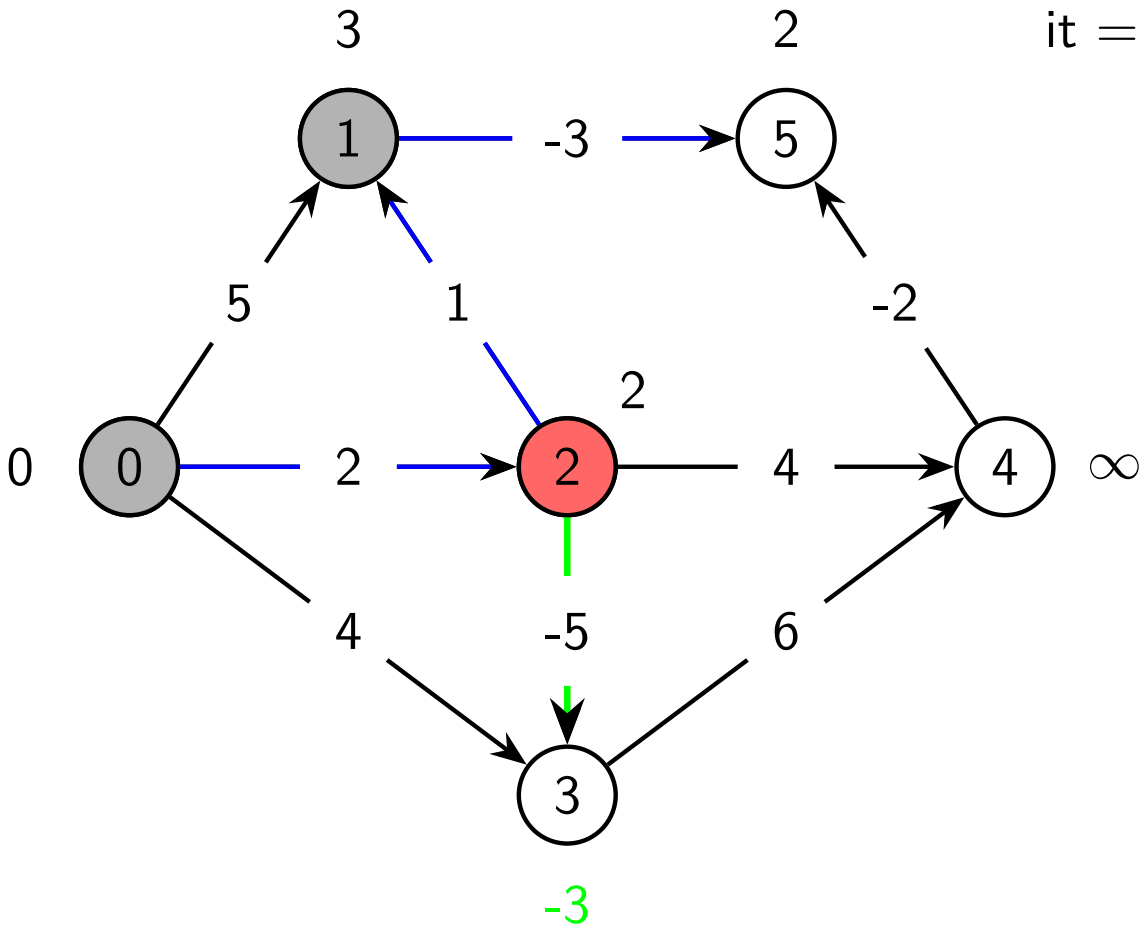
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

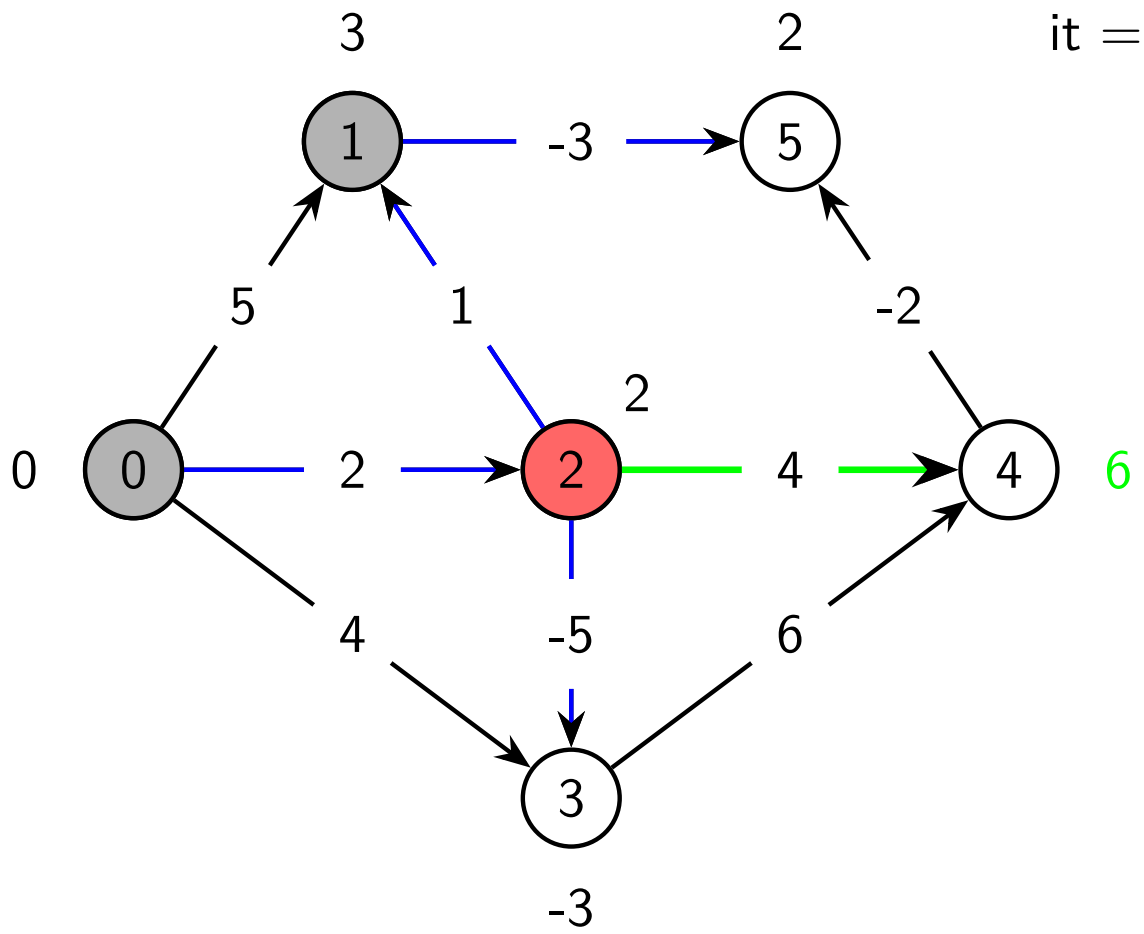
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

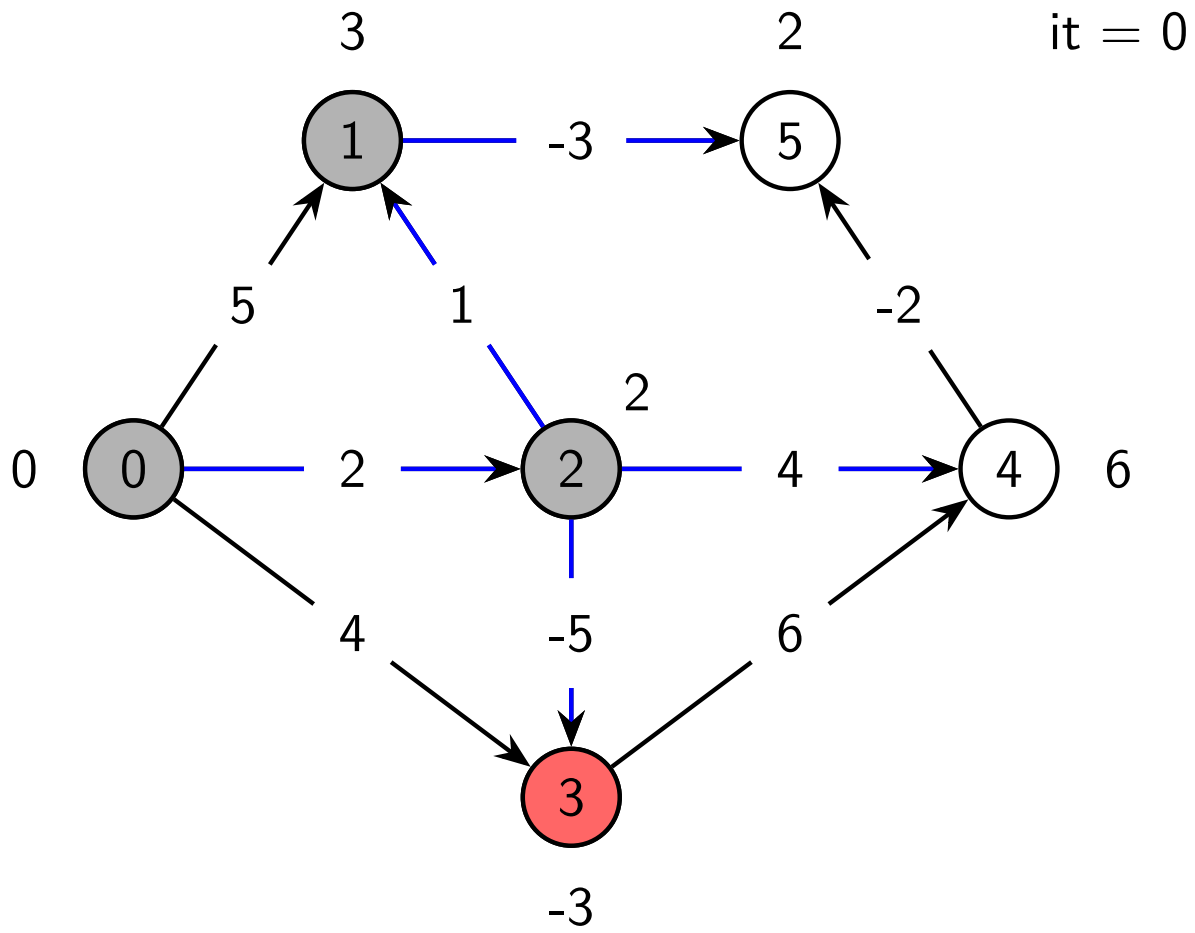
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

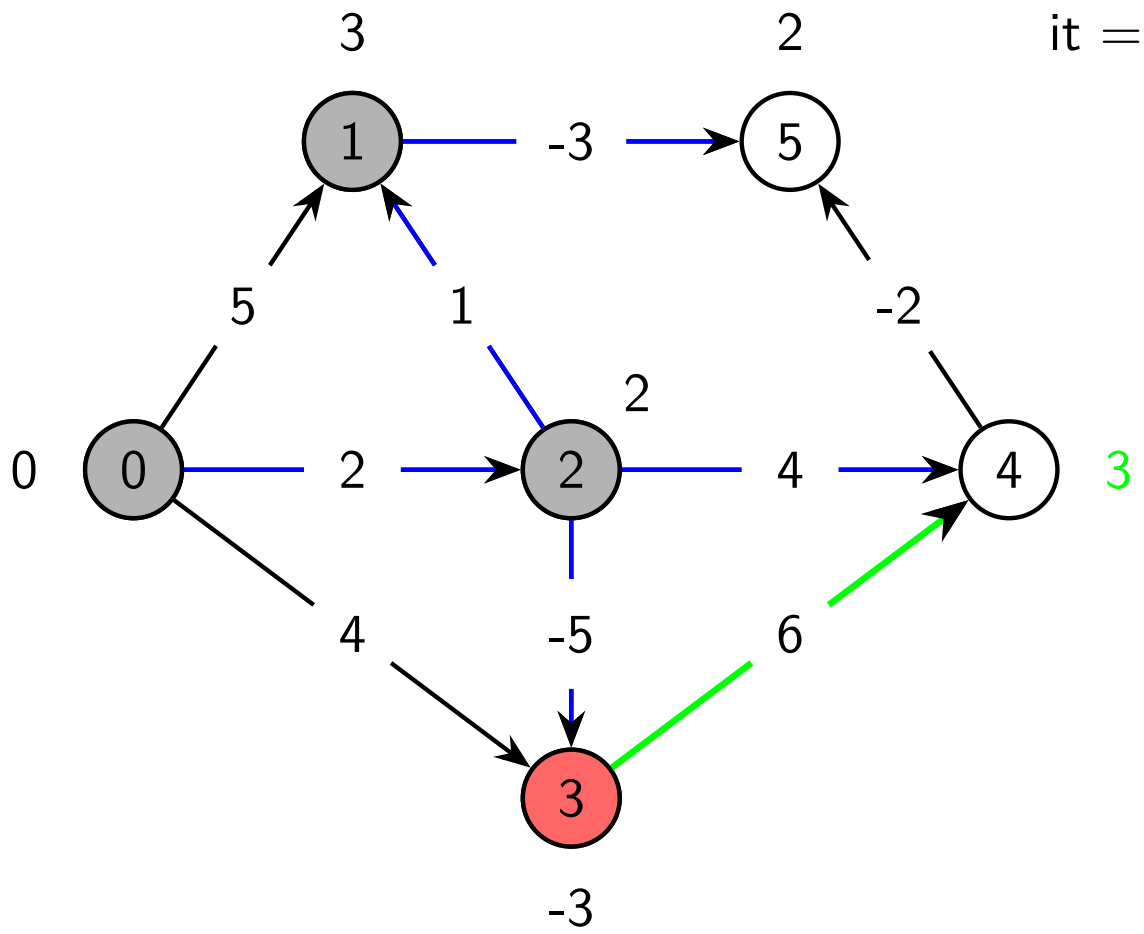
Processiamo in un qualunque ordine tutti i nodi



Algoritmo di Bellman-Ford

Algoritmo

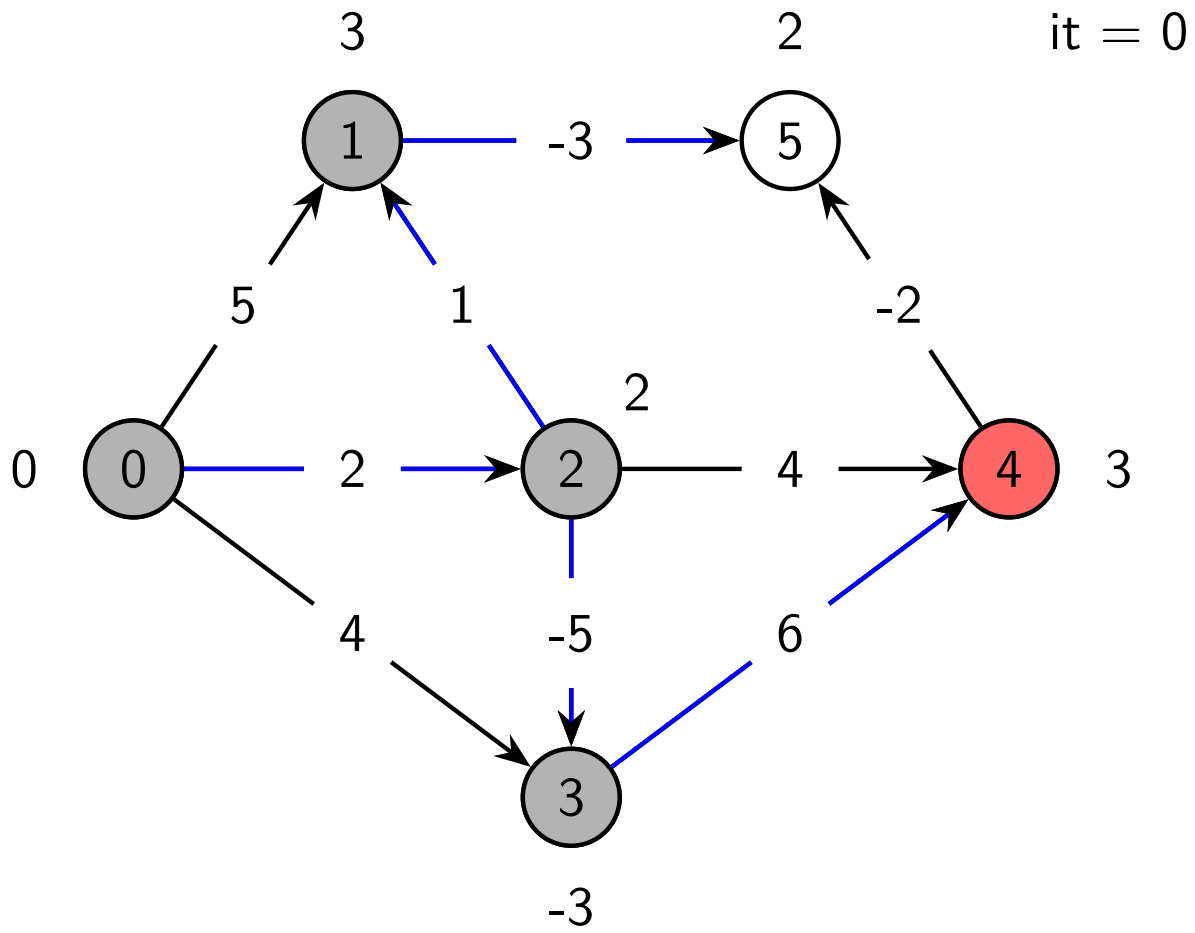
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

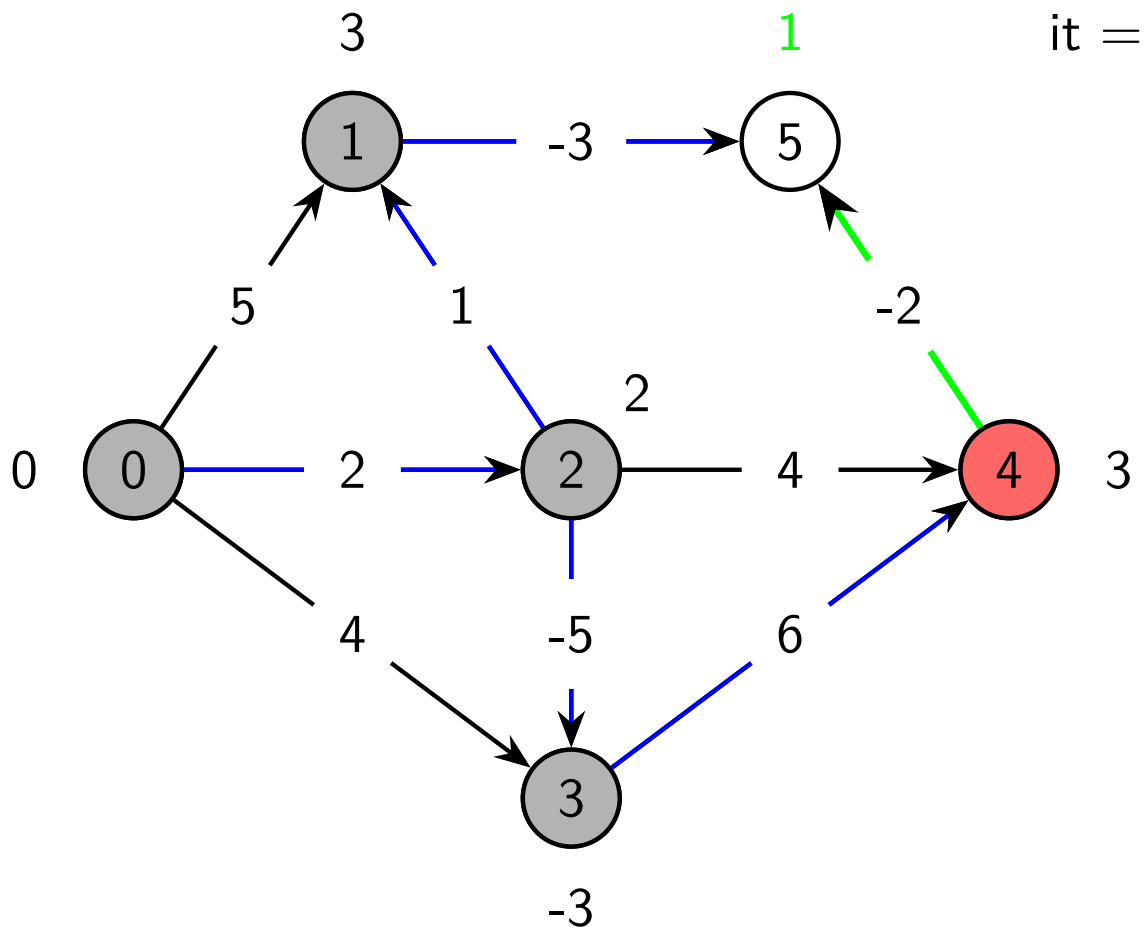
Processiamo in un qualunque ordine tutti i nodi



Algoritmo di Bellman-Ford

Algoritmo

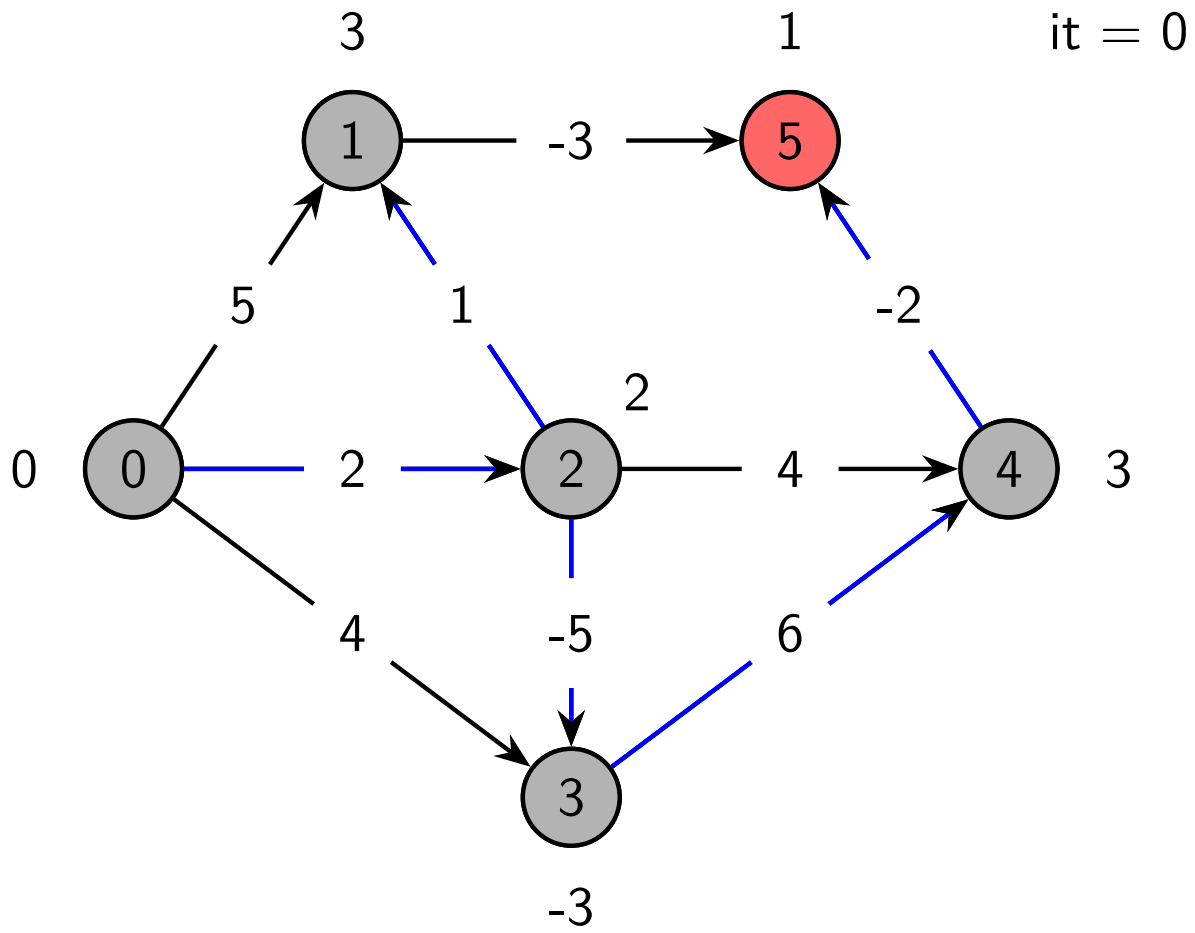
Espandiamo come nelle altre visite ogni arco, aggiornando la distanza se necessario.



Algoritmo di Bellman-Ford

Algoritmo

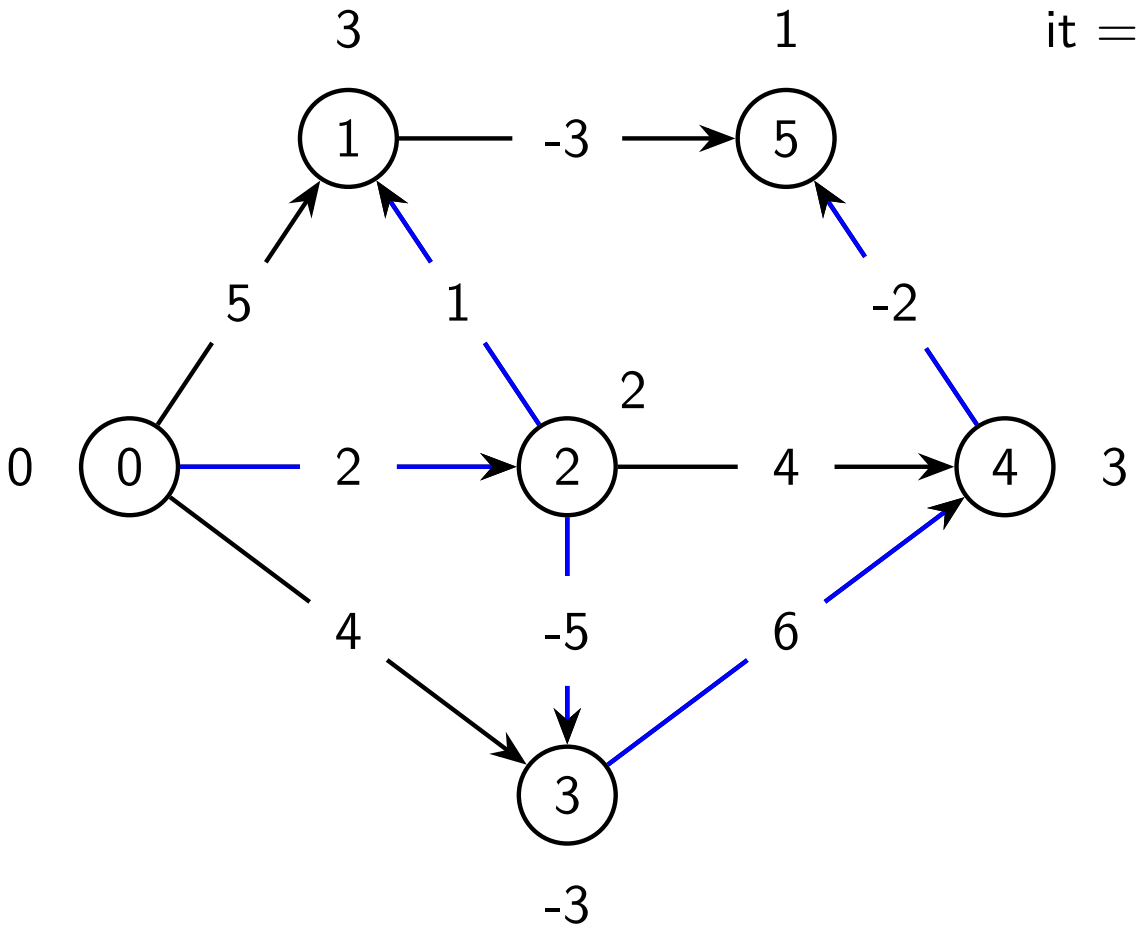
Processiamo in un qualunque ordine tutti i nodi



Algoritmo di Bellman-Ford

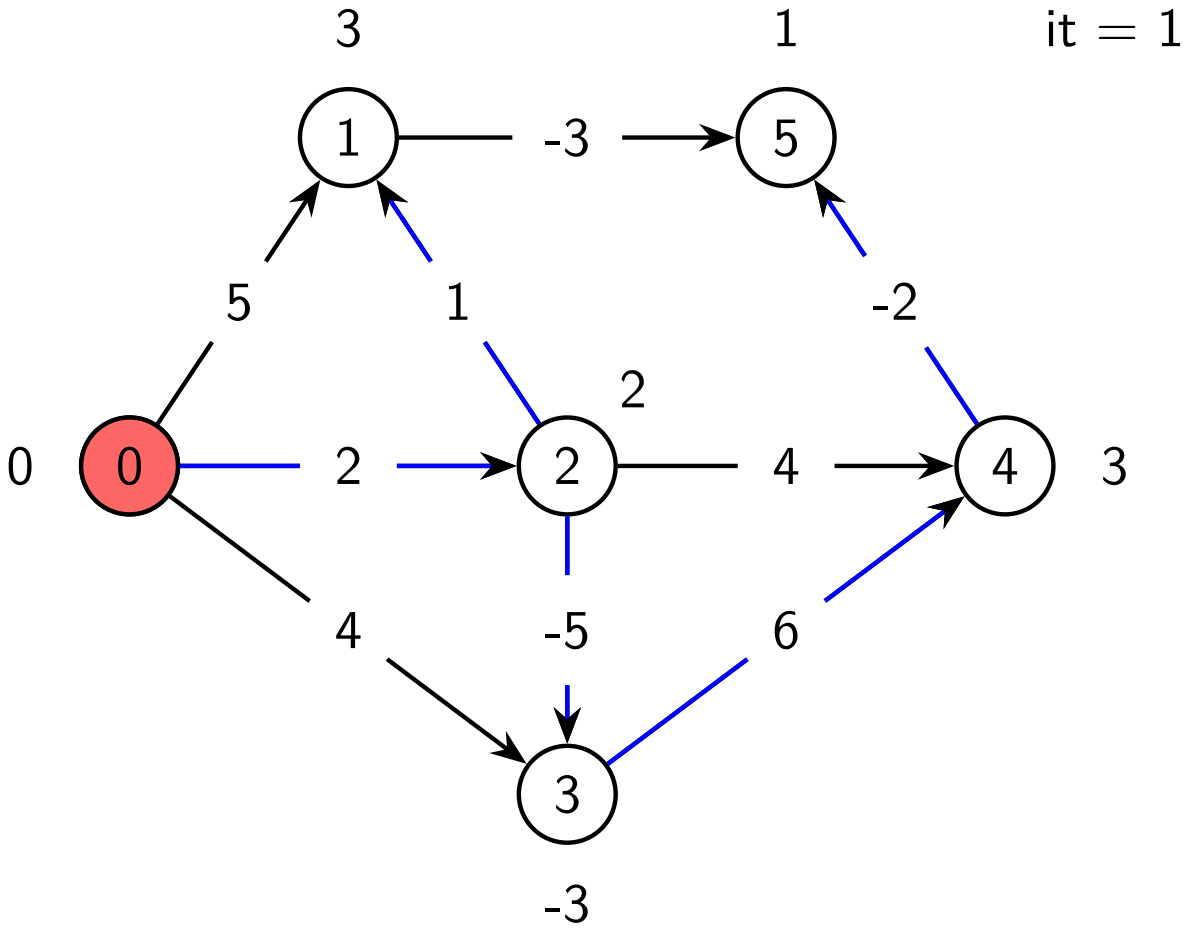
Algoritmo

La prima iterazione si è conclusa. Ora si riparte dal primo nodo e si ripete la visita di tutti i nodi.



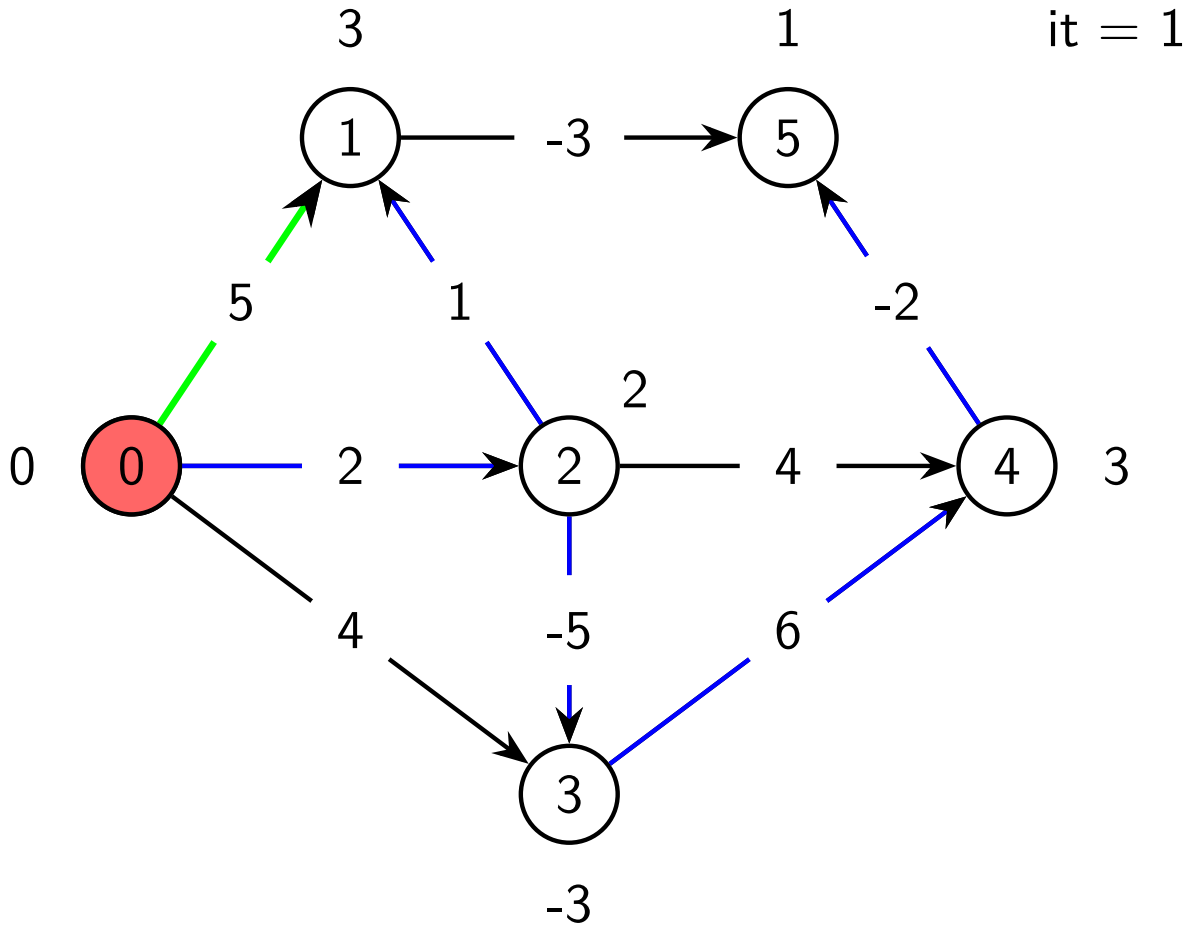
Algoritmo di Bellman-Ford

Algoritmo



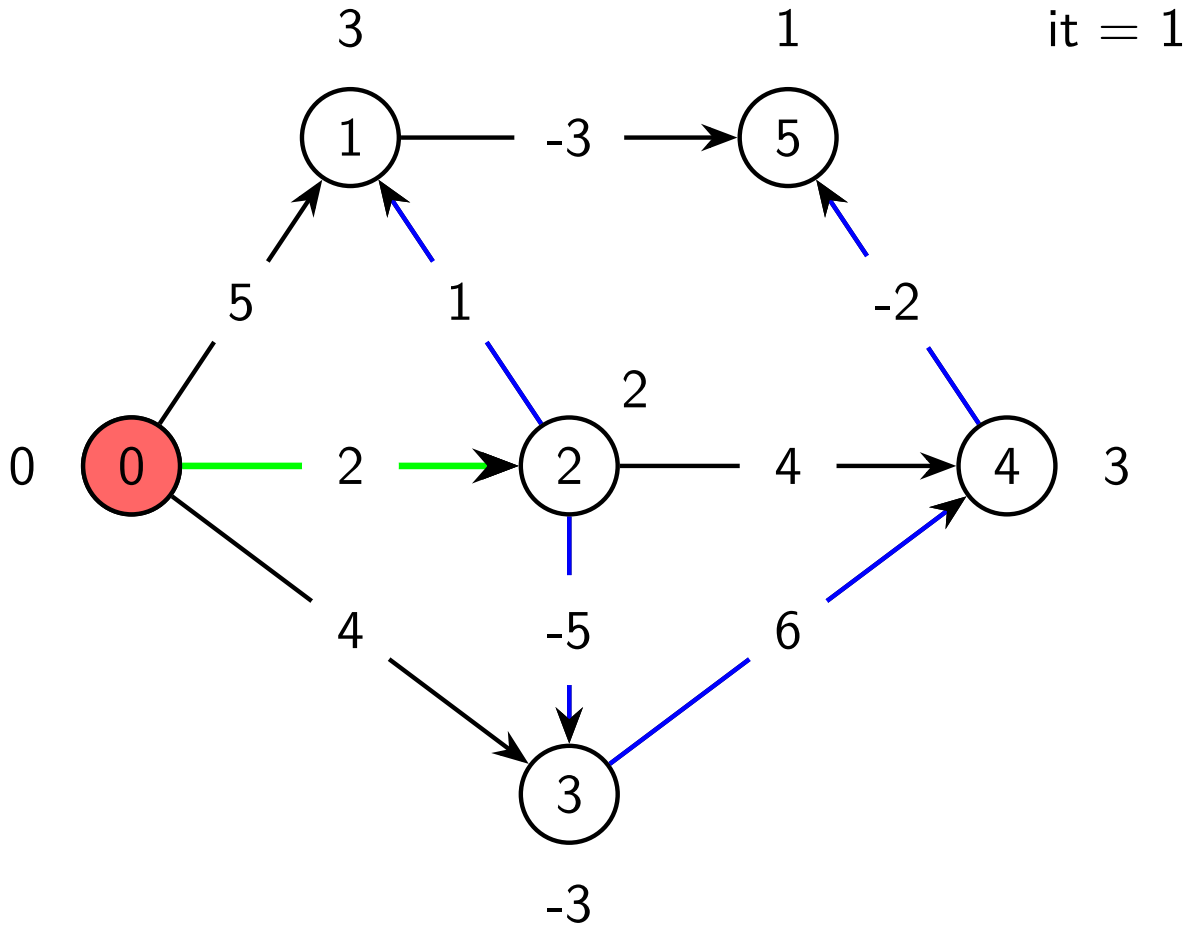
Algoritmo di Bellman-Ford

Algoritmo



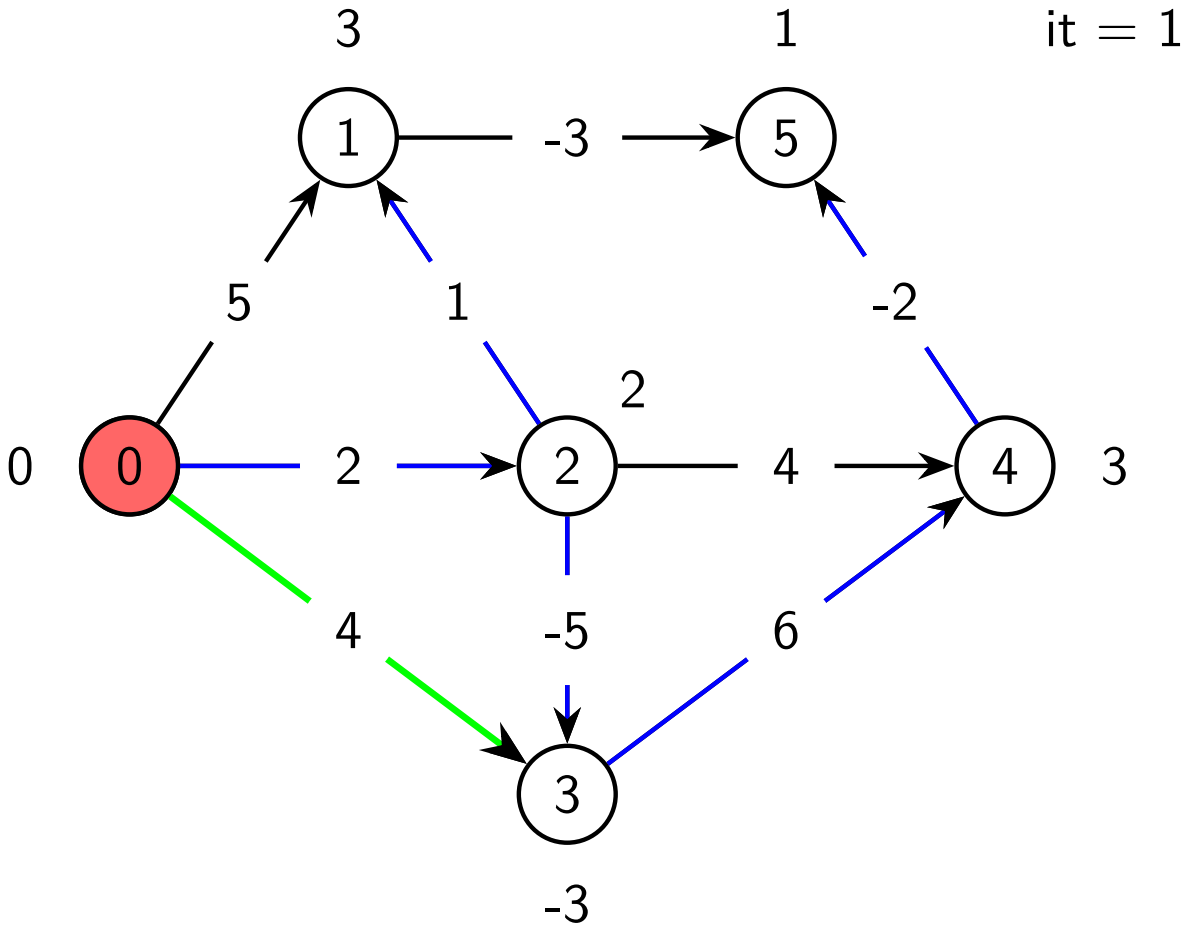
Algoritmo di Bellman-Ford

Algoritmo



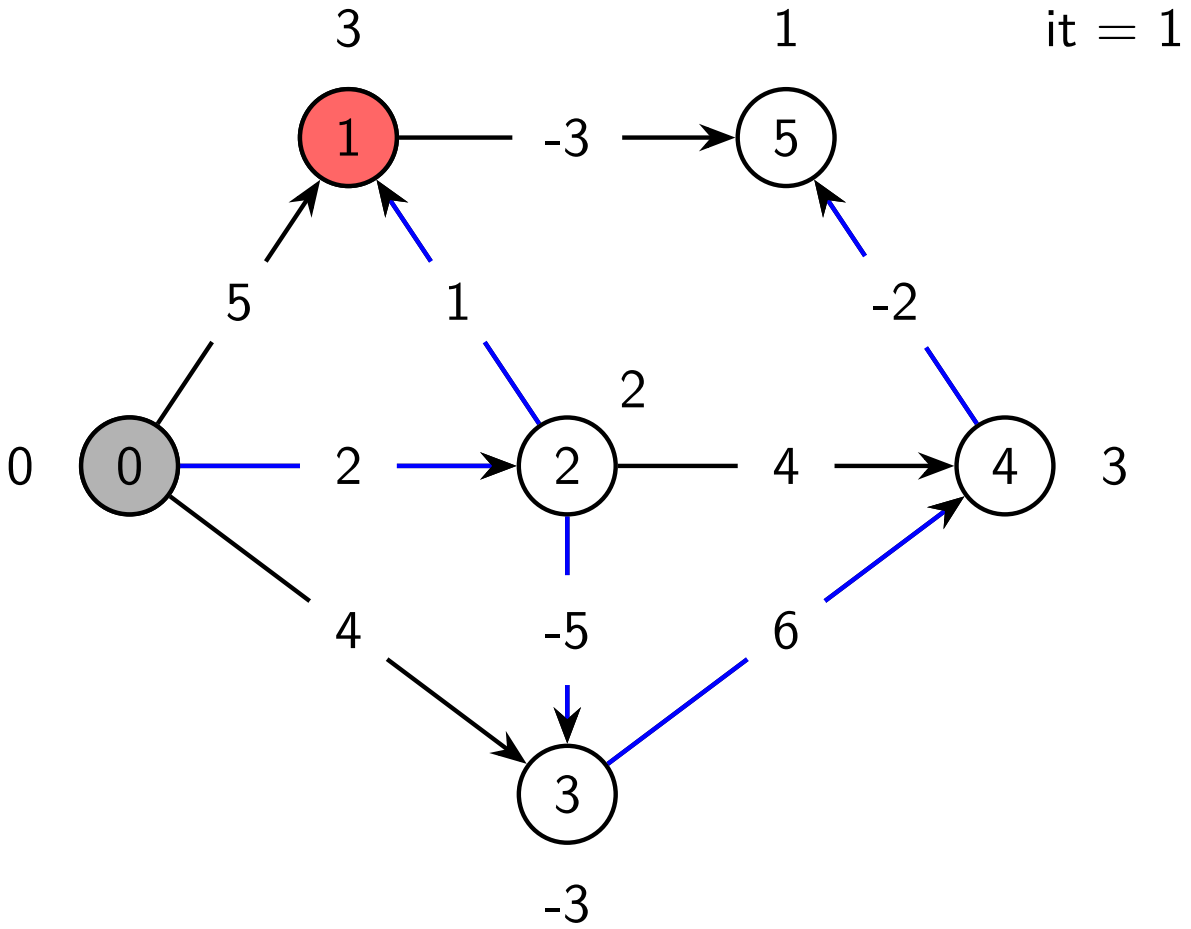
Algoritmo di Bellman-Ford

Algoritmo



Algoritmo di Bellman-Ford

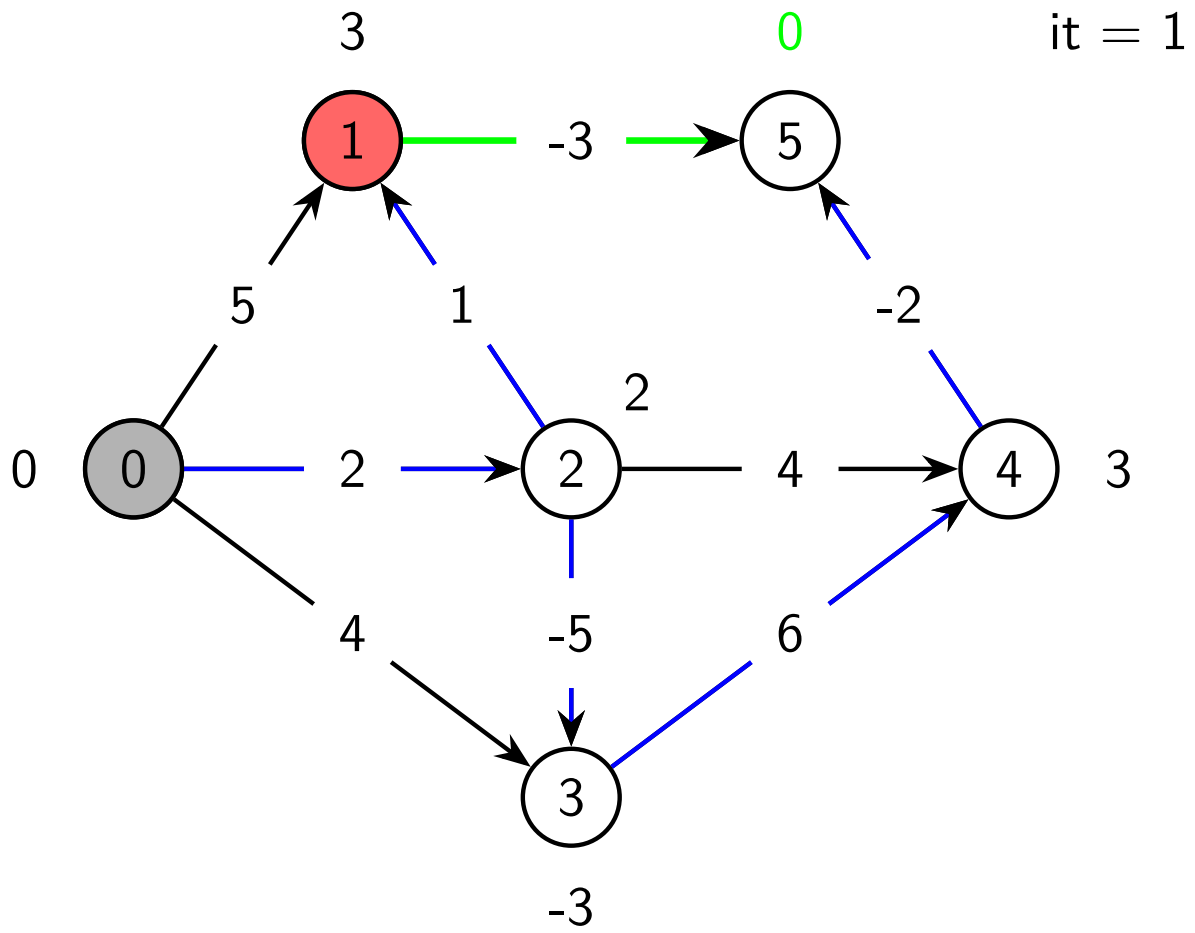
Algoritmo



Algoritmo di Bellman-Ford

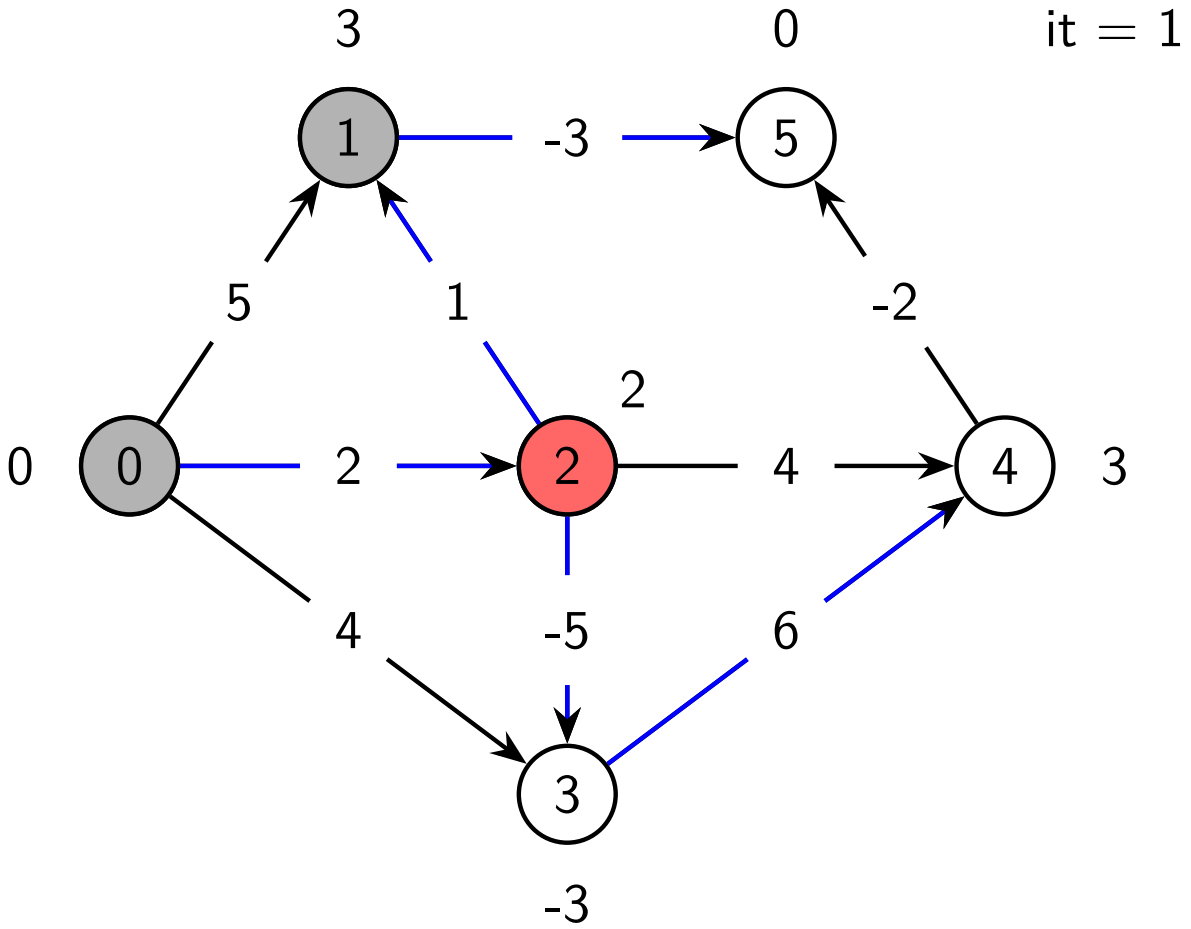
Algoritmo

Alcune distanza potrebbero cambiare.



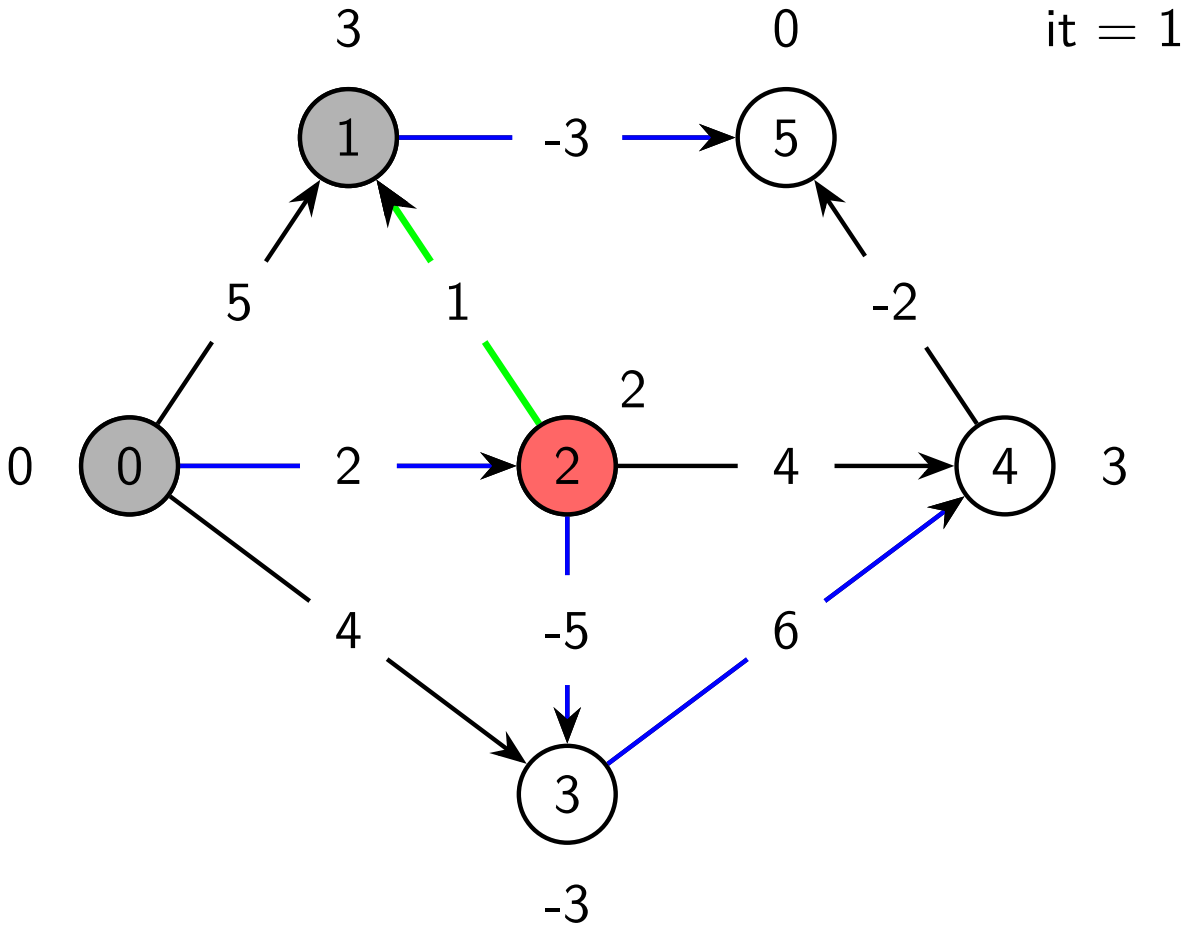
Algoritmo di Bellman-Ford

Algoritmo



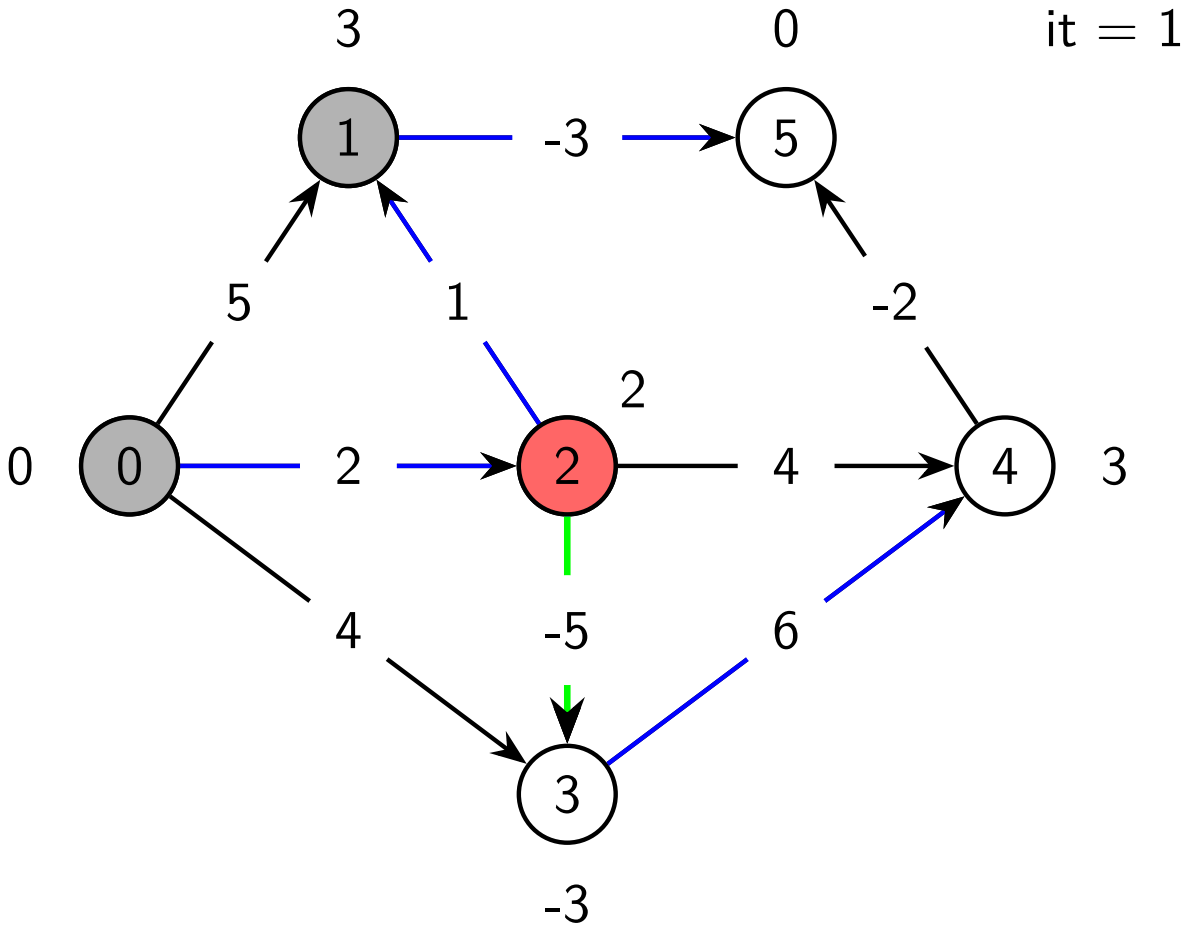
Algoritmo di Bellman-Ford

Algoritmo



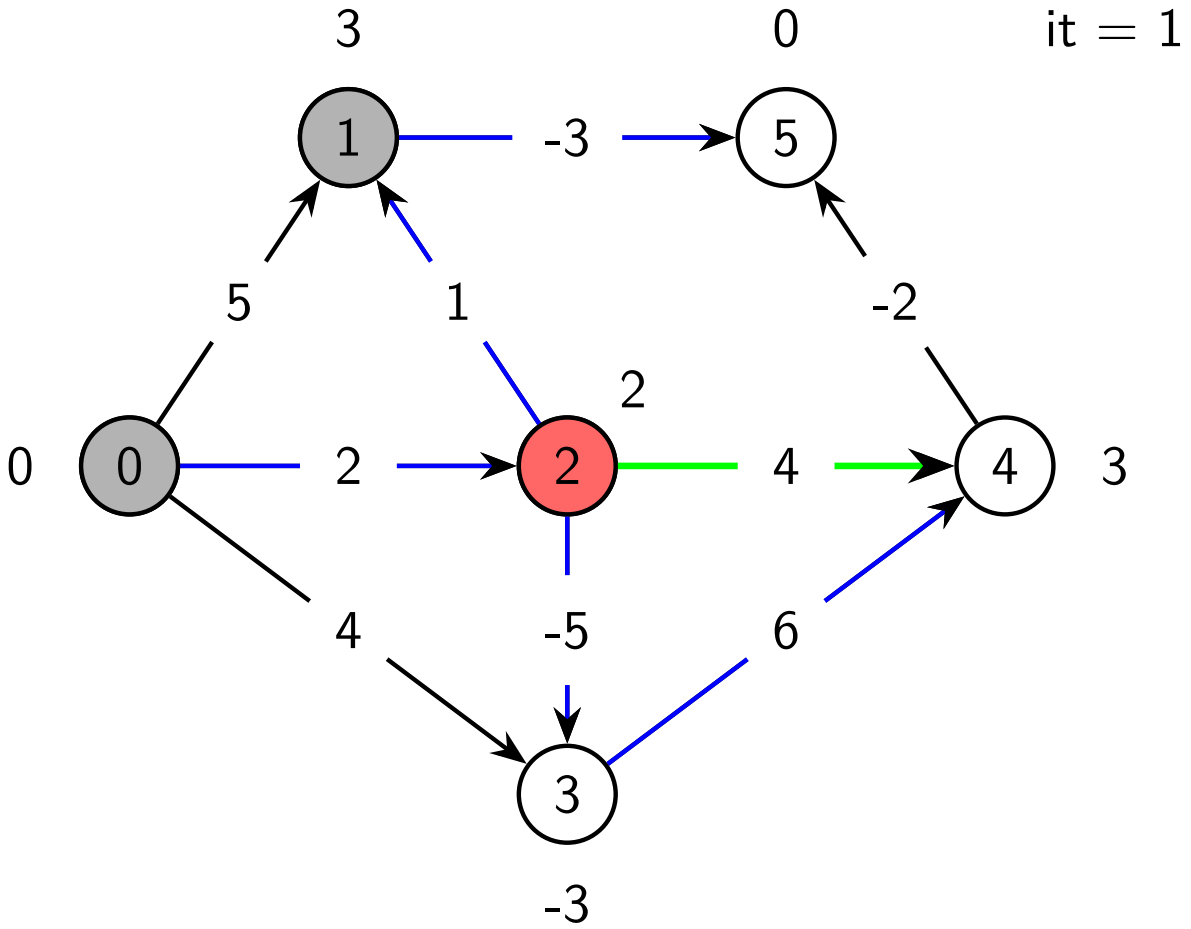
Algoritmo di Bellman-Ford

Algoritmo



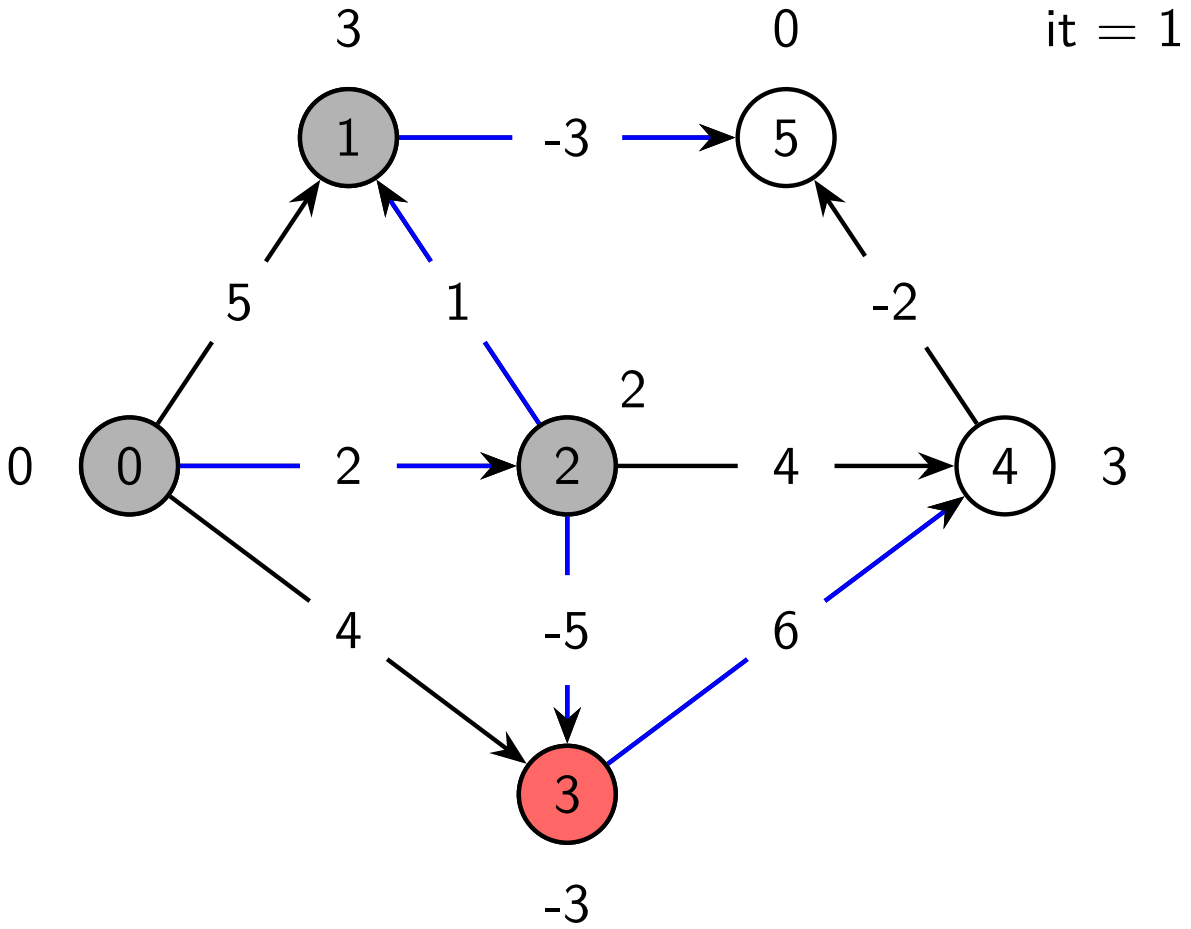
Algoritmo di Bellman-Ford

Algoritmo



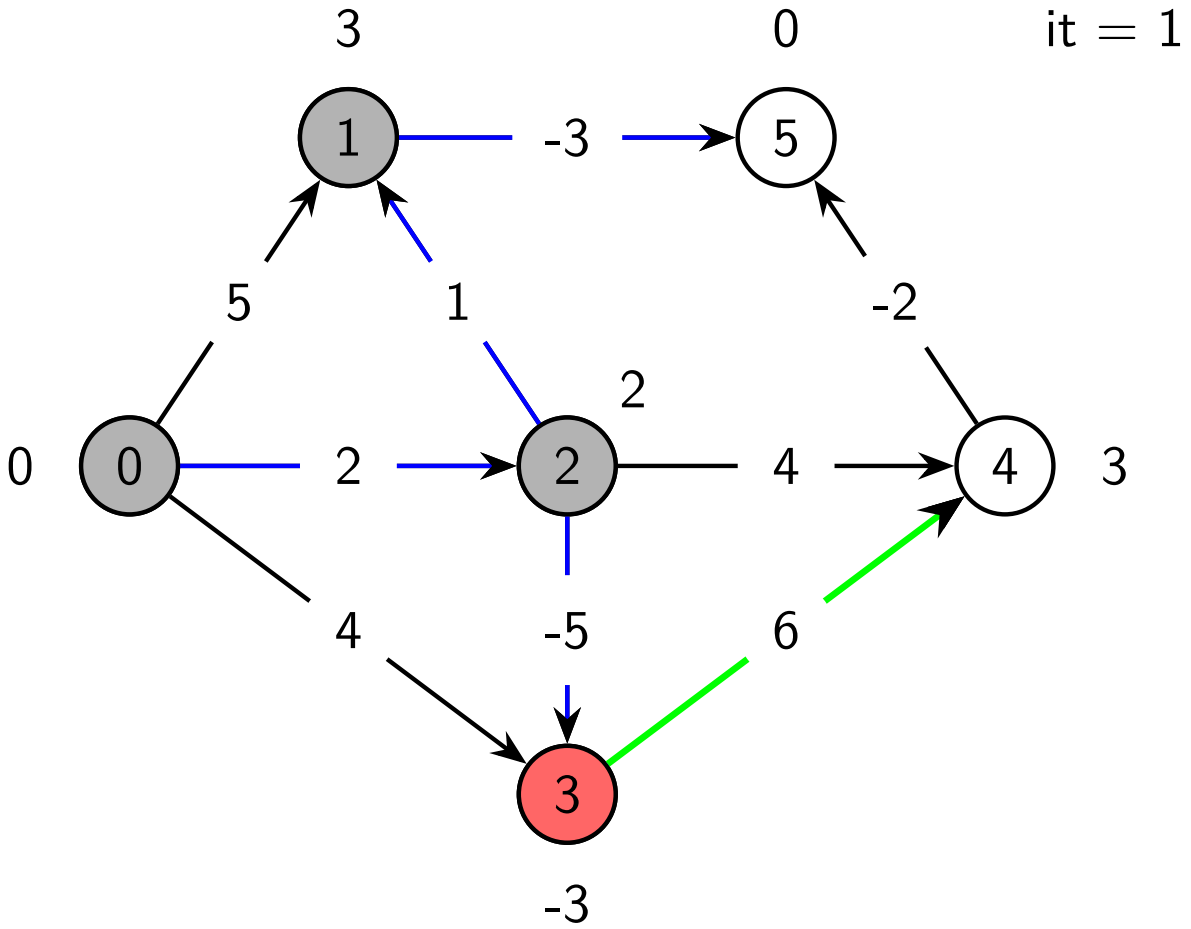
Algoritmo di Bellman-Ford

Algoritmo



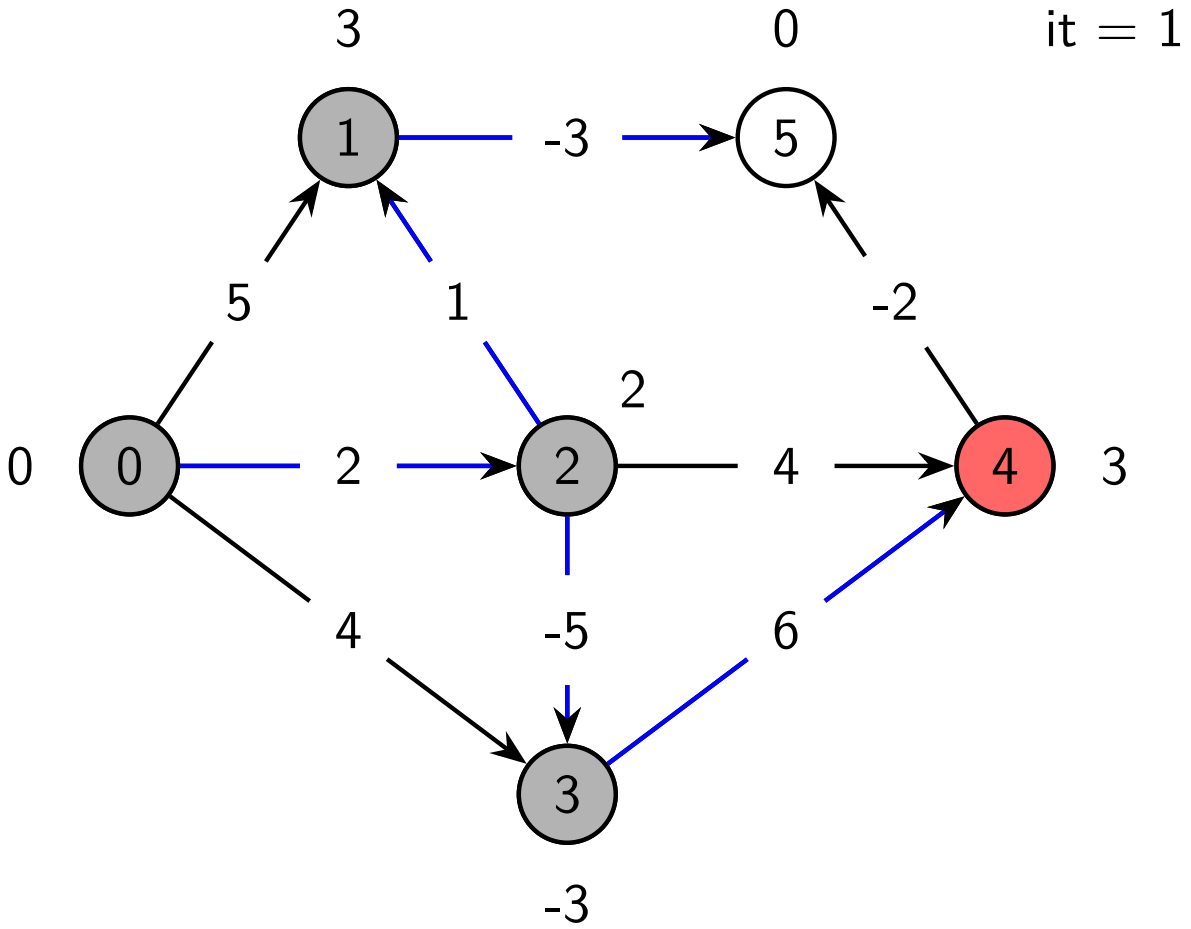
Algoritmo di Bellman-Ford

Algoritmo



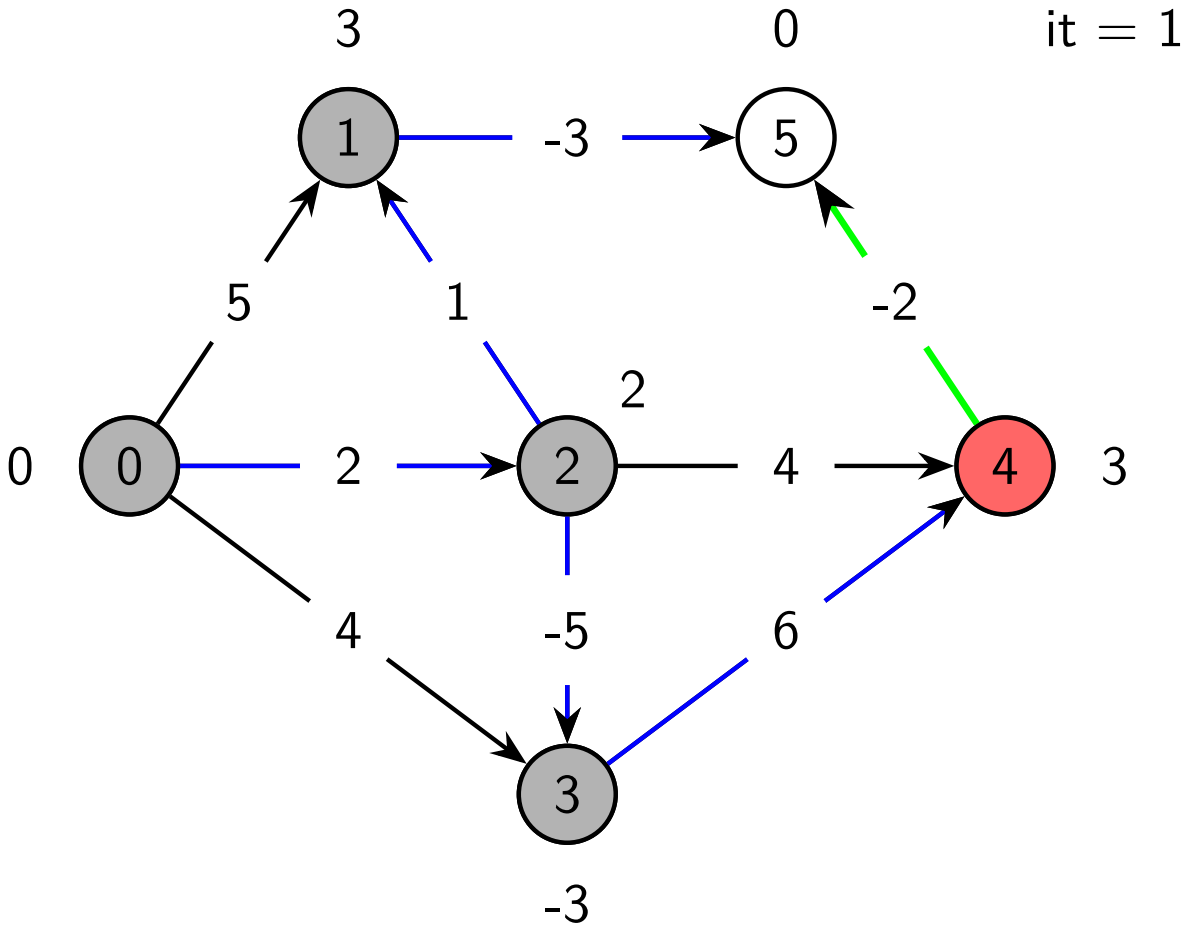
Algoritmo di Bellman-Ford

Algoritmo



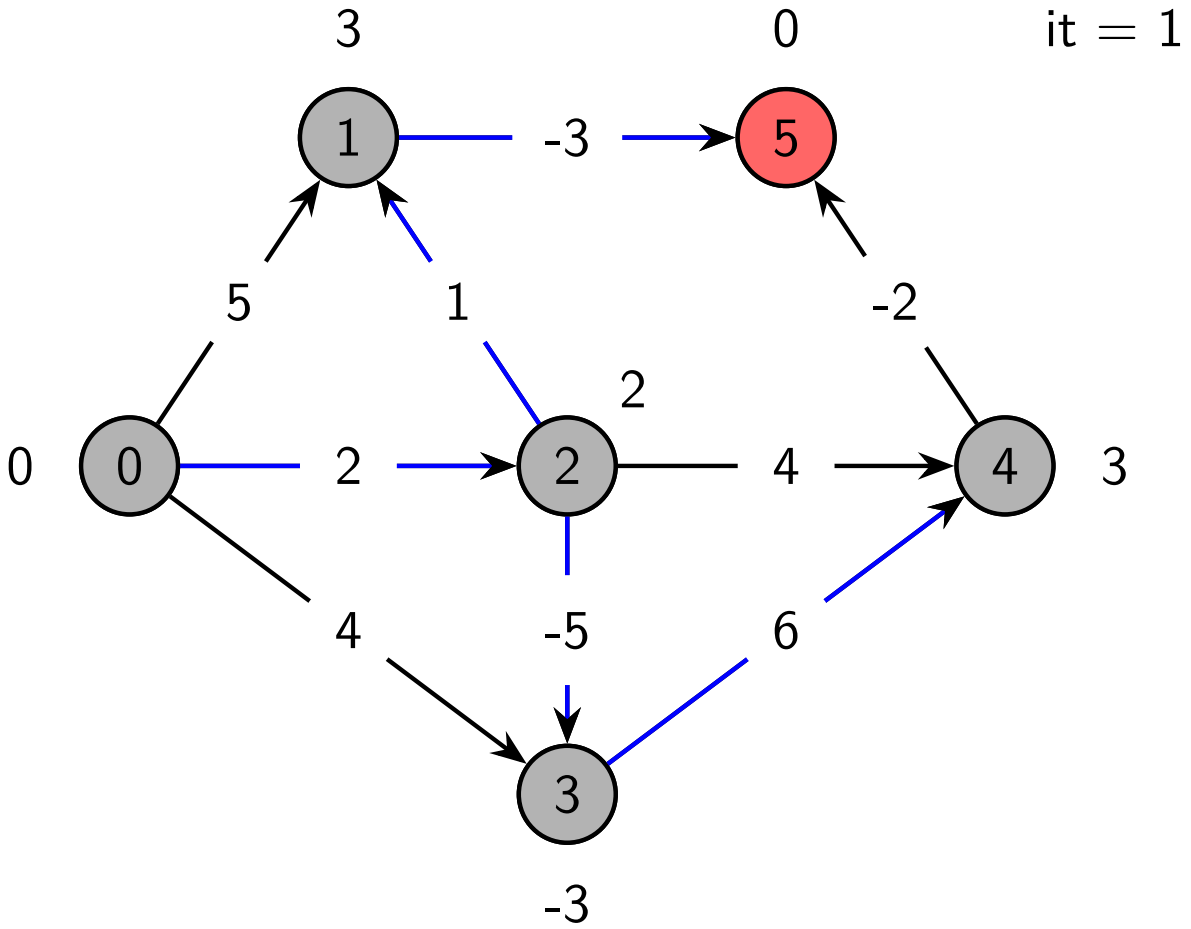
Algoritmo di Bellman-Ford

Algoritmo



Algoritmo di Bellman-Ford

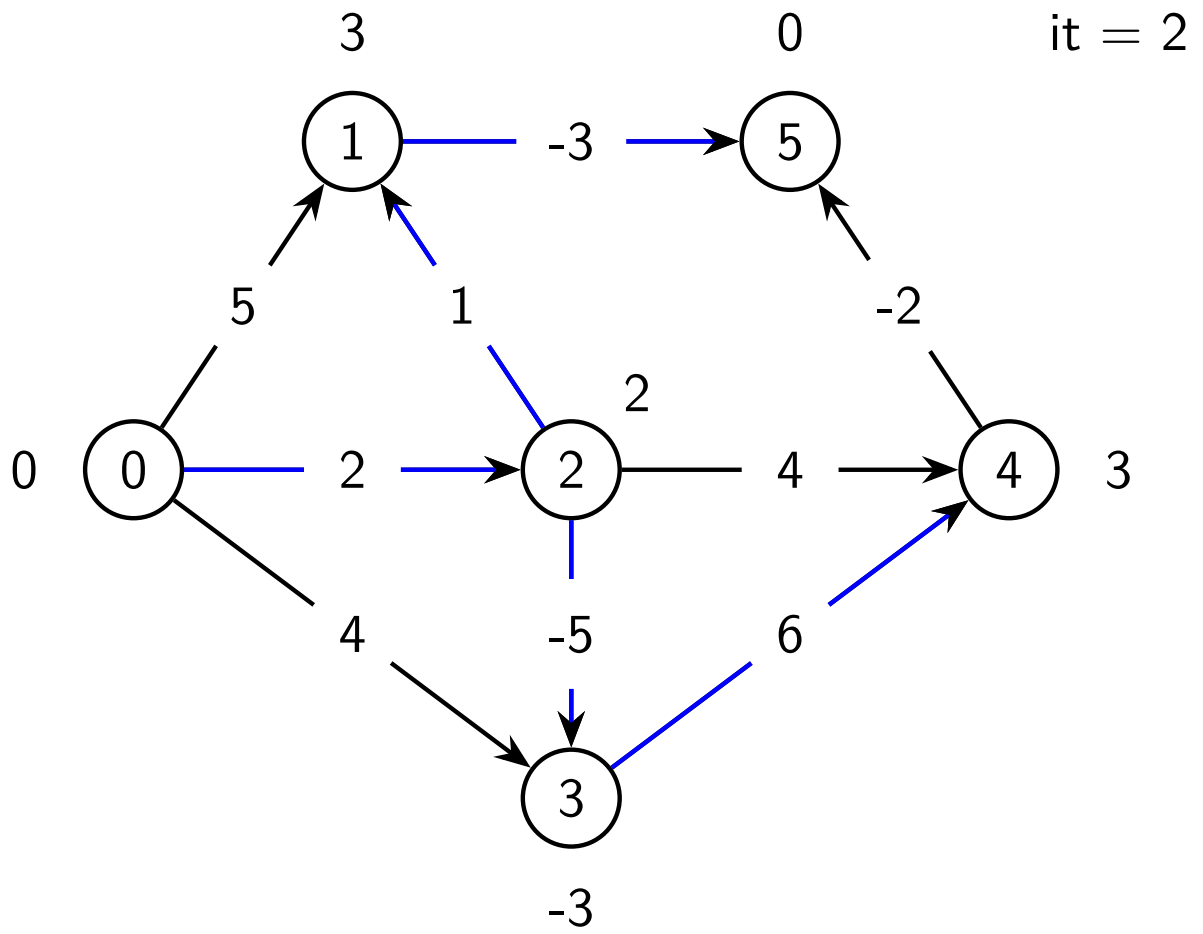
Algoritmo



Algoritmo di Bellman-Ford

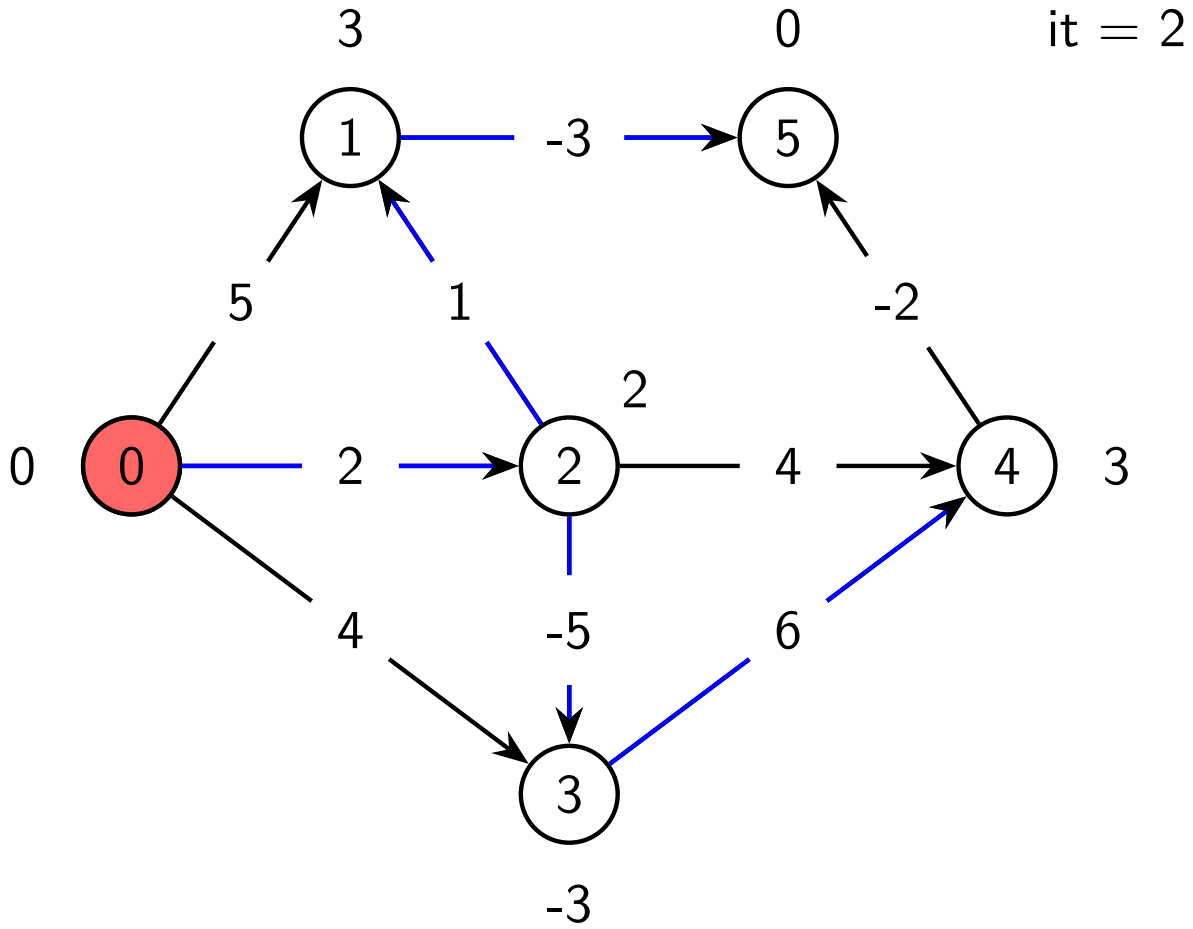
Algoritmo

Visto che ci sono stati dei cambiamenti è necessario ripetere la visita di tutti i nodi.



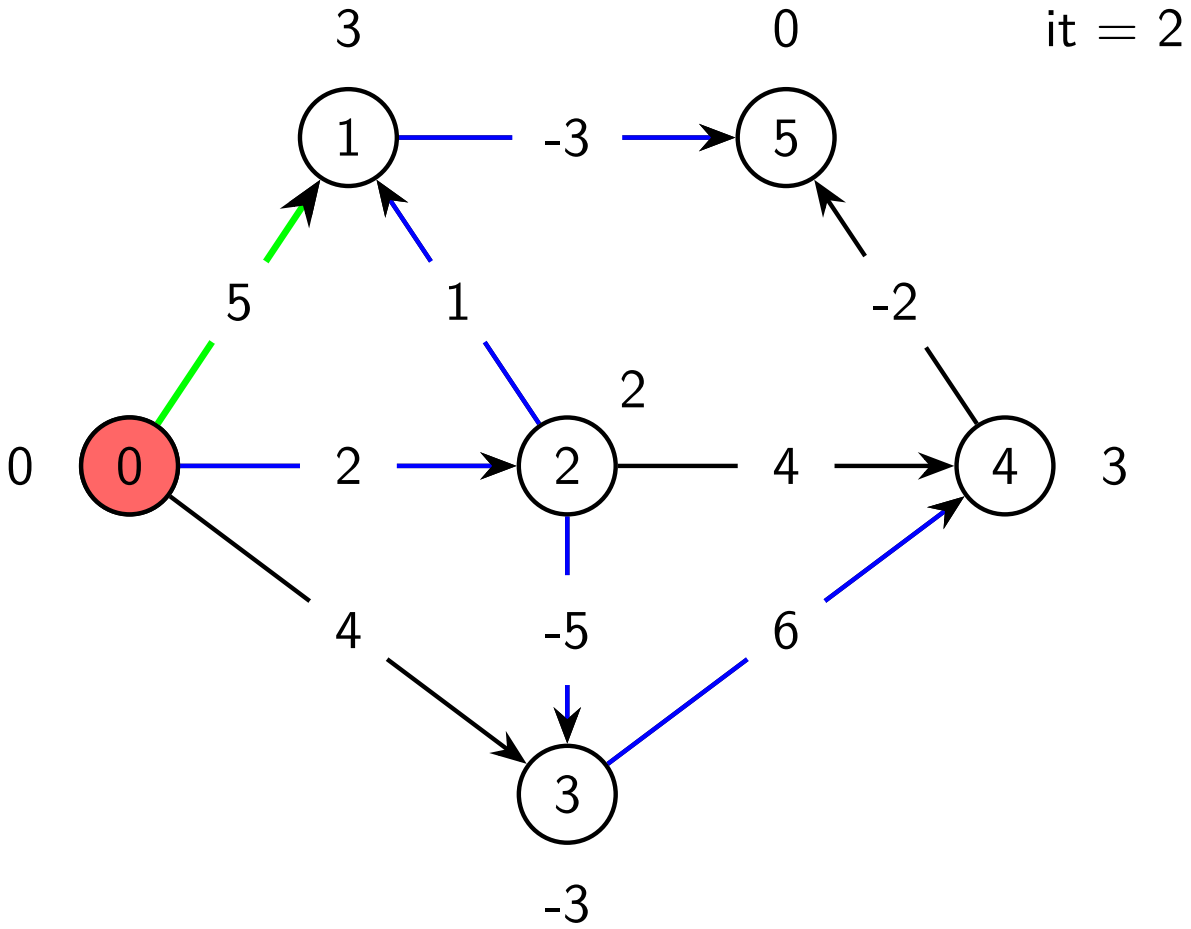
Algoritmo di Bellman-Ford

Algoritmo



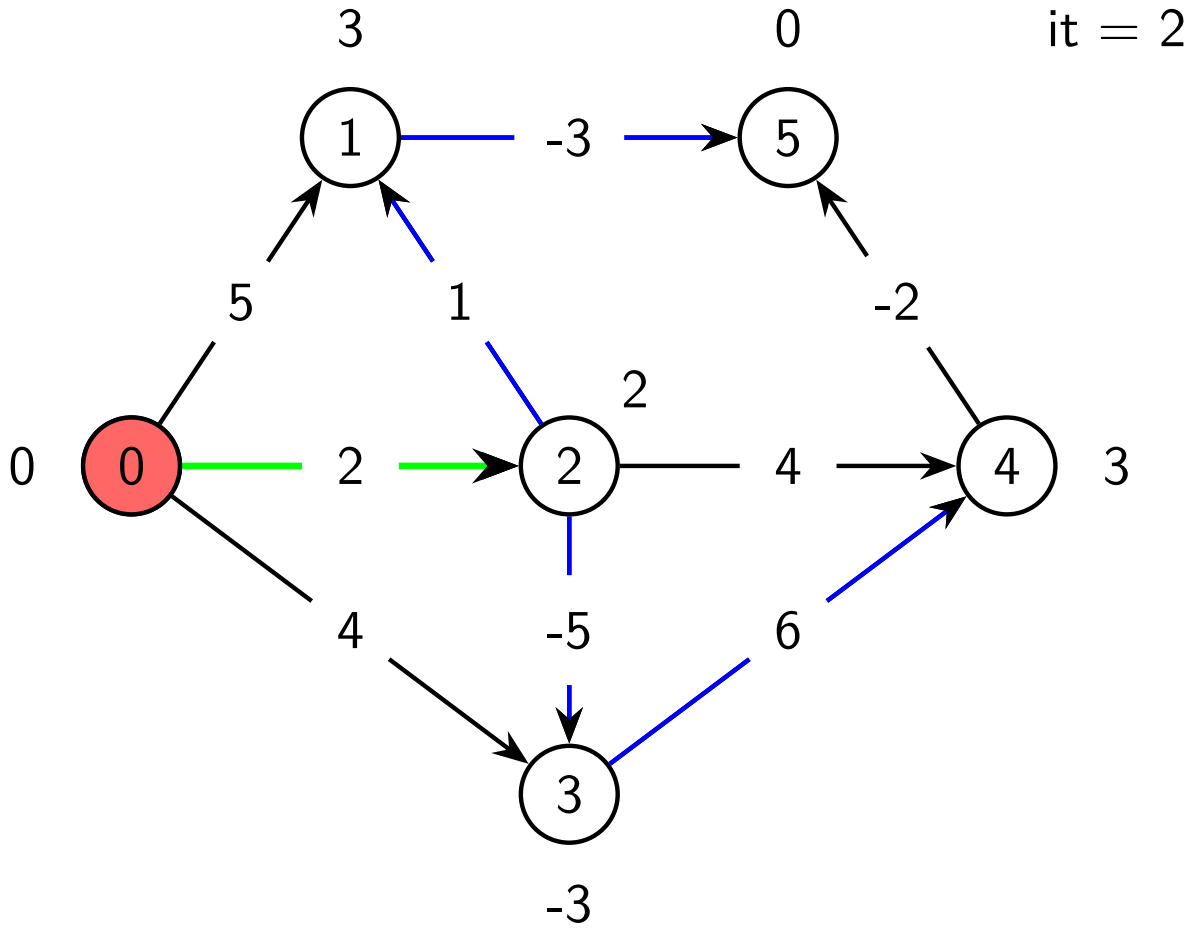
Algoritmo di Bellman-Ford

Algoritmo



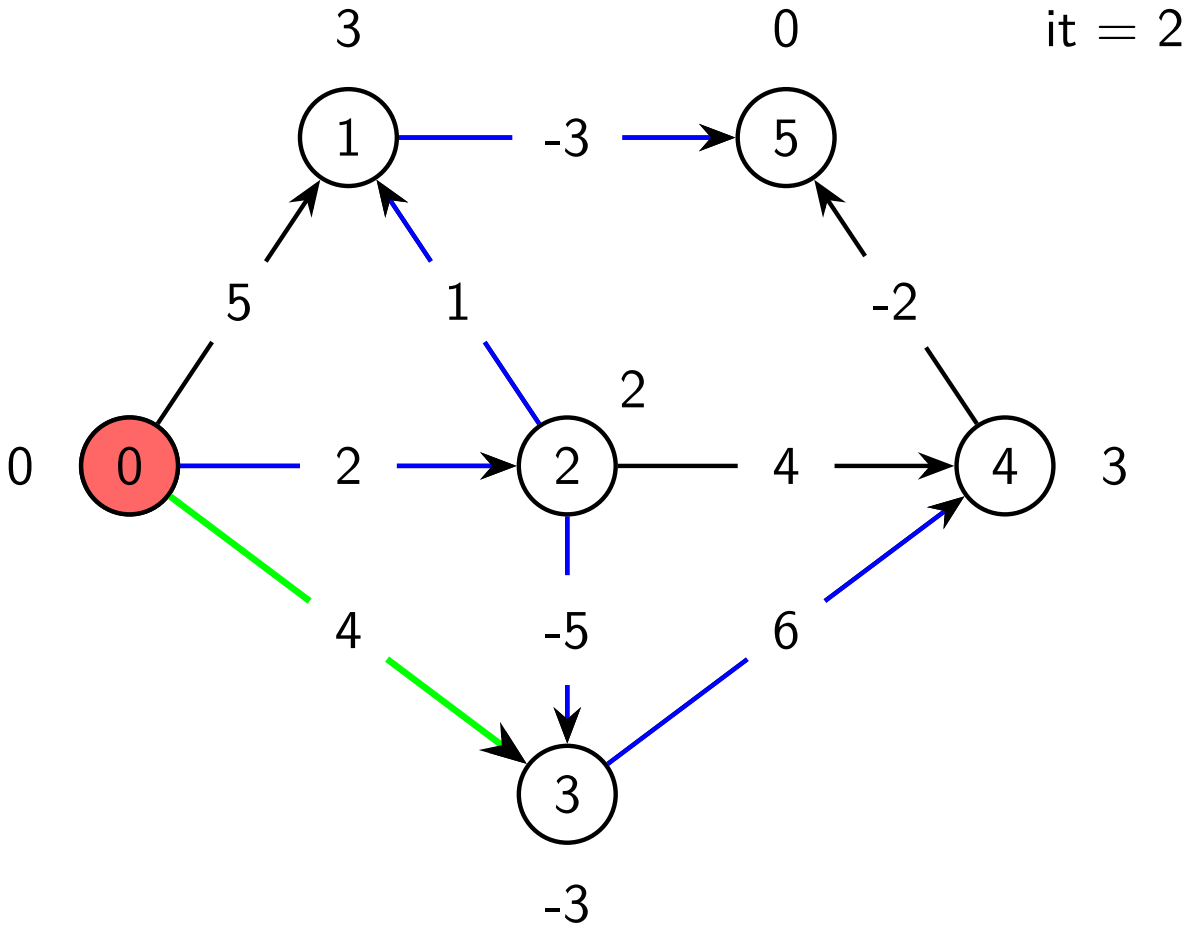
Algoritmo di Bellman-Ford

Algoritmo



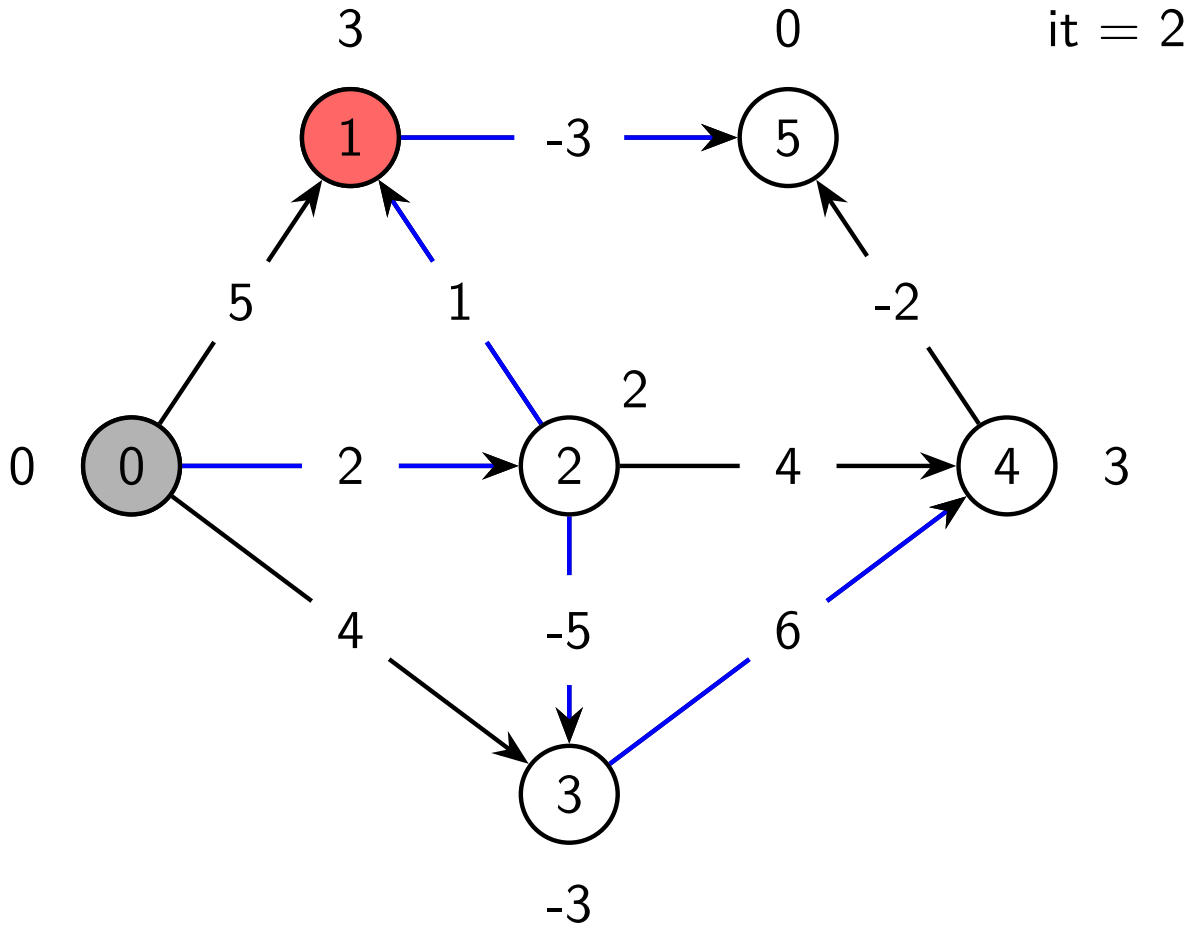
Algoritmo di Bellman-Ford

Algoritmo



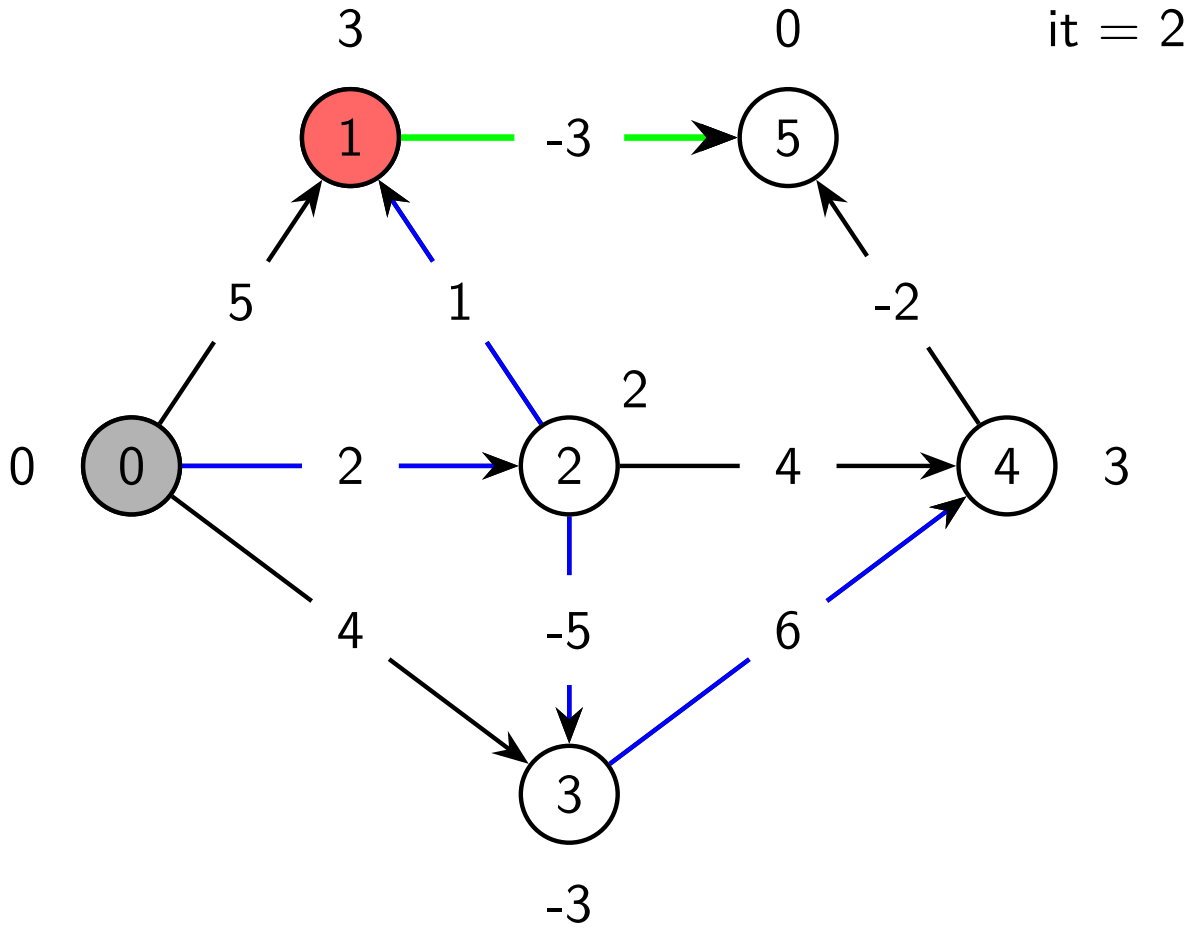
Algoritmo di Bellman-Ford

Algoritmo



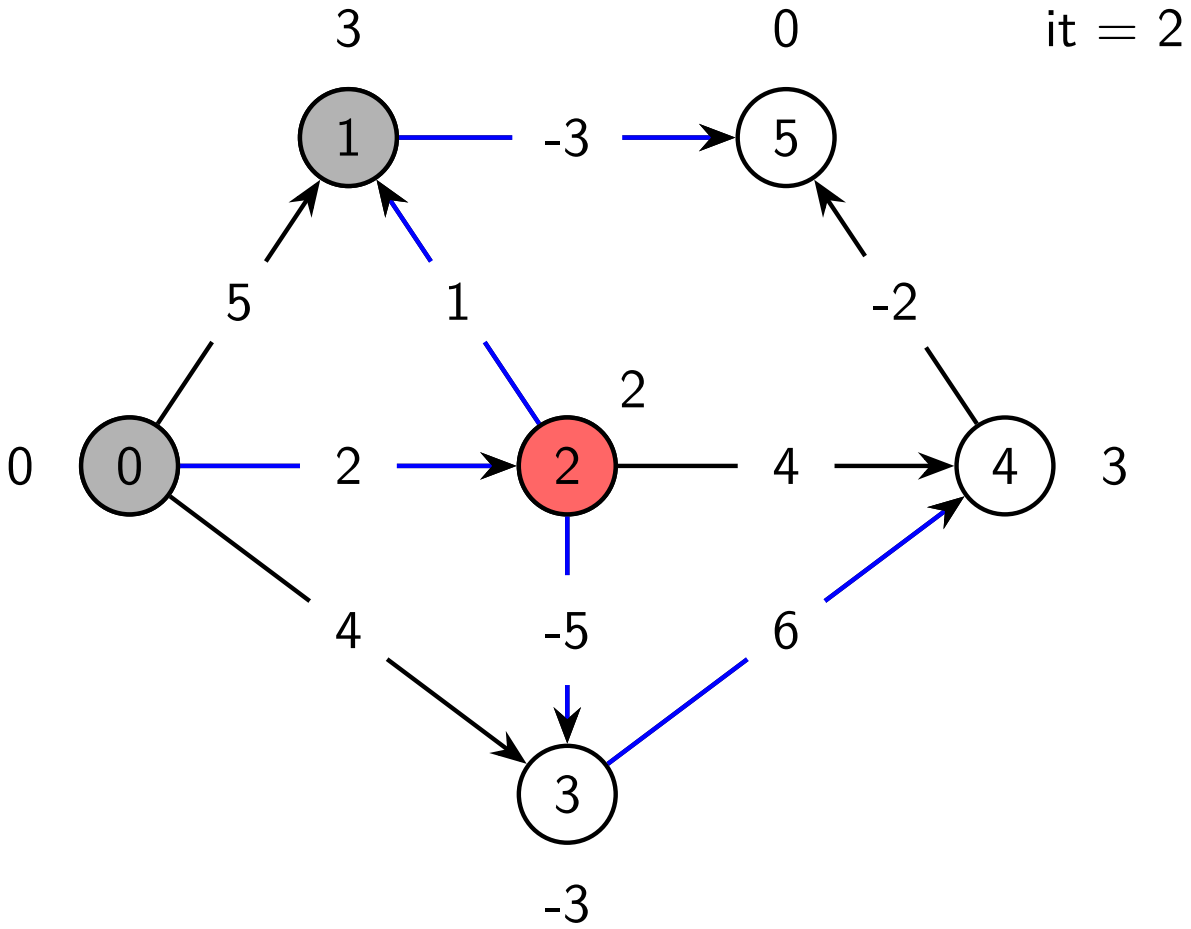
Algoritmo di Bellman-Ford

Algoritmo



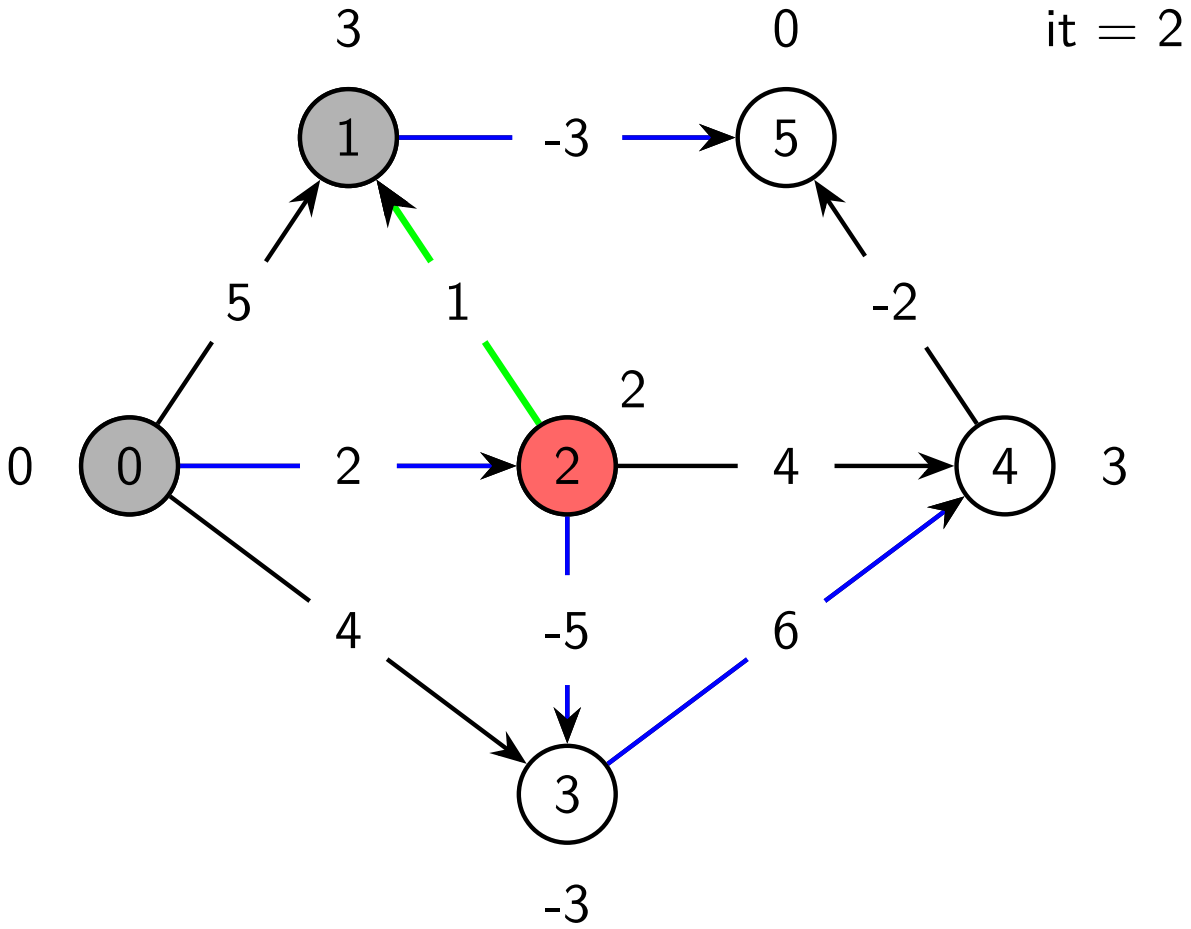
Algoritmo di Bellman-Ford

Algoritmo



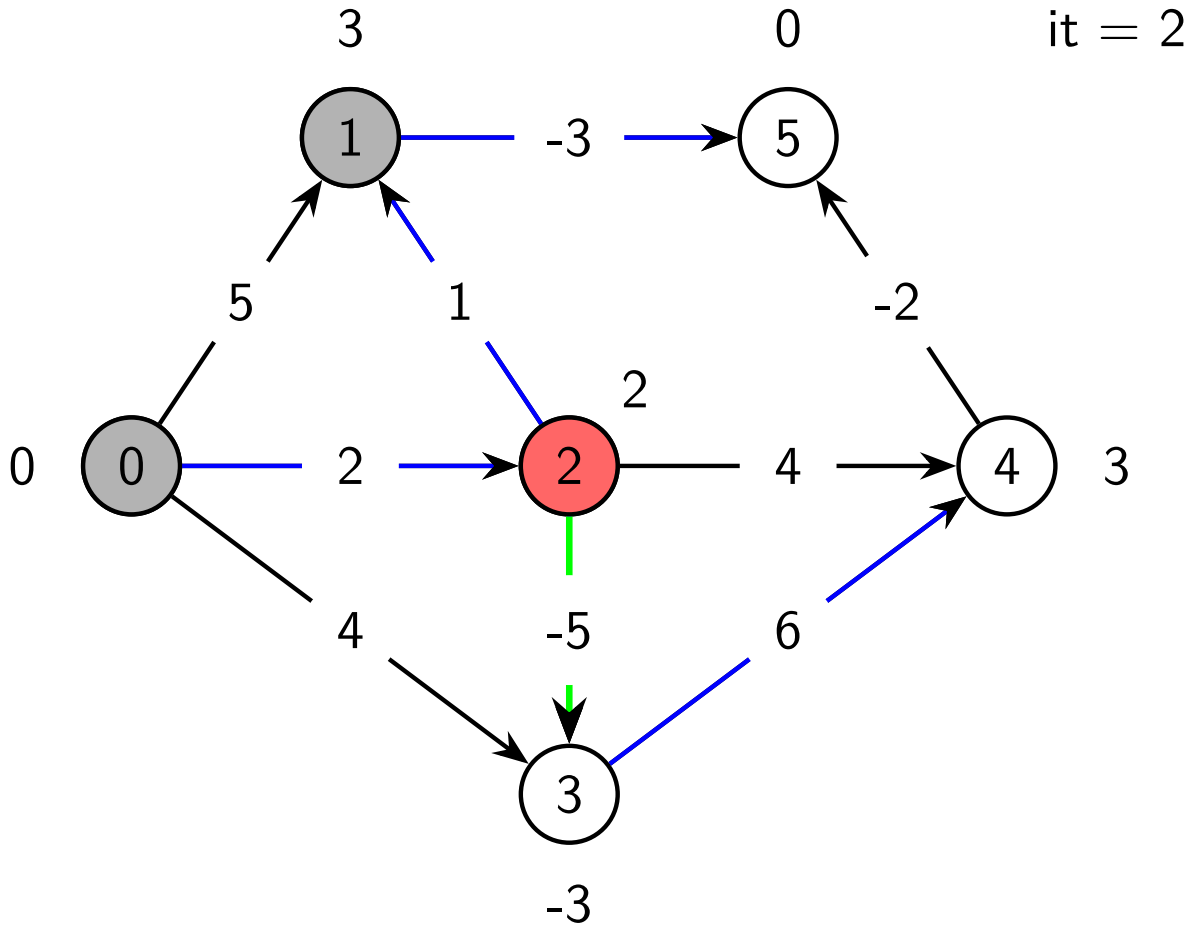
Algoritmo di Bellman-Ford

Algoritmo



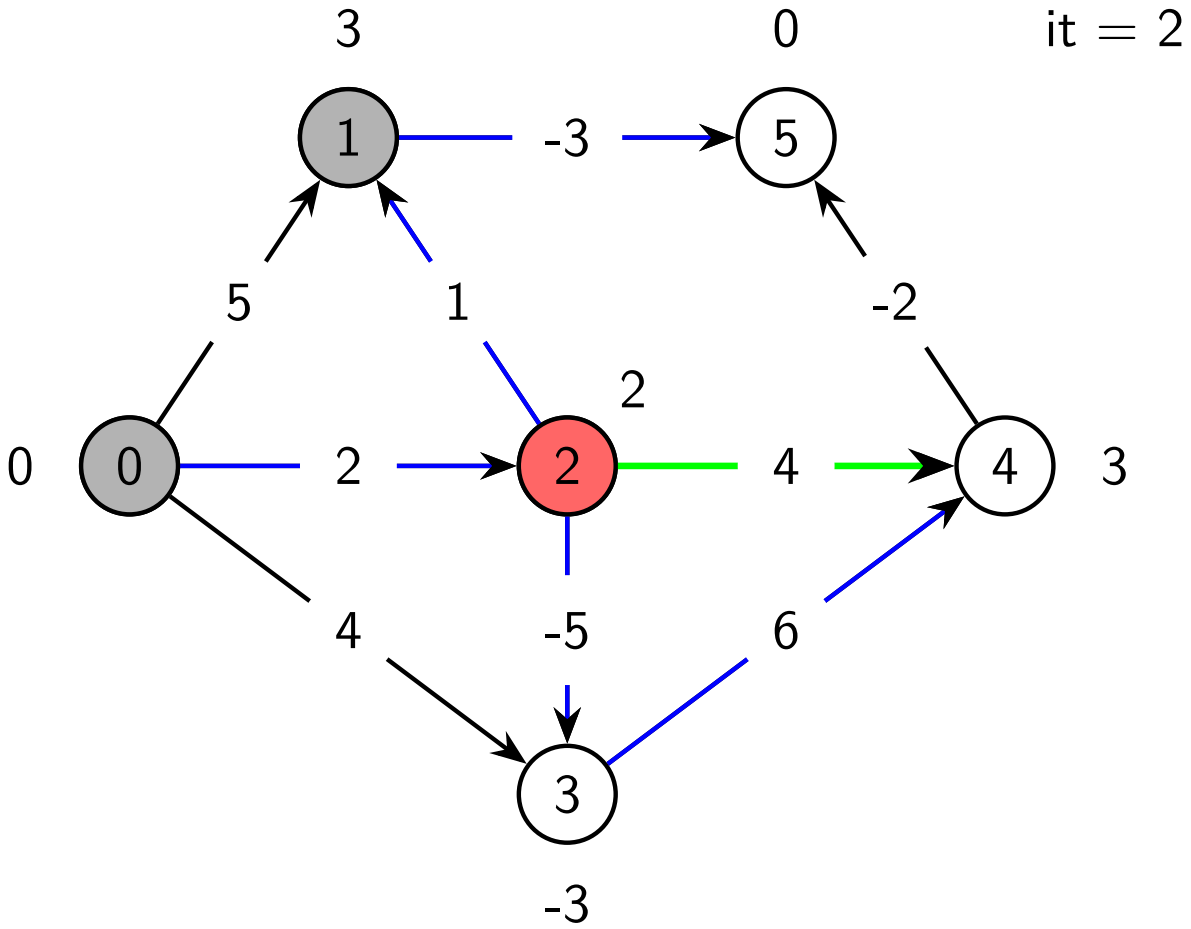
Algoritmo di Bellman-Ford

Algoritmo



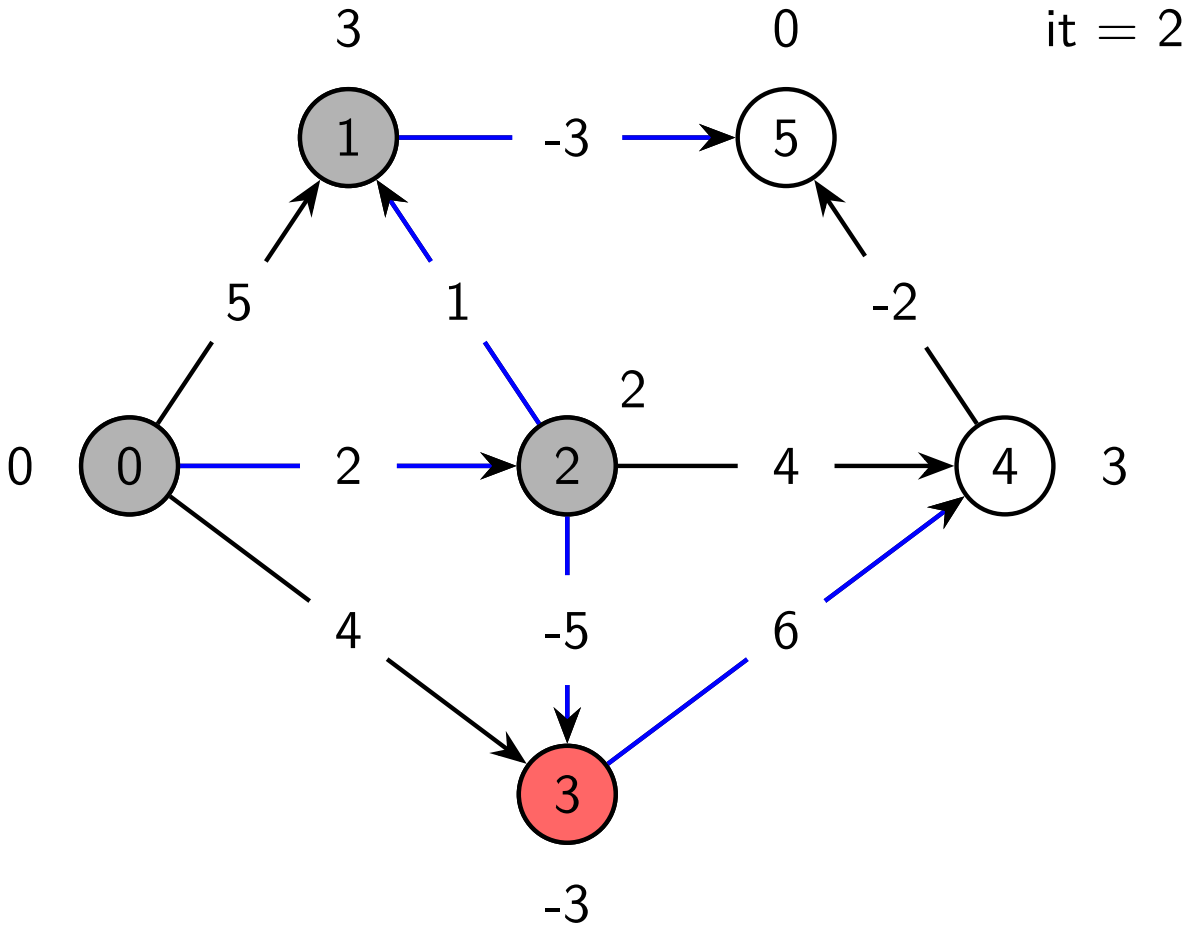
Algoritmo di Bellman-Ford

Algoritmo



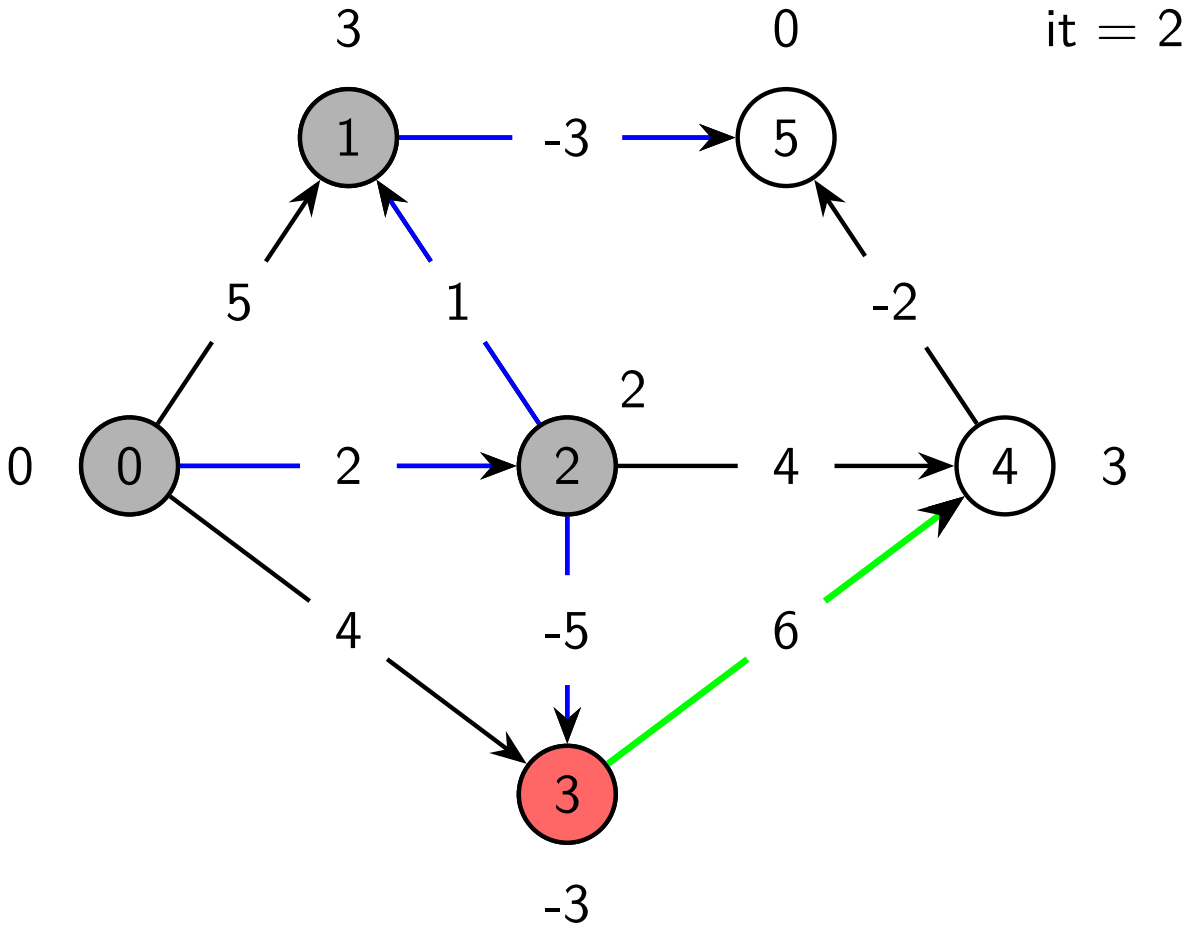
Algoritmo di Bellman-Ford

Algoritmo



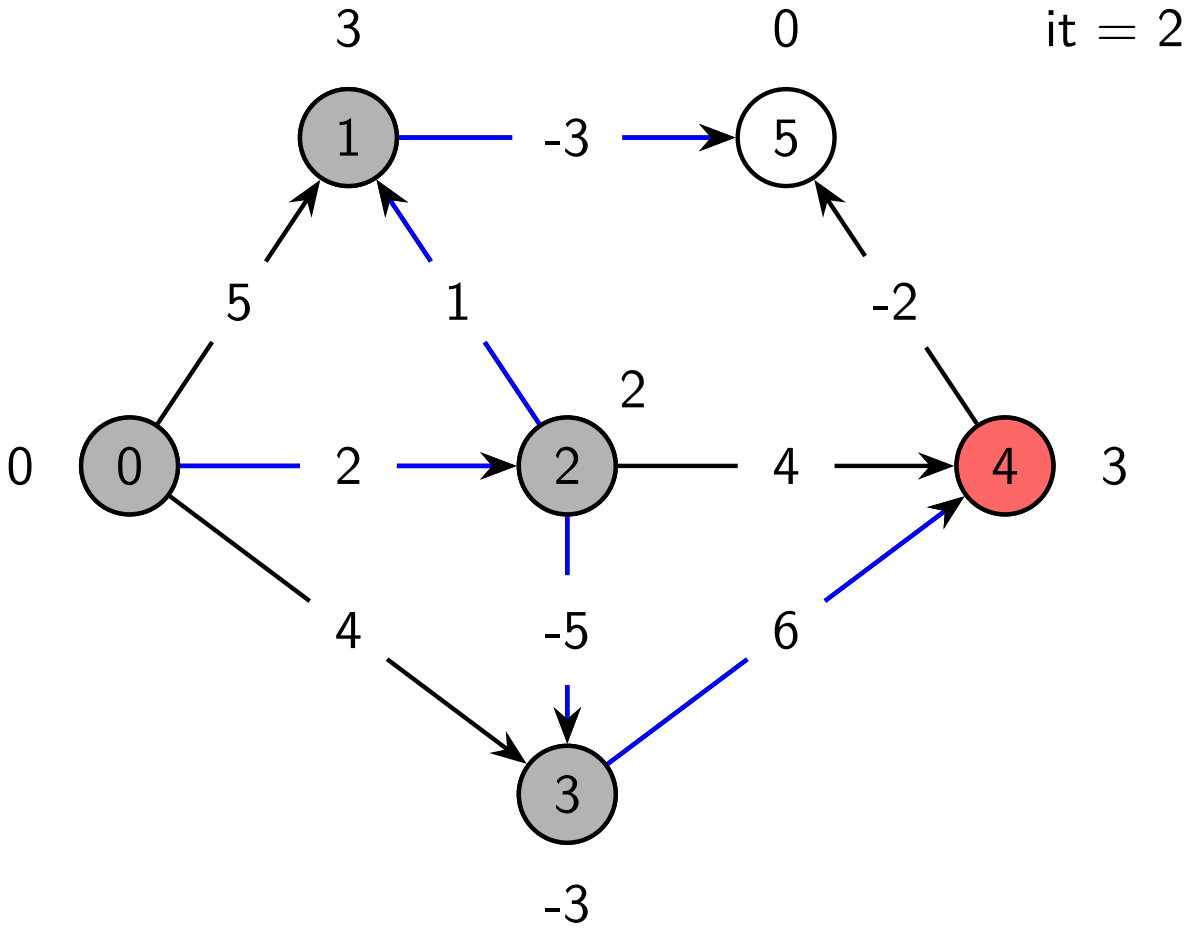
Algoritmo di Bellman-Ford

Algoritmo



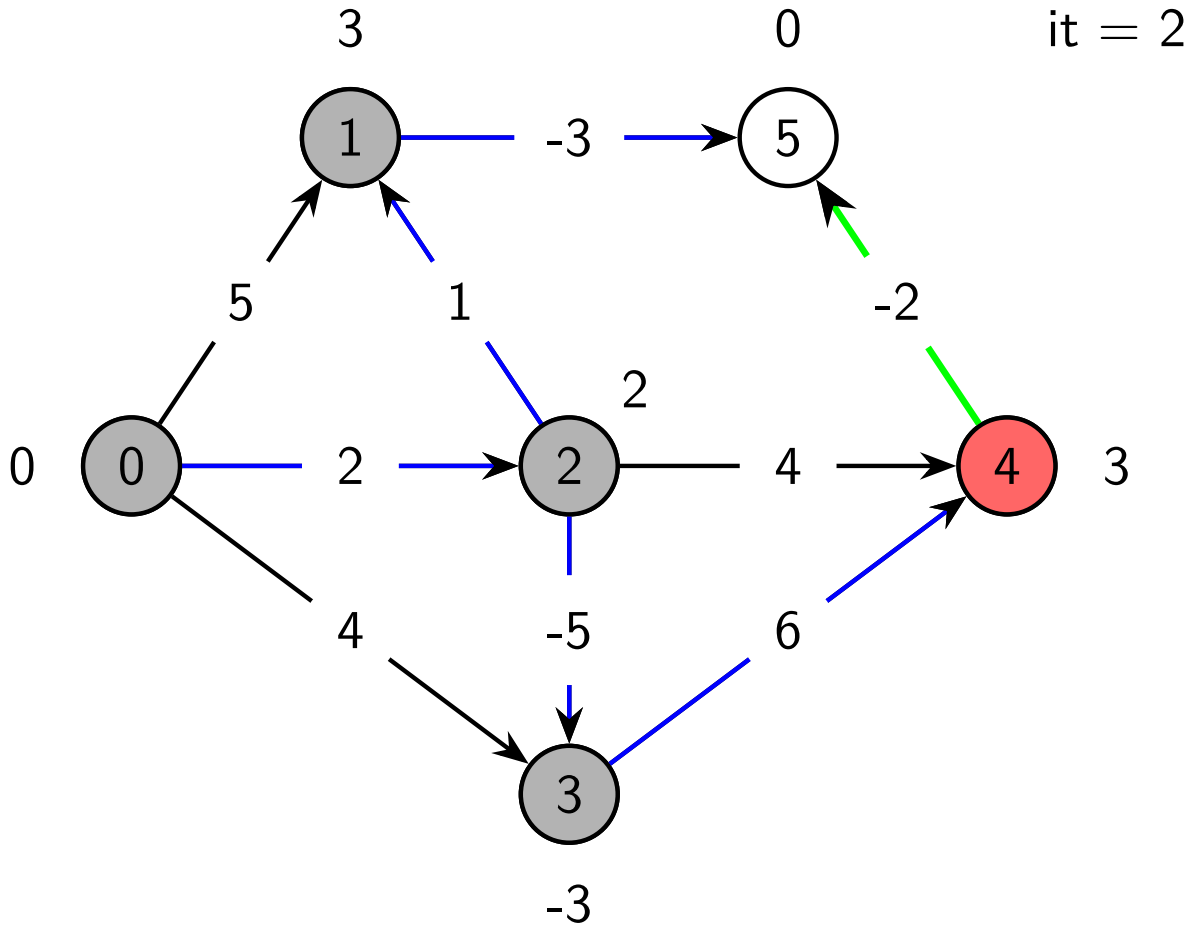
Algoritmo di Bellman-Ford

Algoritmo



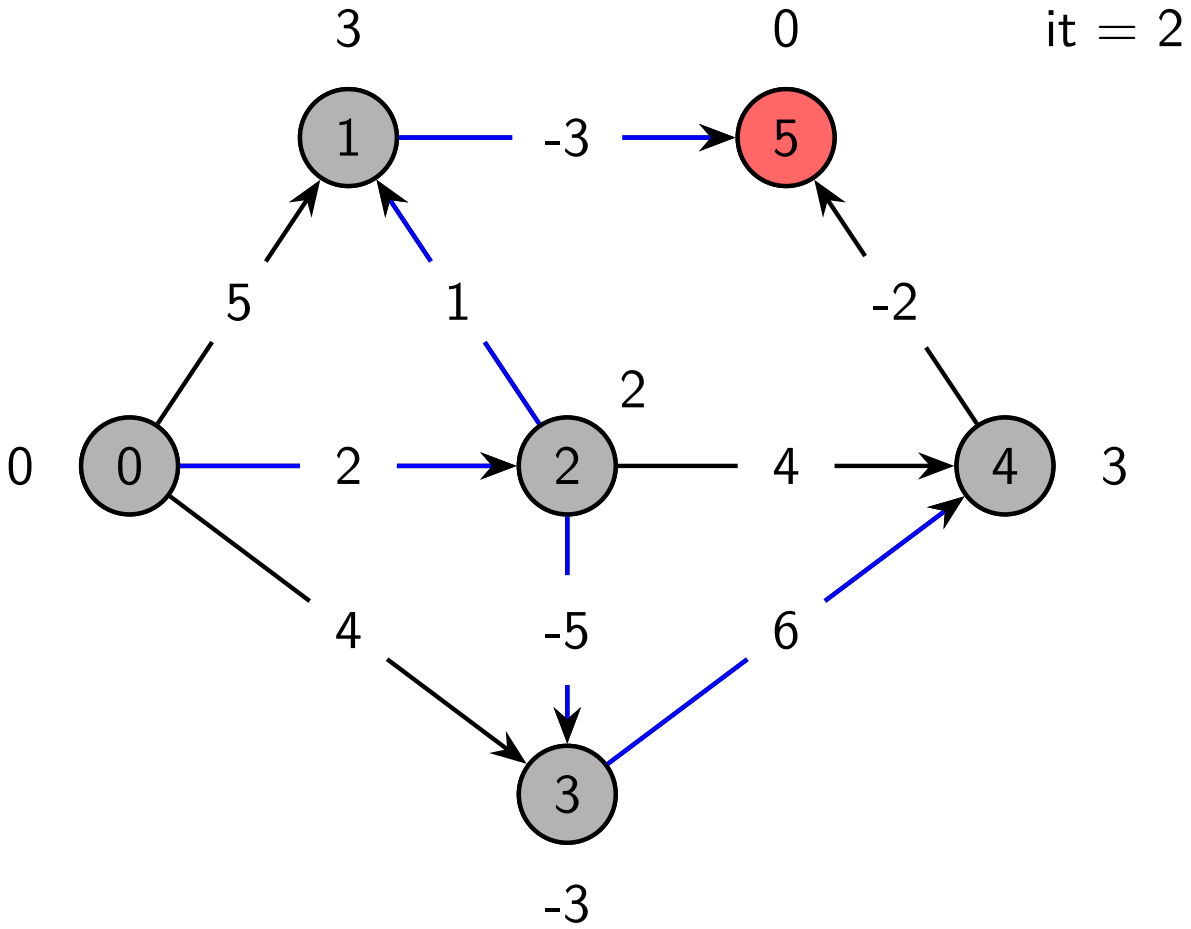
Algoritmo di Bellman-Ford

Algoritmo



Algoritmo di Bellman-Ford

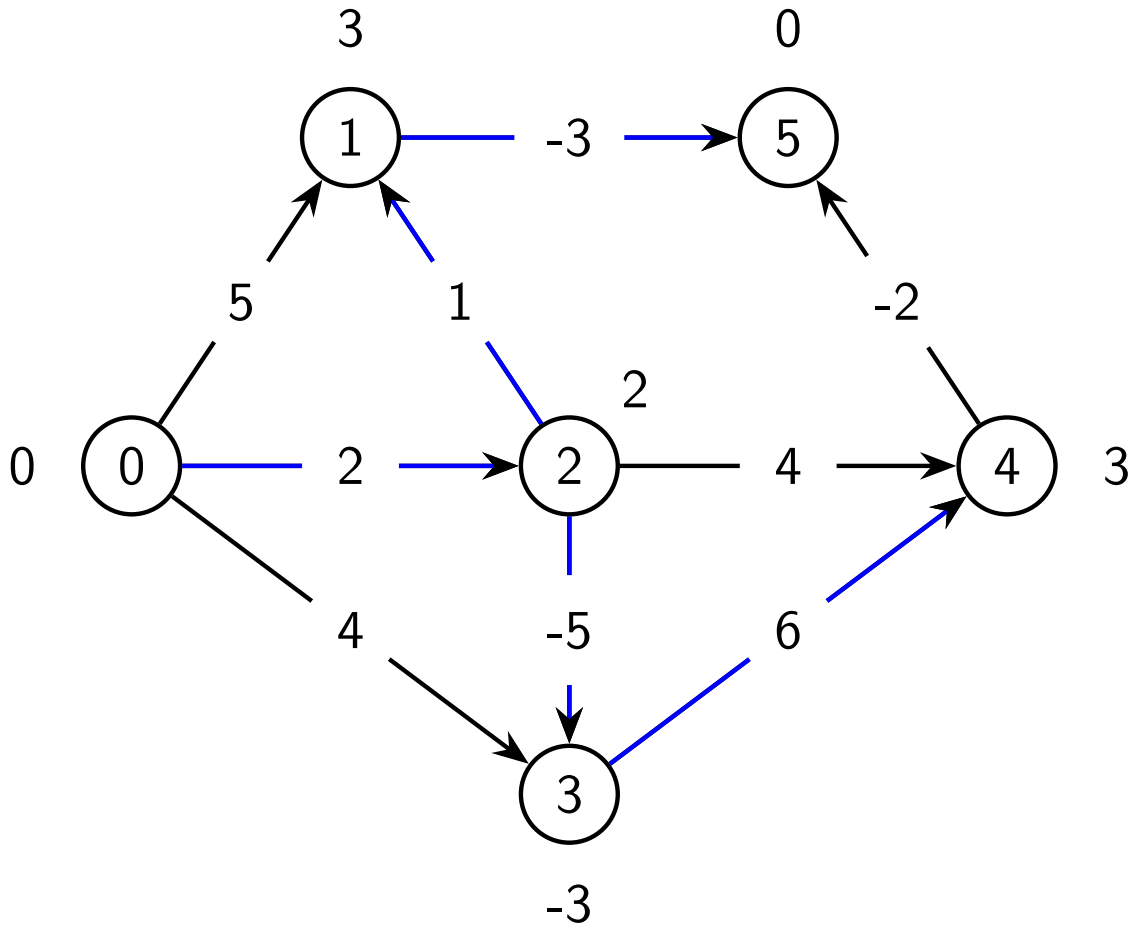
Algoritmo



Algoritmo di Bellman-Ford

Algoritmo

Durante questa iterazione non ci sono stati cambiamenti: l'algoritmo termina.



Algoritmo di Bellman-Ford

Complessità

L'algoritmo di Bellman-Ford termina quando in un'iterazione non ci sono stati aggiornamenti alle distanze.

Algoritmo di Bellman-Ford

Complessità

L'algoritmo di Bellman-Ford termina quando in un'iterazione non ci sono stati aggiornamenti alle distanze.
Se non ci sono cicli di lunghezza negativa questo succede sicuramente prima dell'N-esima iterazione, se cioè non accade vuol dire che siamo in presenza di un ciclo di lunghezza negativa e ci possiamo fermare perché ci saranno vertici a distanza $-\infty$.

Algoritmo di Bellman-Ford

Complessità

L'algoritmo di Bellman-Ford termina quando in un'iterazione non ci sono stati aggiornamenti alle distanze.
Se non ci sono cicli di lunghezza negativa questo succede sicuramente prima dell'N-esima iterazione, se cioè non accade vuol dire che siamo in presenza di un ciclo di lunghezza negativa e ci possiamo fermare perché ci saranno vertici a distanza $-\infty$.
La complessità è dunque $\mathcal{O}(NM)$, poiché ogni iterazione richiede tempo $\mathcal{O}(M)$.

Algoritmo di Floyd-Warshall

L'algoritmo di Floyd-Warshall risolve un problema più generale dei precedenti, cioè trovare le distanze fra *tutte* le coppie di nodi in un grafo senza cicli di lunghezza negativa.

Algoritmo di Floyd-Warshall

L'algoritmo di Floyd-Warshall risolve un problema più generale dei precedenti, cioè trovare le distanze fra *tutte* le coppie di nodi in un grafo senza cicli di lunghezza negativa.

Operazione triangolare
Per ogni nodo h, e per ogni coppia di nodi i, j controlla se il percorso $i \to h \to j$ conviene rispetto al percorso $i \to j$.

Per ricostruire la soluzione possiamo ricordarci, per ogni coppia di nodi, quale arco prendere.

Algoritmo di Floyd-Warshall

$D_{ij} = \infty, N_{ij} = -1 \qquad \forall i, j$

foreach *edge* (i, j) **do**

$D_{ij} = w_{ij}$

$N_{ij} = j$

end

foreach *node* i **do**

$D_{ii} = 0$

end

Algoritmo di Floyd-Warshall

$D_{ij} = \infty, N_{ij} = -1 \qquad \forall i, j$

foreach *edge* (i, j) **do**

$D_{ij} = w_{ij}$

$N_{ij} = j$

end

foreach *node* i **do**

$D_{ii} = 0$

end

for $h \leftarrow 0$ **to** $N - 1$ **do**

end

Algoritmo di Floyd-Warshall

$D_{ij} = \infty, N_{ij} = -1 \qquad \forall i, j$

foreach *edge* (i, j) **do**

$D_{ij} = w_{ij}$

$N_{ij} = j$

end

foreach *node* i **do**

$D_{ii} = 0$

end

for $h \leftarrow 0$ **to** $N - 1$ **do**

for $i \leftarrow 0$ **to** $N - 1$ **do**

for $j \leftarrow 0$ **to** $N - 1$ **do**

end

end

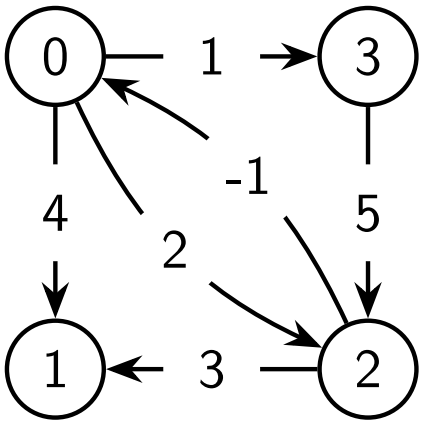
end

Algoritmo di Floyd-Warshall

$D_{ij} = \infty, N_{ij} = -1 \qquad \forall i, j$
foreach *edge* (i, j) **do**
  $D_{ij} = w_{ij}$
  $N_{ij} = j$
end
foreach *node* i **do**
  $D_{ii} = 0$
end
for $h \leftarrow 0$ **to** $N - 1$ **do**
  for $i \leftarrow 0$ **to** $N - 1$ **do**
    for $j \leftarrow 0$ **to** $N - 1$ **do**
      if $D_{ij} > D_{ih} + D_{hj}$ **then**
        $D_{ij} = D_{ih} + D_{hj}$
        $N_{ij} = N_{ih}$
      end
    end
  end
end

Algoritmo di Floyd-Warshall

Algoritmo



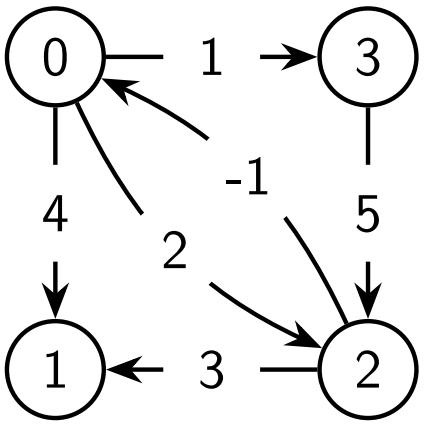
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



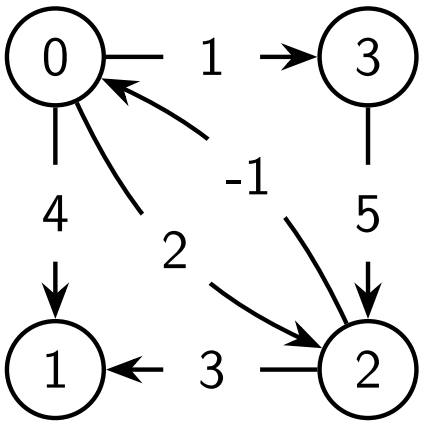
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



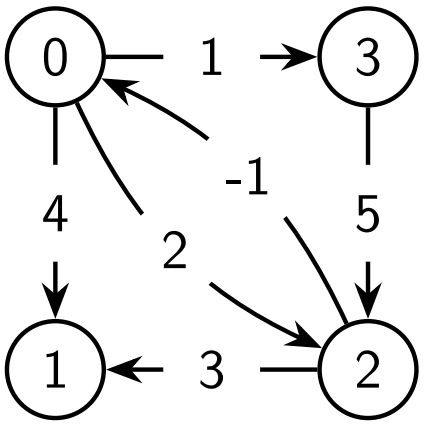
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



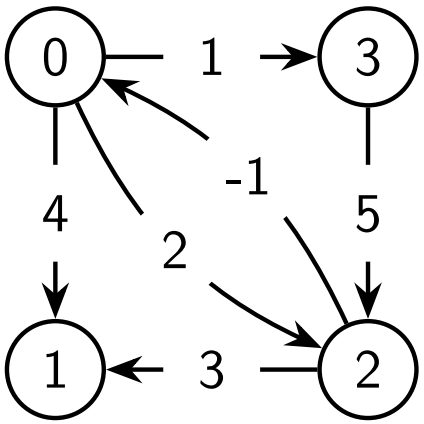
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



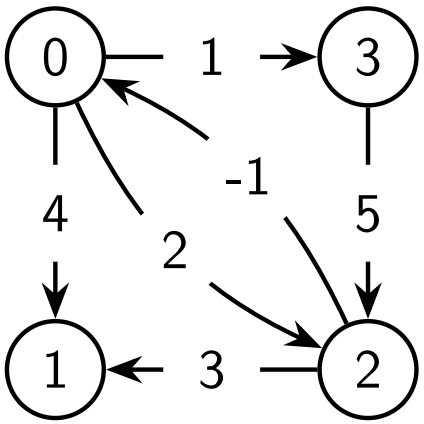
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



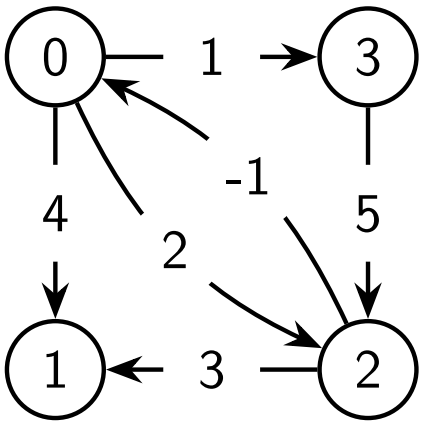
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



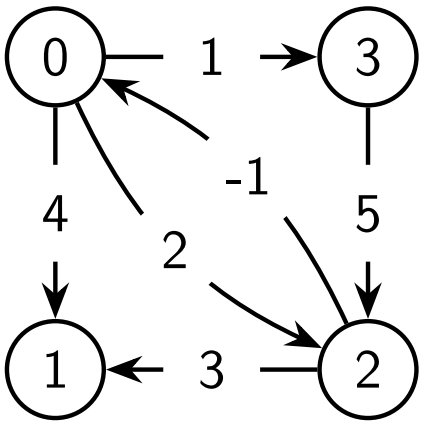
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	∞
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	-1
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



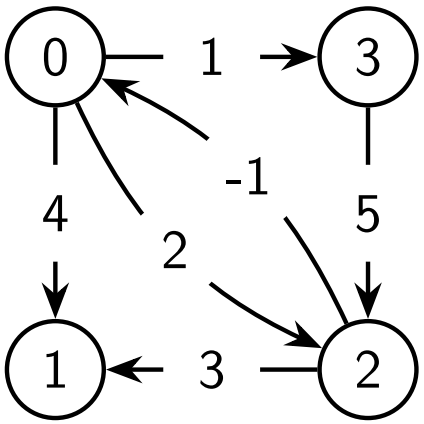
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



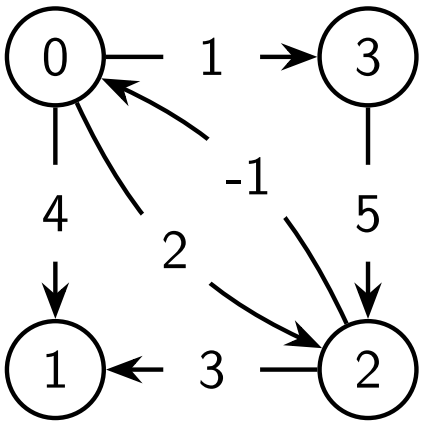
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



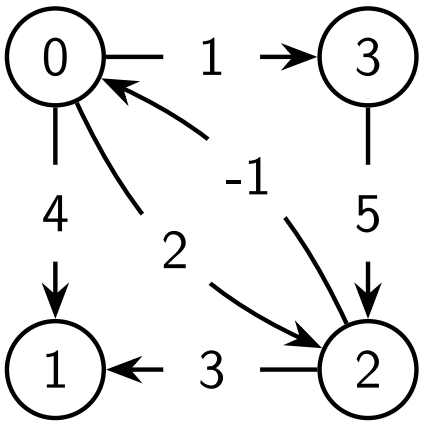
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 0$



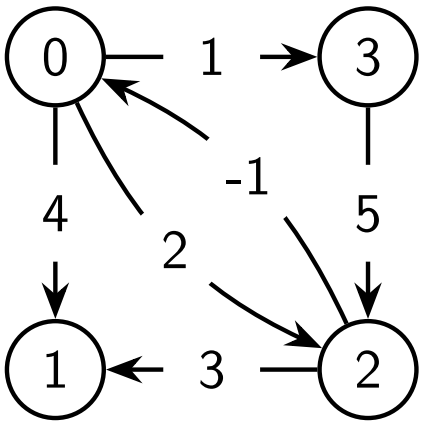
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



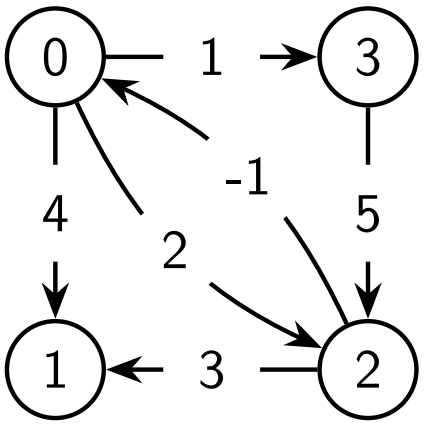
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



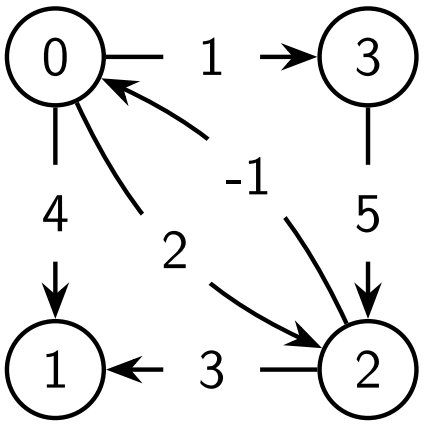
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



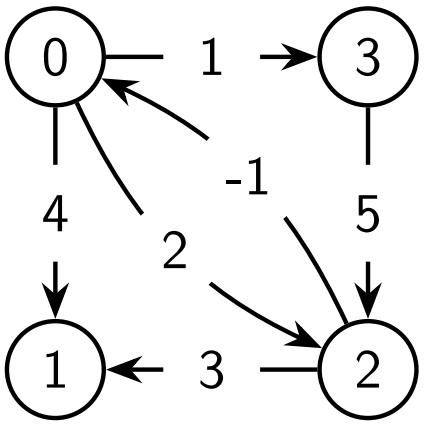
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



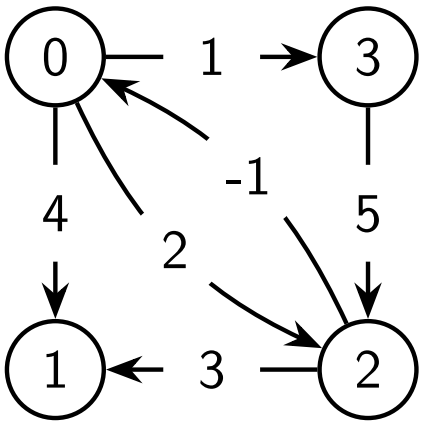
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



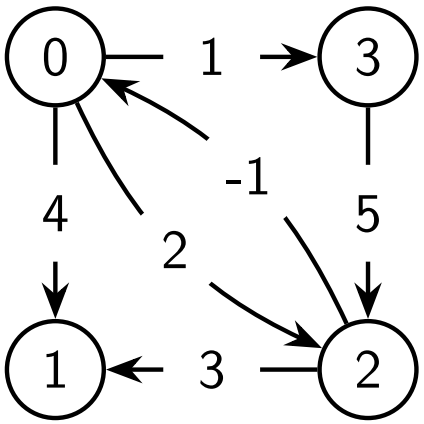
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



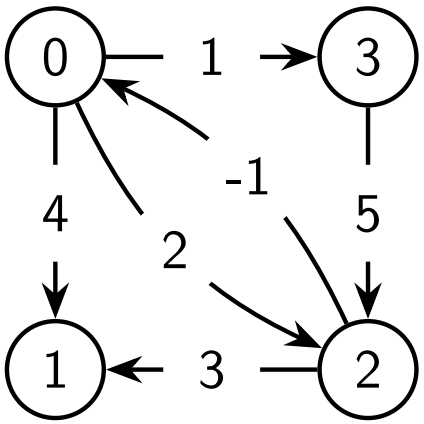
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



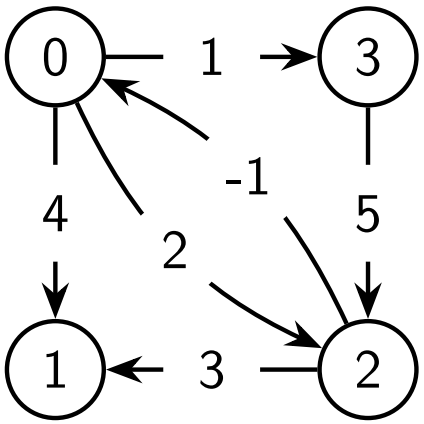
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



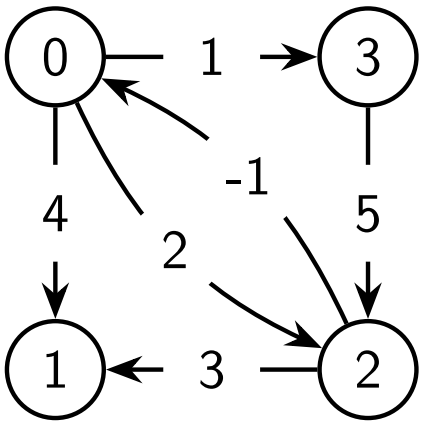
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 1$



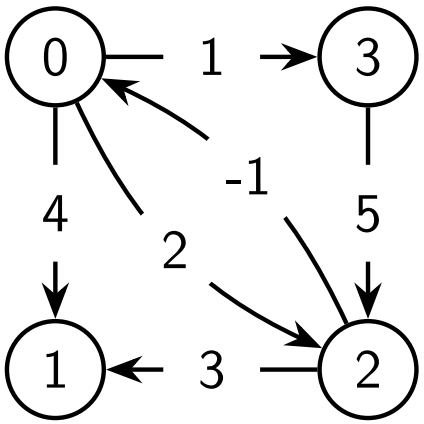
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



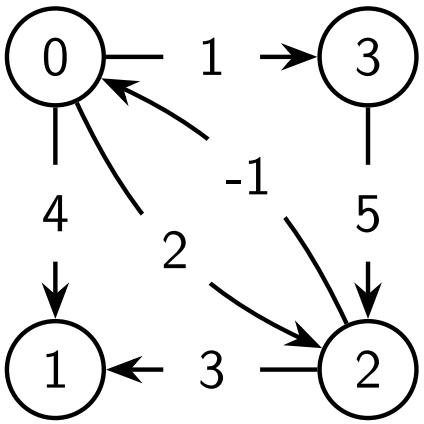
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



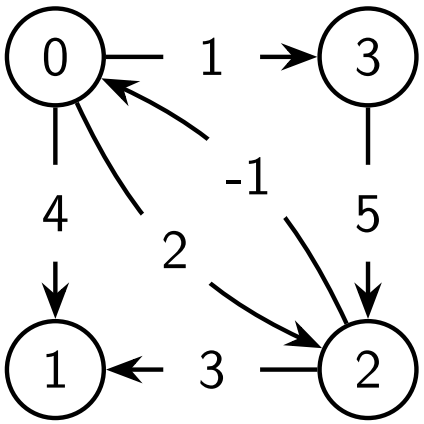
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



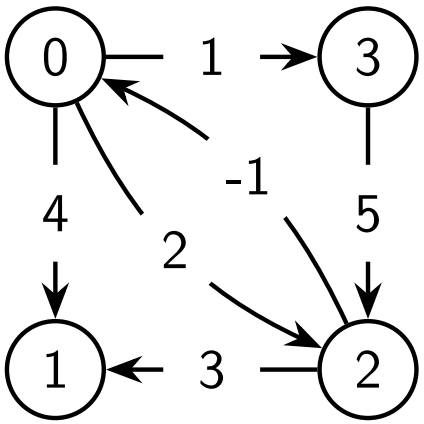
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



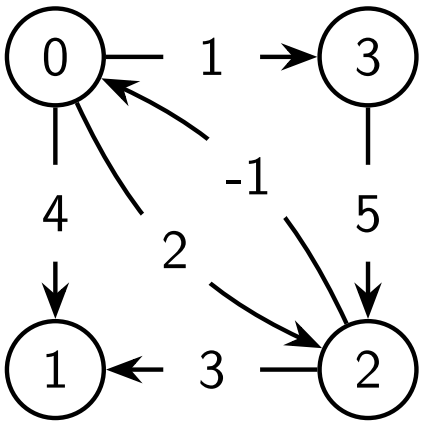
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



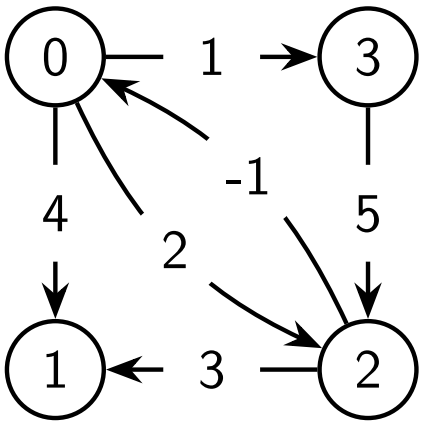
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



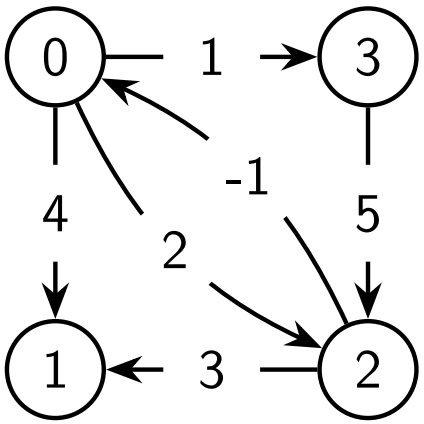
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



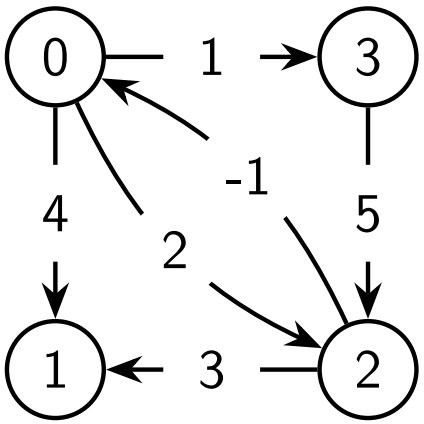
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	∞	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	-1	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



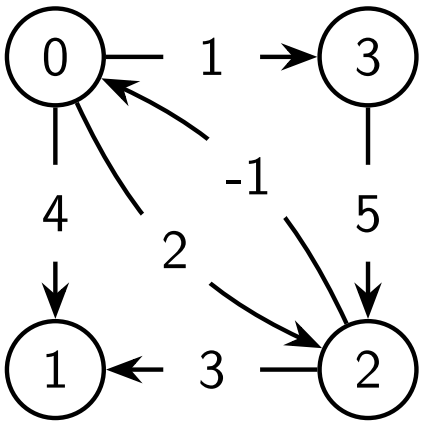
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



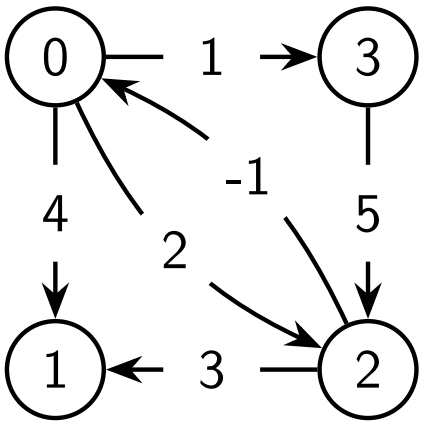
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	∞	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	-1	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



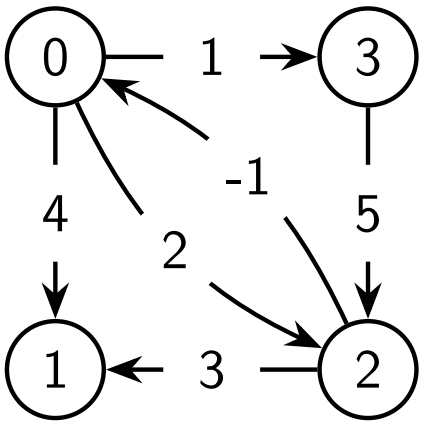
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 2$



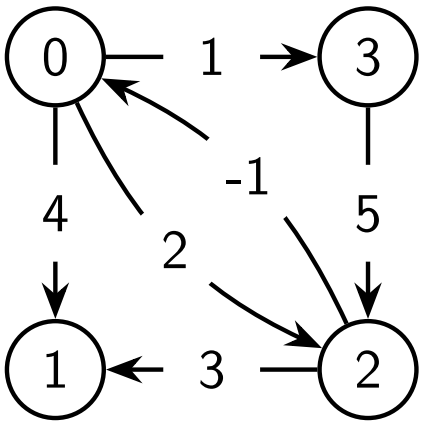
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



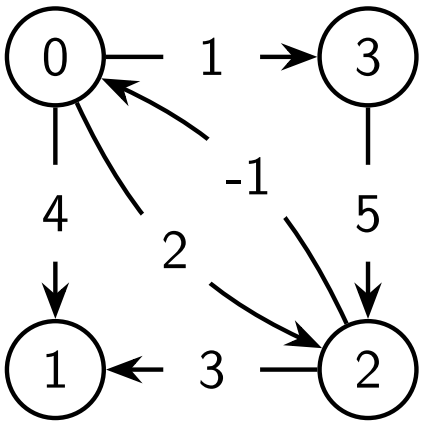
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



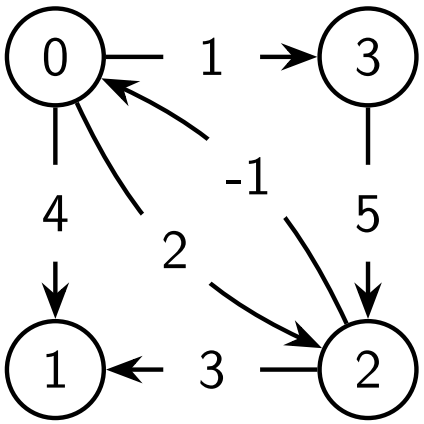
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



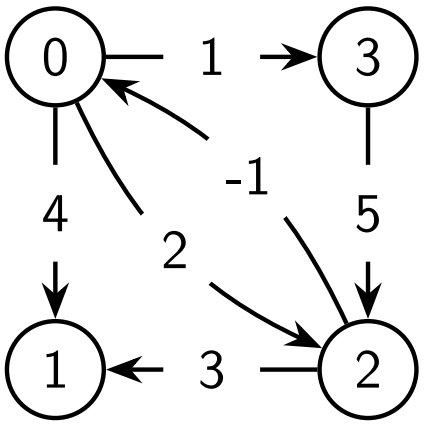
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



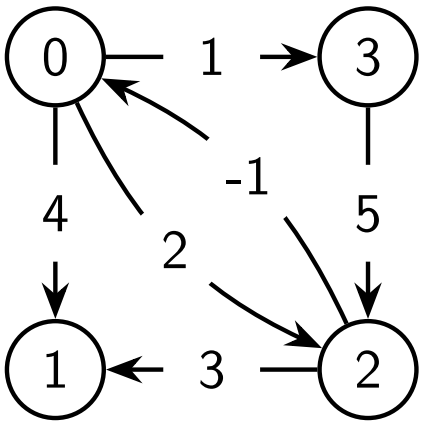
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



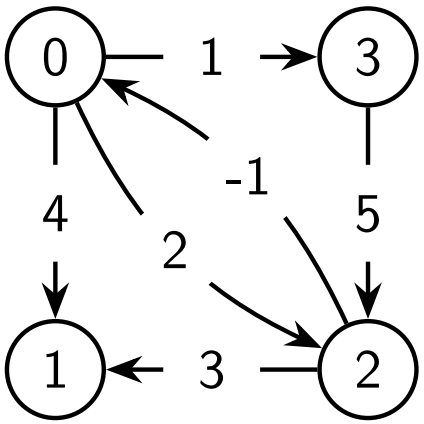
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



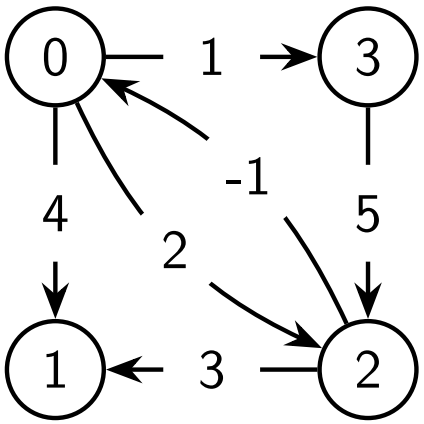
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



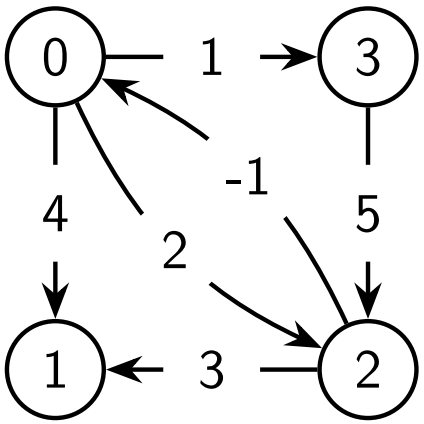
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$



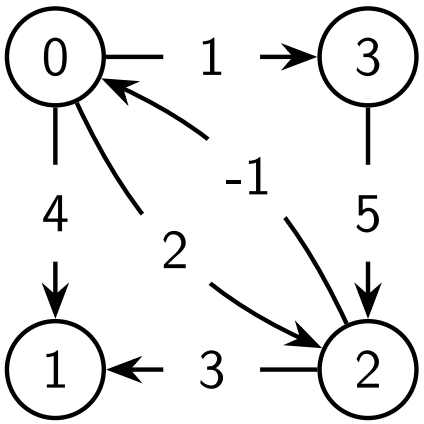
D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo

$h = 3$

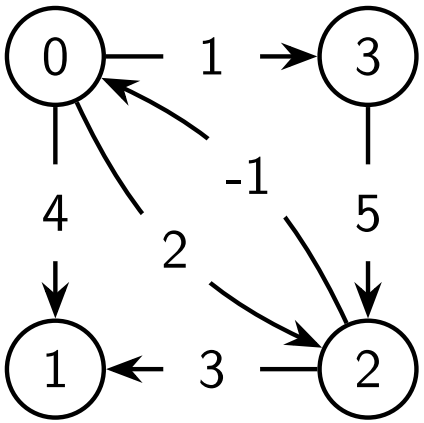


D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Algoritmo



D	0	1	2	3
0	0	4	2	1
1	∞	0	∞	∞
2	-1	3	0	0
3	4	8	5	0

N	0	1	2	3
0	0	1	2	3
1	-1	1	-1	-1
2	0	1	2	0
3	2	2	2	3

Algoritmo di Floyd-Warshall

Complessità

Per sua stessa costruzione, l'algoritmo di Floyd-Warshall ha complessità $\mathcal{O}(N^3)$.

Osservazione

Dijkstra usando una priority queue ha complessità $\mathcal{O}(N \cdot M \log M)$, quindi Floyd-Warshall è più veloce su grafi densi ($M = \mathcal{O}(N^2)$).

Esercizio

- *Algoritmo di Dijkstra* *cses – Shortest Routes I*
- *Algoritmo di Bellman-Ford* *cses – High Score*
- *Algoritmo di Floyd-Warshall* *cses – Shortest Routes II*