UNIVERSITÀ DELLA CALABRIA
DIPARTIMENTO DI MATEMATICA E INFORMATICA

Algoritmo di Dijkstra

Esercitazione del 07/05/2020

Francesco Cauteruccio, Ph.D.
cauteruccio@mat.unical.it
francescocauteruccio.info

Inside the Vault EDITION

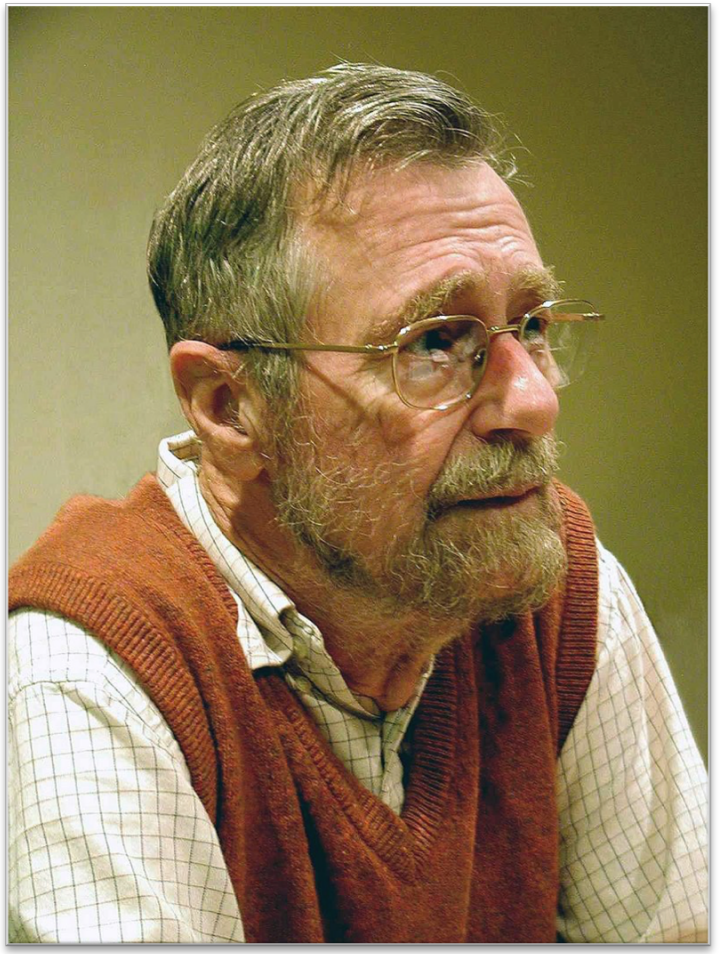
Problema dei cammini minimi

- **Shortest-paths problem**
  - Dato un grafo diretto e pesato, trovare i cammini minimi all'interno del grafo.
- Diverse varianti
  - **Single-source shortest-paths problem**
    - Trovare un cammino minimo tra un nodo sorgente verso tutti gli altri.
  - **Single-destination shortest-paths problem**
    - Trovare un cammino minimo da tutti i nodi verso un particolare nodo destinazione.
  - **Single-pair shortest-path problem**
    - Data una coppia di nodi, trovare un cammino minimo tra essi.
  - **All-pairs shortest-paths problem**
    - Trovare un cammino minimo per ogni coppia di nodi nel grafo.
- e diversi algoritmi
  - **Bellman-Ford**: single-source su grafi con pesi arbitrari,
  - **Dijkstra**: single-source su grafi con pesi non negativi.

Edsger Dijkstra (1930 – 2002)

- Impatto e contributi in diverse aree
  - Algoritmica,
  - Programmazione concorrente e distribuita,
  - Paradigmi di programmazione,
  - Verifica formale dei programmi,
  - Sistemi operativi,
  - …
- Una citazione a lui attribuita:

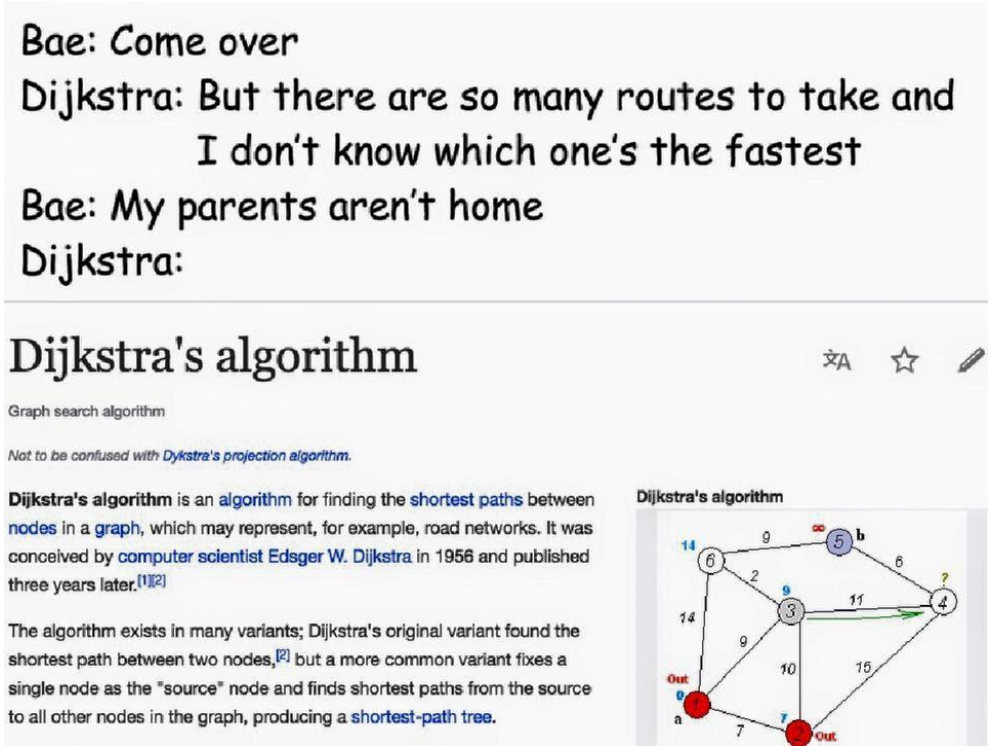
  "Computer science is no more about computers than astronomy is about telescopes."



source: wikipedia.org

Algoritmo di Dijkstra

- Soluzione alla variante **single-source shortest path** del problema
- Condizione:
  - I pesi sugli archi devono essere **non negativi**.
- Con una implementazione efficiente, è più veloce di Bellman-Ford.
- Intuizione del funzionamento
  - Itera partendo dal nodo sorgente x
  - Per ogni nodo v l'algoritmo memorizza due attributi:
    - la distanza del cammino minimo verso v trovato fino all'iterazione corrente;
    - il predecessore (il nodo u da cui si è arrivati a v)
  - Termina quando tutti i nodi sono stati visitati



Algoritmo di Dijkstra

```
1.  Dijkstra(G(V,E), w, x):
2.    for v in V:
3.      v.d = ∞
4.      v.π = null
5.    x.d = 0
6.
7.    S = {}
8.    Q = V
9.
10.   while(!Q.empty):
11.     u = ExtractMin(Q)
12.     S = S ∪ {u}
13.
14.     for v in adj(u):
15.       if v.d > u.d + w(u,v):
16.         v.d = u.d + w(u,v)
17.         v.π = u
```

Ogni nodo v ha due attributi:

- v.d = lunghezza del cammino minimo verso v
- v.π = nodo predecessore di v

L'algoritmo mantiene due insiemi:

- S = insieme dei nodi visitati (per i quali il cammino minimo è stato calcolato)
- Q = insieme dei nodi non ancora visitati

Funzioni di utilità:

- w(*u*,*v*) = restituisce il peso dell'arco (u,v)
- adj(*u*) = restituisce i nodi adiacenti a u
- ExtractMin(*Q*) = estrae da Q il nodo u con valore u.d più piccolo

Algoritmo di Dijkstra

```
1.  Dijkstra(G(V,E), w, x):
2.    for v in V:
3.      v.d = ∞
4.      v.π = null
5.    x.d = 0
6.
7.    S = {}
8.    Q = V
9.
10.   while(!Q.empty):
11.     u = ExtractMin(Q)
12.     S = S ∪ {u}
13.
14.     for v in adj(u):
15.       if v.d > u.d + w(u,v):
16.         v.d = u.d + w(u,v)
17.         v.π = u
```

Fase di inizializzazione

- v.d = ∞ (nessun nodo è stato ancora raggiunto) e v.π = null (nessun nodo ha un predecessore)
- x.d = 0 (il nodo sorgente ha distanza 0 da sé stesso)
- S = {} (nessun nodo è stato visitato) e Q = V (tutti i nodi sono ancora da visitare)

Algoritmo di Dijkstra

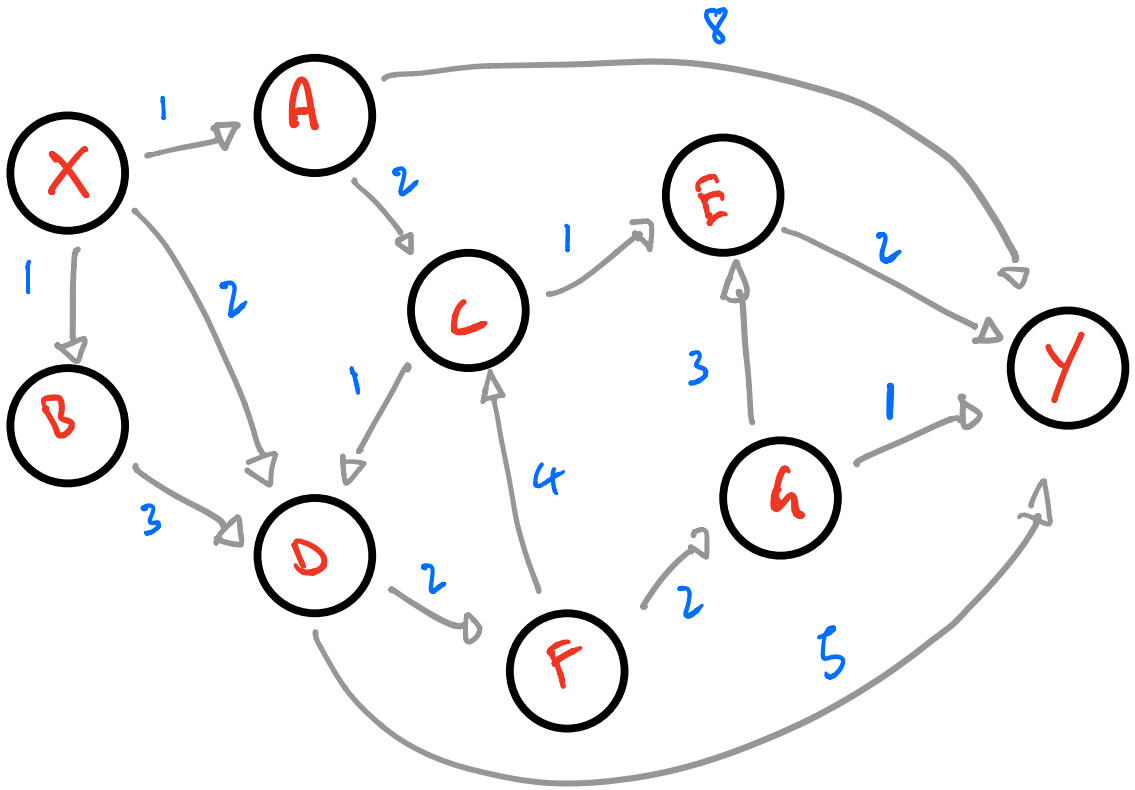
```
1.  Dijkstra(G(V,E), w, x):
2.    for v in V:
3.      v.d = ∞
4.      v.π = null
5.    x.d = 0
6.
7.    S = {}
8.    Q = V
9.
10.   while(!Q.empty):
11.     u = ExtractMin(Q)
12.     S = S ∪ {u}
13.
14.     for v in adj(u):
15.       if v.d > u.d + w(u,v):
16.         v.d = u.d + w(u,v)
17.         v.π = u
```

Fase di iterazione

- Si estrae da Q il nodo u con u.d più piccolo (nella prima iterazione, è il nodo x)
- Si segna u come visitato
- Per ogni nodo v adiacente ad u:
  - Se la distanza corrente di v è maggiore della somma della distanza corrente di u e il peso dell'arco (u,v) $\Rightarrow$ si aggiorna v.d e v.π

Fun fact: questa fase è anche chiamata *rilassamento*

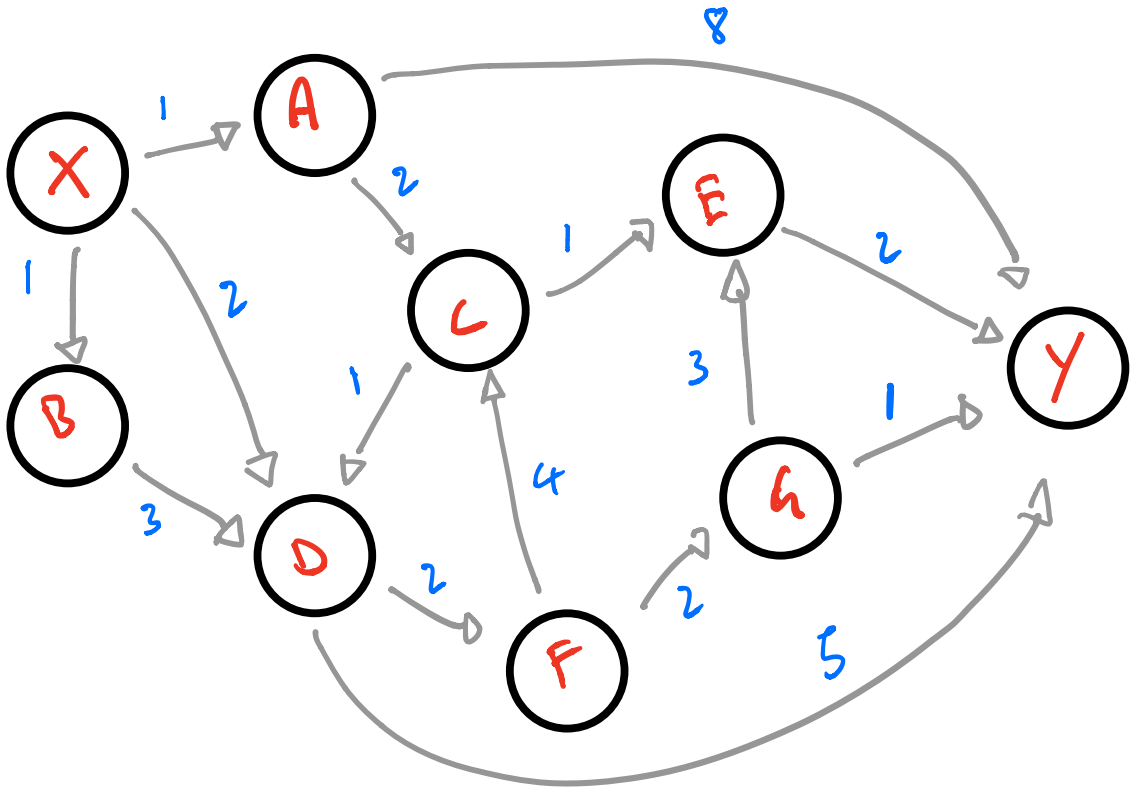
Esempio



Trovare il cammino minimo dal nodo X al nodo Y.

- Una semplice BFS/DFS non basta.
- Nel grafo non sono presenti archi con peso negativo
  - Possiamo usare Dikstra.

Esempio



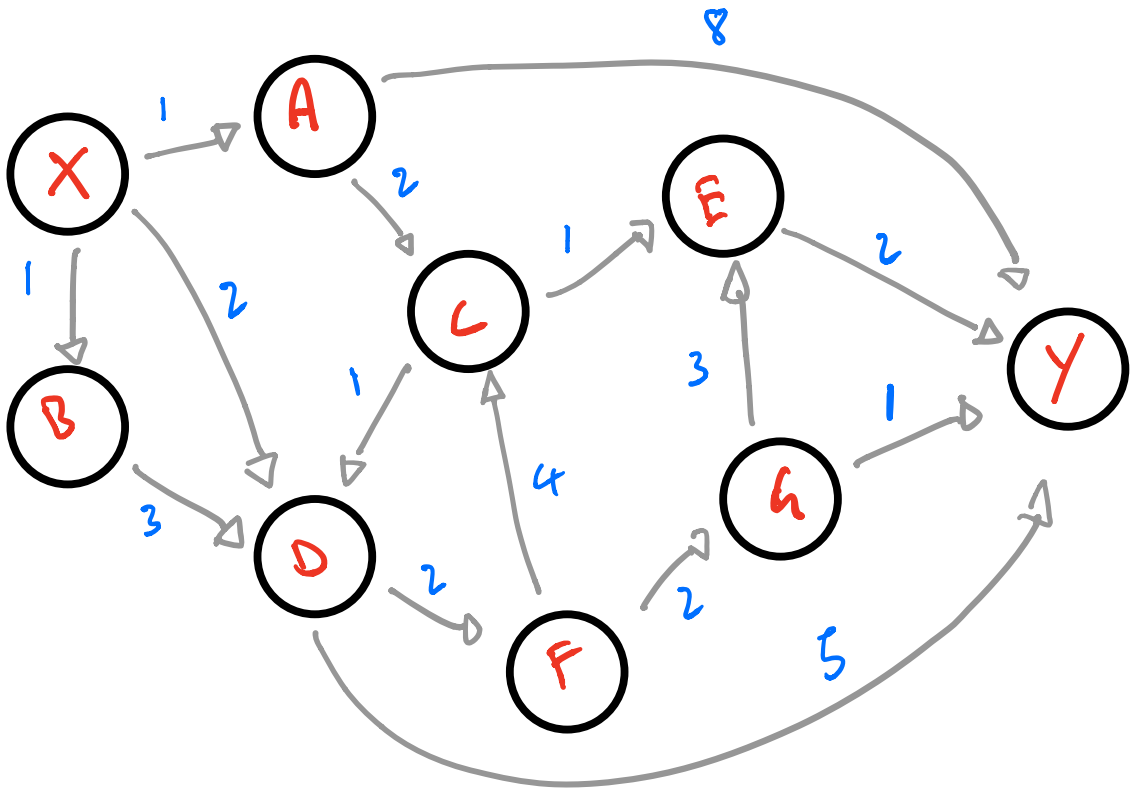
Fase di inizializzazione, creazione di S e Q

	A	B	C	D	E	F	G	X	Y
d	∞	∞	∞	∞	∞	∞	∞	0	∞
π	–	–	–	–	–	–	–	–	–

$S = \{\}$

$Q = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}, \text{G}, \text{X}, \text{Y}\}$

Esempio



Iterazione:

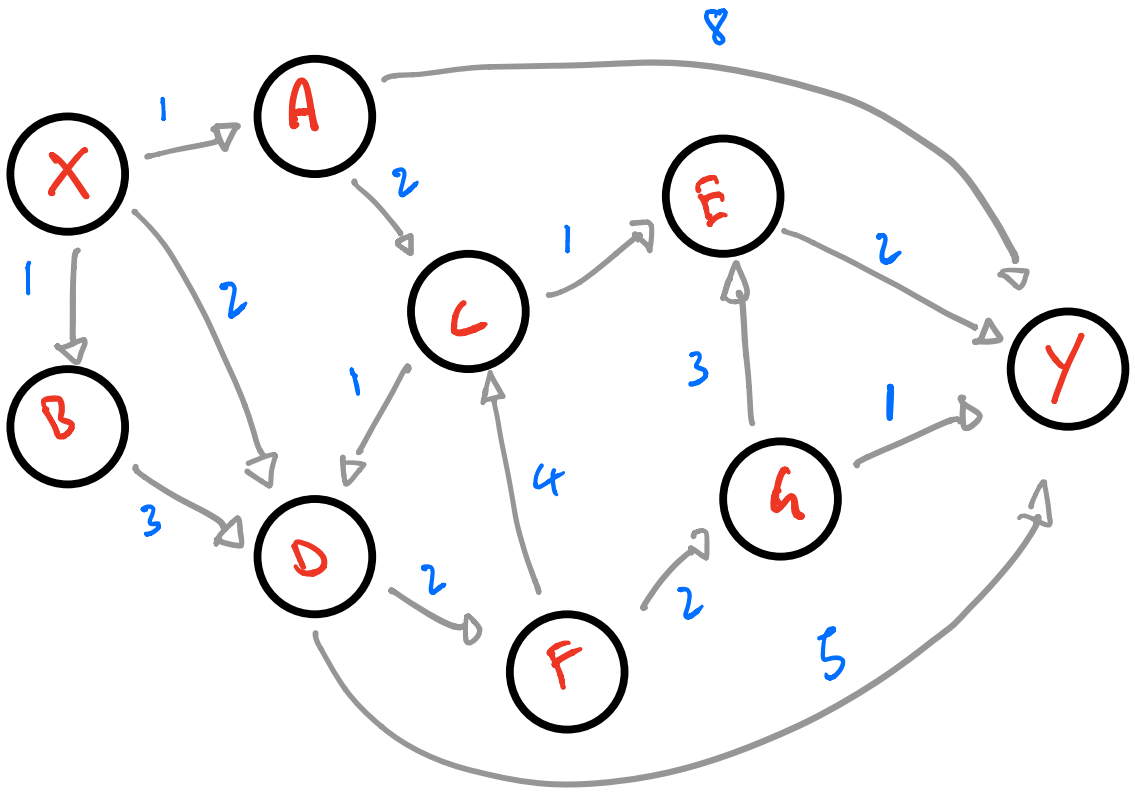
- nodo con d minimo in Q: X
- aggiungo X ad S
- aggiorno d degli adiacenti di X
- aggiorno π degli adiacenti di X

	A	B	C	D	E	F	G	X	Y
d	1	1	∞	2	∞	∞	∞	0	∞
π	X	X	–	X	–	–	–	–	–

$S = \{\text{X}\}$

$Q = \{\text{A}, \text{B}, \text{C}, \text{D}, \text{E}, \text{F}, \text{G}, \text{Y}\}$

Esempio



Iterazione:

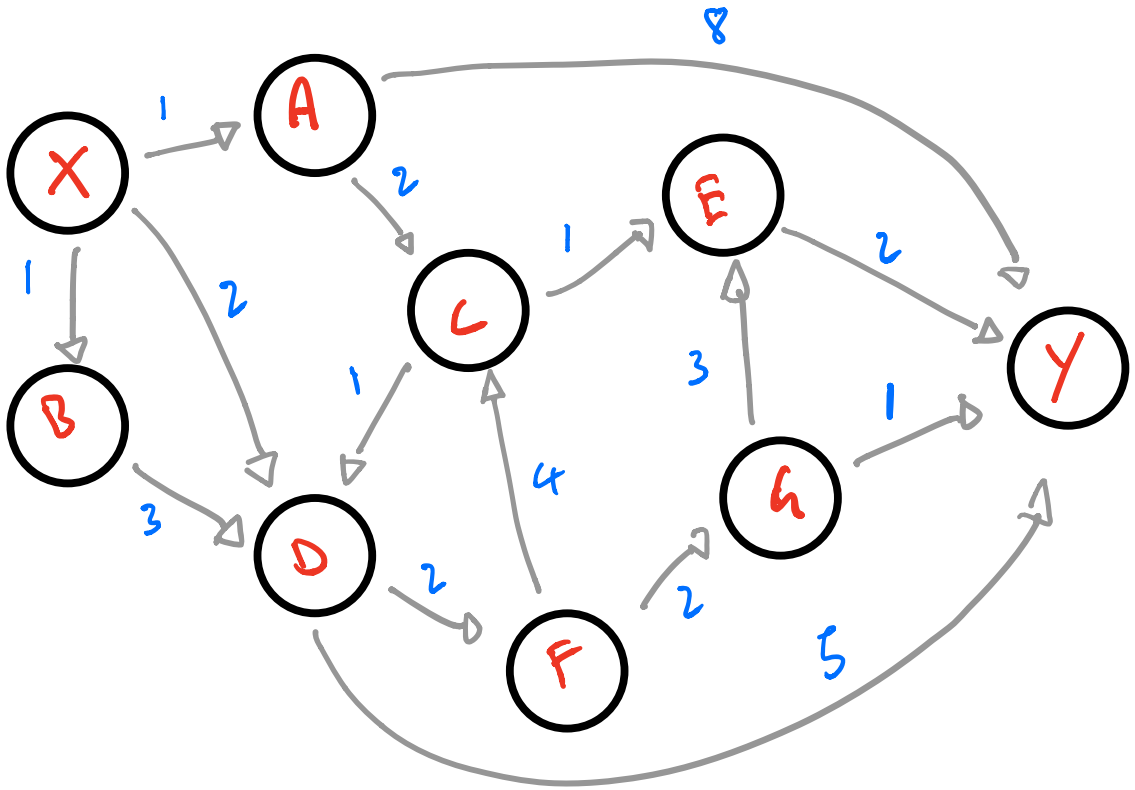
- nodo con d minimo in Q: A
- aggiungo A ad S
- aggiorno d degli adiacenti di A
- aggiorno π degli adiacenti di A

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	∞	∞	∞	0	9
π	X	X	A	X	–	–	–	–	A

$S = \{\text{X}, \text{A}\}$

$Q = \{\text{B}, \text{C}, \text{D}, \text{E}, \text{F}, \text{G}, \text{Y}\}$

Esempio



Iterazione:

- nodo con d minimo in Q: B
- aggiungo B ad S
- aggiorno d degli adiacenti di B
- aggiorno π degli adiacenti di B

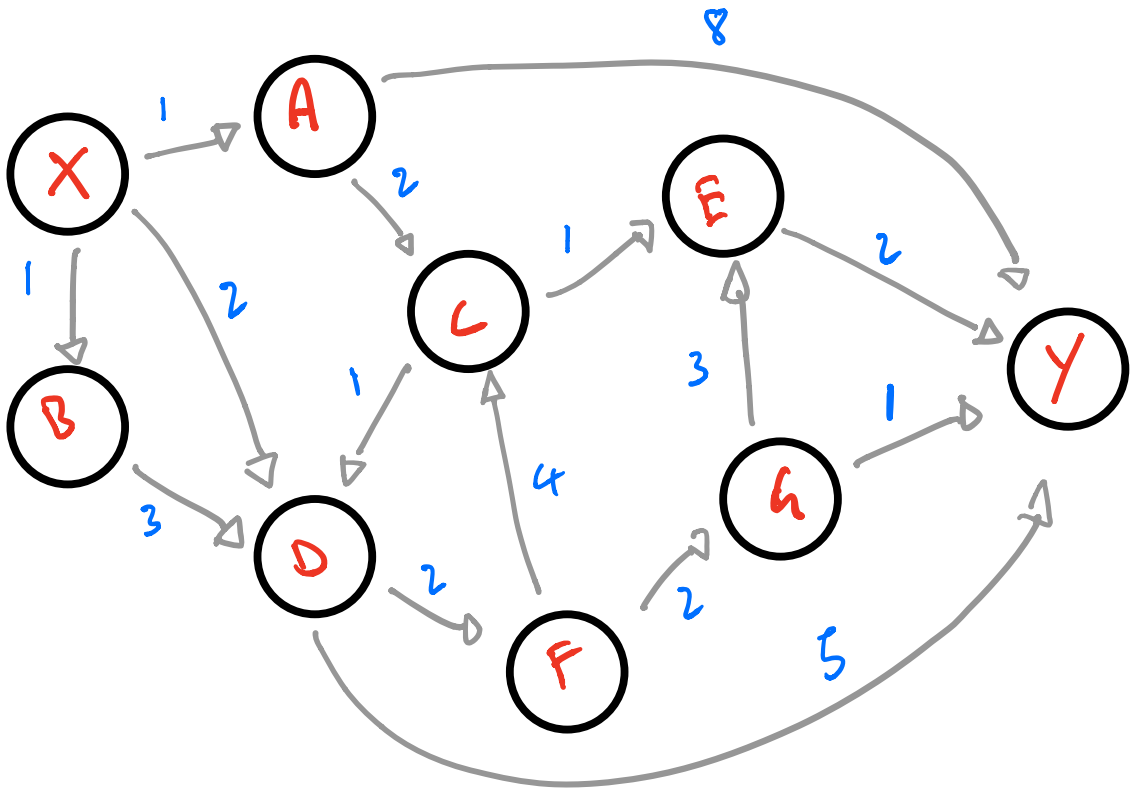
	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	∞	∞	∞	0	9
π	X	X	A	X	–	–	–	–	A

$$S = \{\text{X}, \text{A}, \text{B}\}$$

$$Q = \{\text{C}, \text{D}, \text{E}, \text{F}, \text{G}, \text{Y}\}$$

Fun fact: qui non faccio niente, poiché il costo di arrivare a D tramite B è maggiore di quello dato dal cammino già scoperto in precedenza.

Esempio



Iterazione:

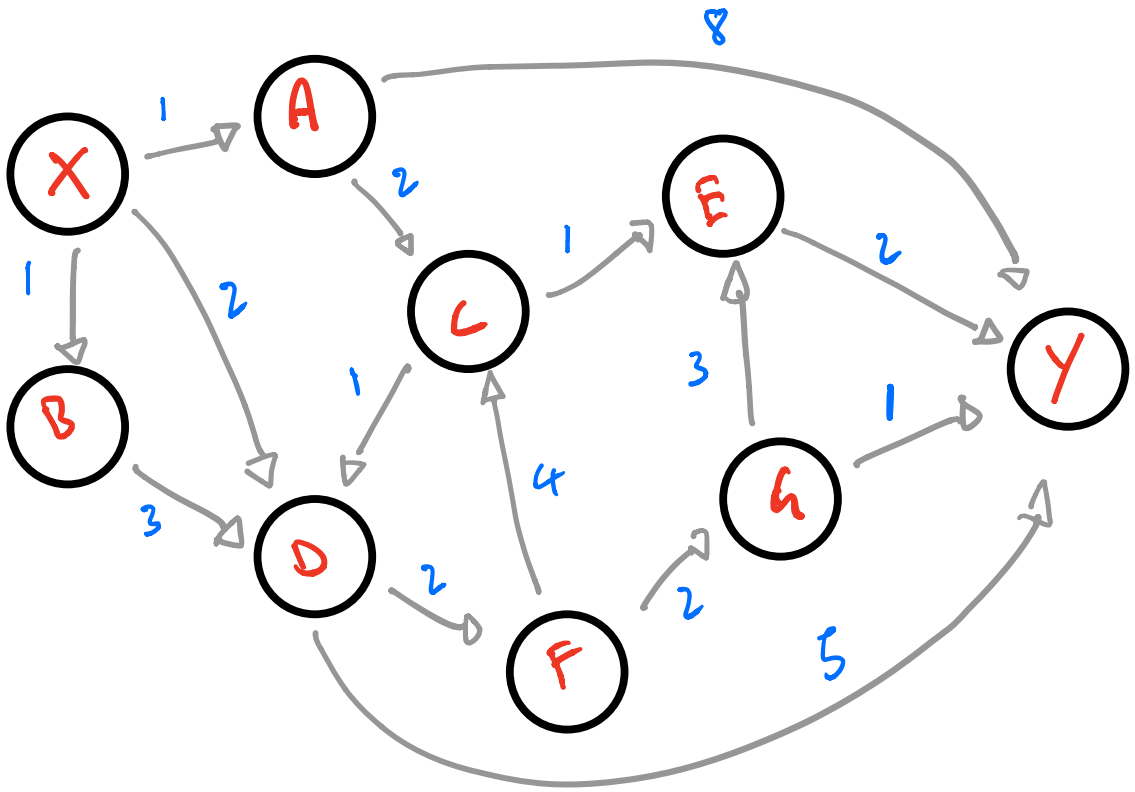
- nodo con d minimo in Q: D
- aggiungo D ad S
- aggiorno d degli adiacenti di D
- aggiorno π degli adiacenti di D

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	∞	4	∞	0	7
π	X	X	A	X	–	D	–	–	D

$S = \{\text{X}, \text{A}, \text{B}, \text{D}\}$

$Q = \{\text{C}, \text{E}, \text{F}, \text{G}, \text{Y}\}$

Esempio



Iterazione:

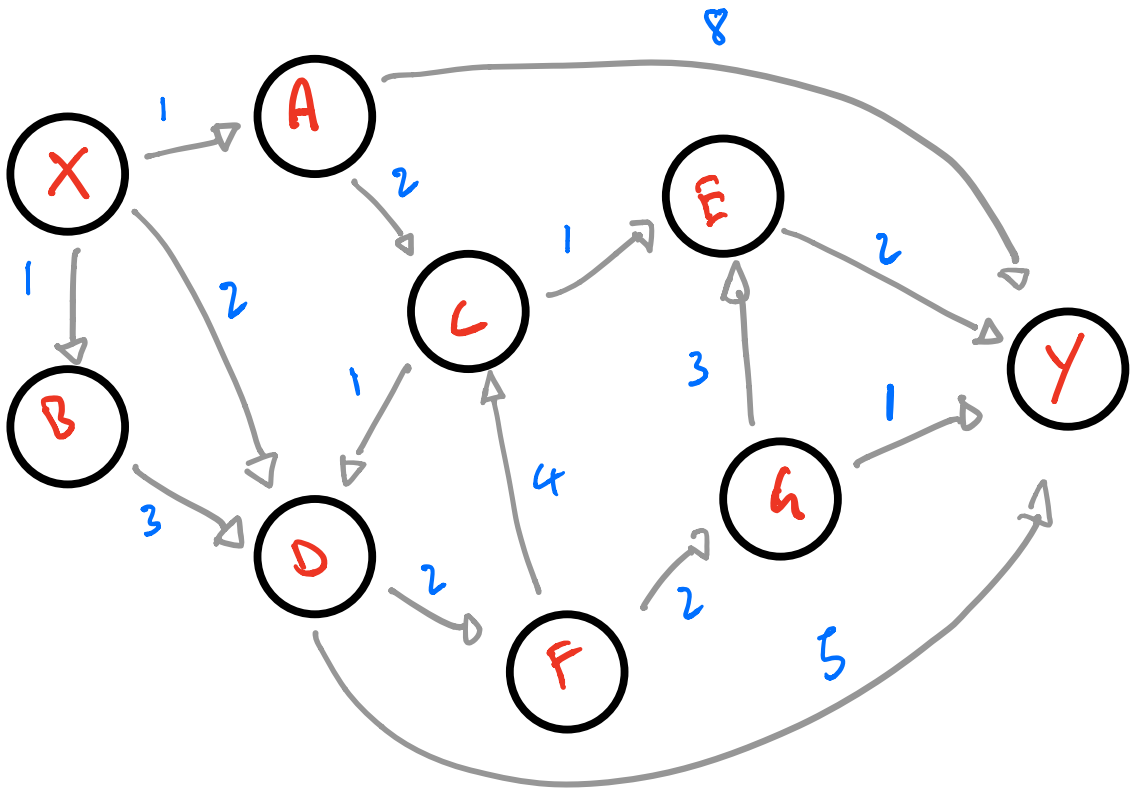
- nodo con *d* minimo in *Q:* C
- aggiungo C ad *S*
- aggiorno *d* degli adiacenti di C
- aggiorno π degli adiacenti di C

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	4	4	∞	0	7
π	X	X	A	X	C	D	–	–	D

$S = \{\text{X}, \text{A}, \text{B}, \text{D}, \text{C}\}$

$Q = \{\text{E}, \text{F}, \text{G}, \text{Y}\}$

Esempio



Iterazione:

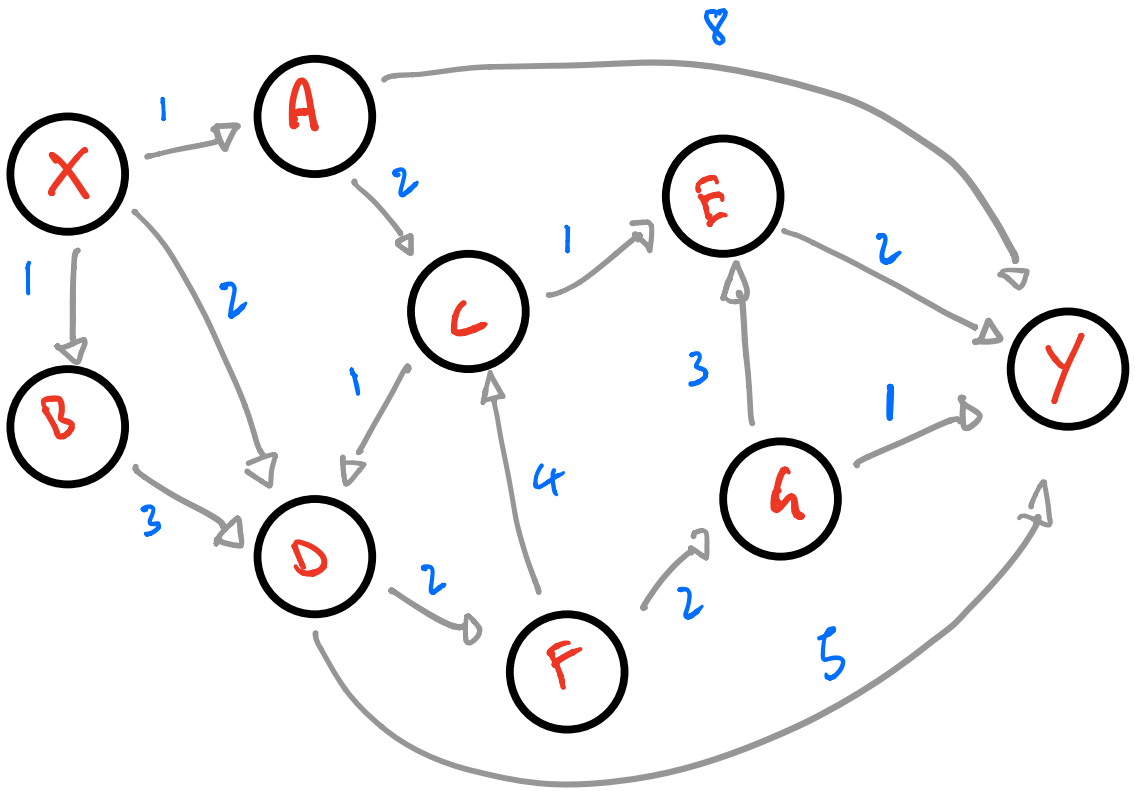
- nodo con *d* minimo in *Q:* E
- aggiungo E ad *S*
- aggiorno *d* degli adiacenti di E
- aggiorno π degli adiacenti di E

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	4	4	∞	0	6
π	X	X	A	X	C	D	–	–	E

$S = \{\text{X}, \text{A}, \text{B}, \text{D}, \text{C}, \text{E}\}$

$Q = \{\text{F}, \text{G}, \text{Y}\}$

Esempio



Iterazione:

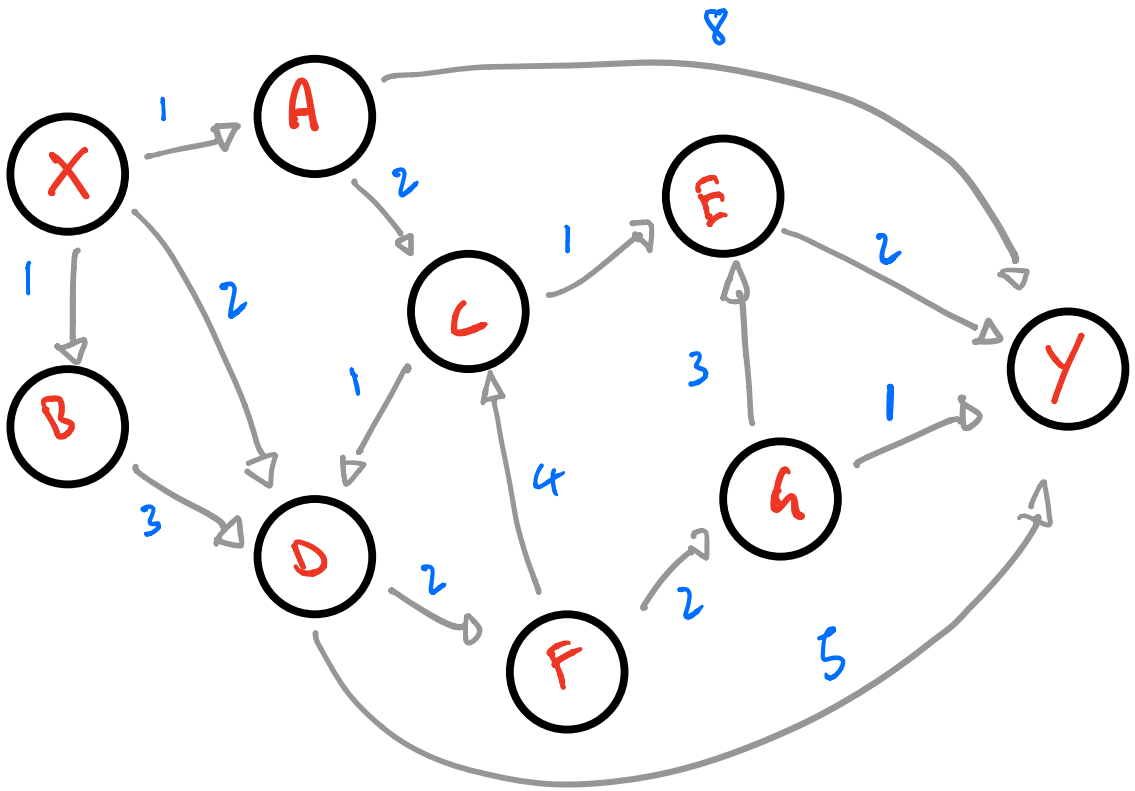
- nodo con *d* minimo in *Q:* F
- aggiungo F ad *S*
- aggiorno *d* degli adiacenti di F
- aggiorno π degli adiacenti di F

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	4	4	6	0	6
π	X	X	A	X	C	D	F	–	E

$S = \{\text{X}, \text{A}, \text{B}, \text{D}, \text{C}, \text{E}, \text{F}\}$

$Q = \{\text{G}, \text{Y}\}$

Esempio



Iterazione:

- nodo con *d* minimo in *Q:* G
- aggiungo G ad *S*
- aggiorno *d* degli adiacenti di G
- aggiorno π degli adiacenti di G

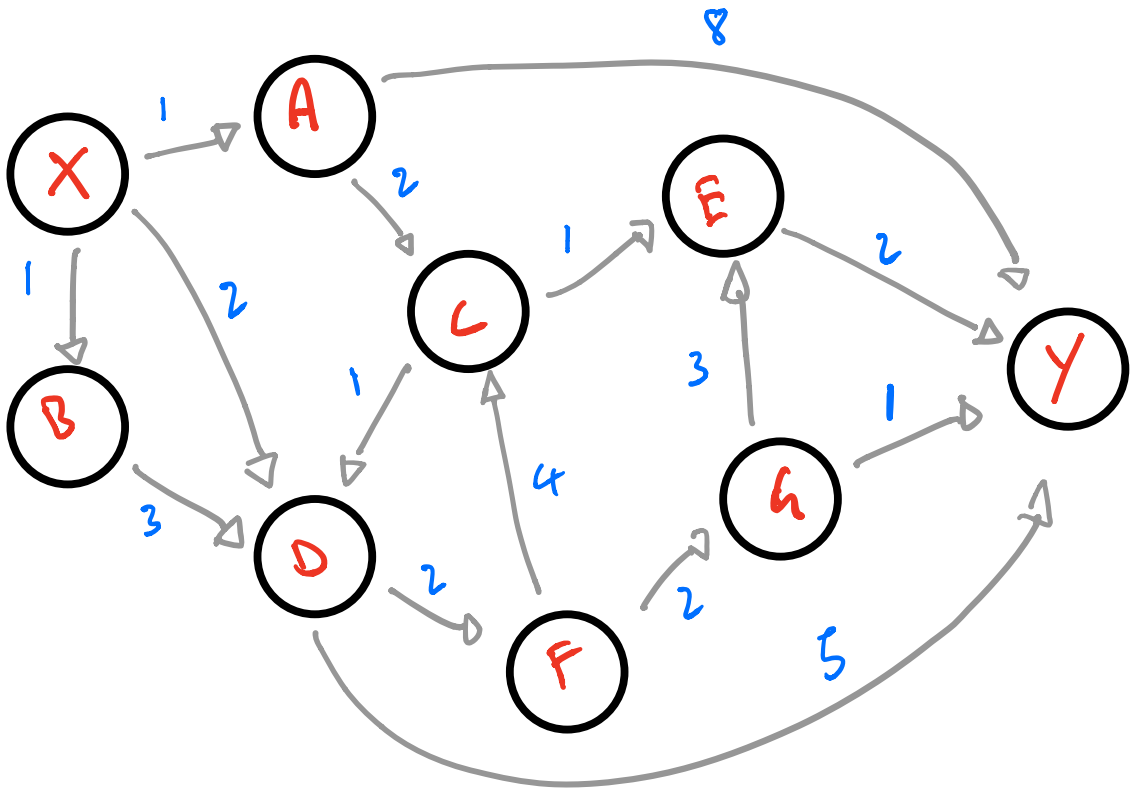
	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	4	4	6	0	6
π	X	X	A	X	C	D	F	–	E

$$S = \{X, A, B, D, C, E, F, G\}$$

$$Q = \{Y\}$$

Fun fact: qui non faccio niente, poiché il costo di arrivare a Y tramite G è maggiore di quello dato dal cammino già scoperto in precedenza.

Esempio



Iterazione:

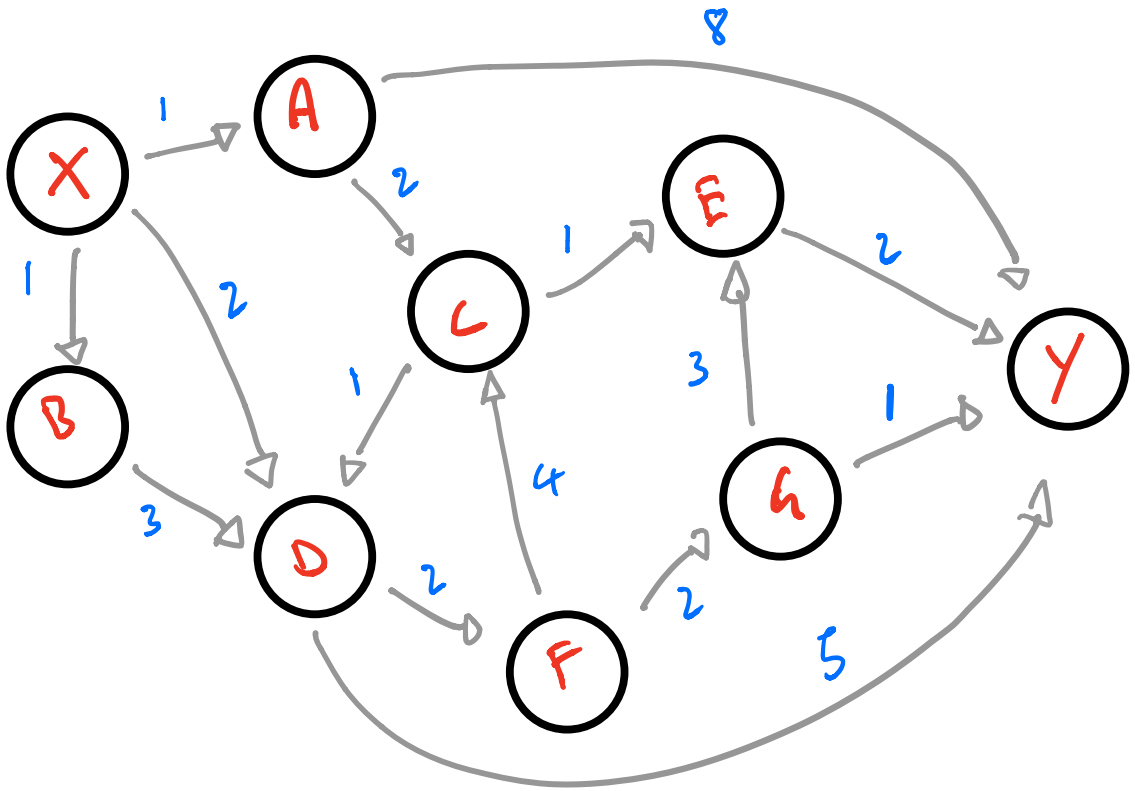
- nodo con *d* minimo in *Q:* Y
- aggiungo Y ad *S*
- aggiorno *d* degli adiacenti di Y
- aggiorno π degli adiacenti di Y

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	4	4	6	0	6
π	X	X	A	X	C	D	F	–	E

$S = \{\text{X}, \text{A}, \text{B}, \text{D}, \text{C}, \text{E}, \text{F}, \text{G}, \text{Y}\}$

$Q = \{\}$

Esempio



STOP!

- Q è adesso vuota,
- Il costo del cammino minimo da X a Y è pari a 6.

	A	B	C	D	E	F	G	X	Y
d	1	1	3	2	4	4	6	0	6
π	X	X	A	X	C	D	F	–	E

$$S = \{\text{X}, \text{A}, \text{B}, \text{D}, \text{C}, \text{E}, \text{F}, \text{G}, \text{Y}\}$$

$$Q = \{\}$$

Fun fact: per ricostruire il cammino, basta guardare i predecessori π, ottenendo: Y ← E ← C ← A ← X.