POLITECNICO
DI TORINO

Dipartimento
di Automatica e Informatica

I cammini minimi

Paolo Camurati

I cammini minimi

G=(V,E) grafo orientato, pesato (w: E$\rightarrow$***R***).

Definizioni:

peso w(p) di un cammino p:

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

peso $\delta(s,v)$ di un cammino minimo da s a v:

$$\delta(s,v) = \begin{cases} \min\{w(p): \text{se } \exists\ s \rightarrow_p v\ \} \\ \\ \infty \quad \text{altrimenti} \end{cases}$$

Cammino minimo da s a v:

qualsiasi cammino p con $w(p) = \delta(s,v)$

Problemi classici

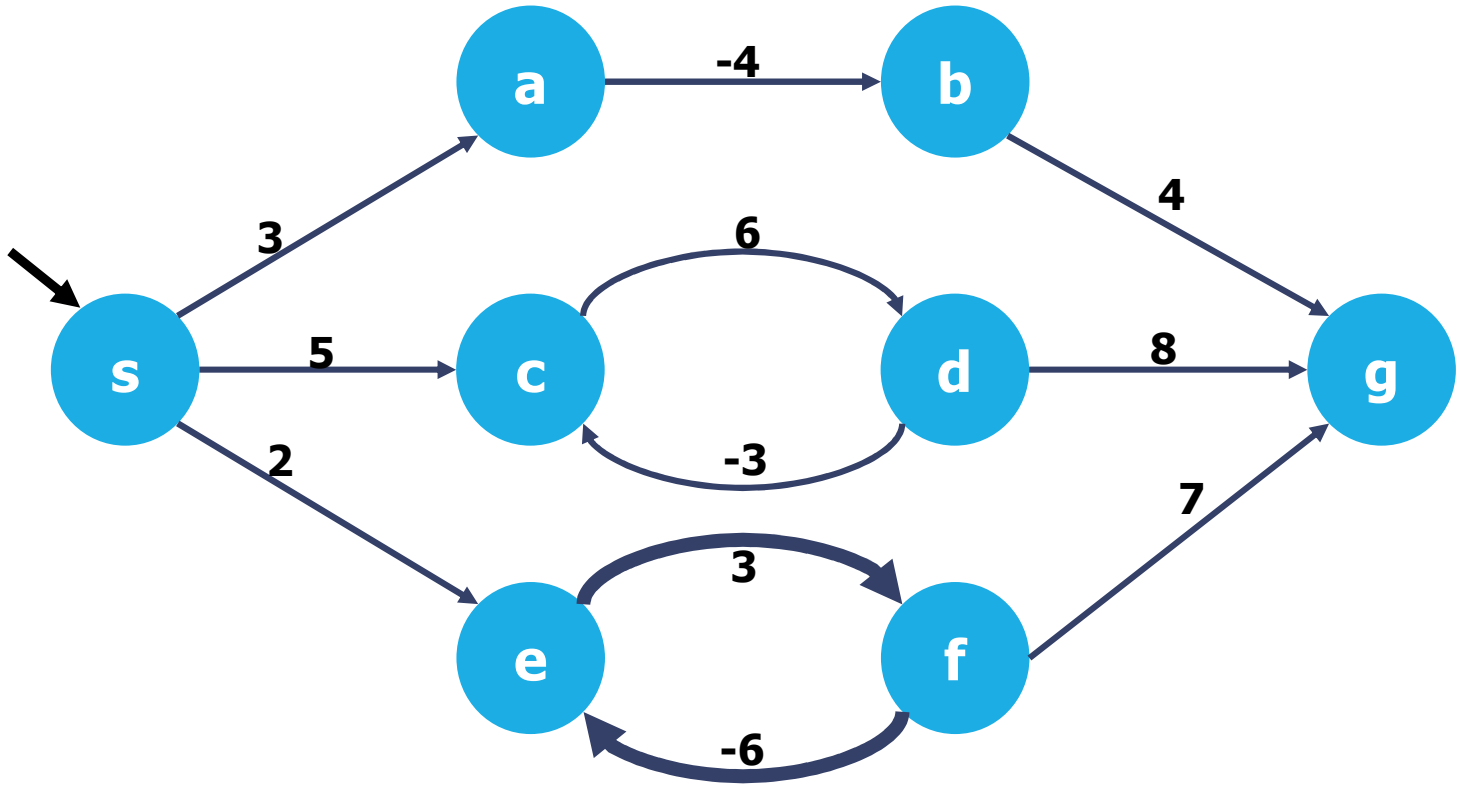
Cammini minimi:

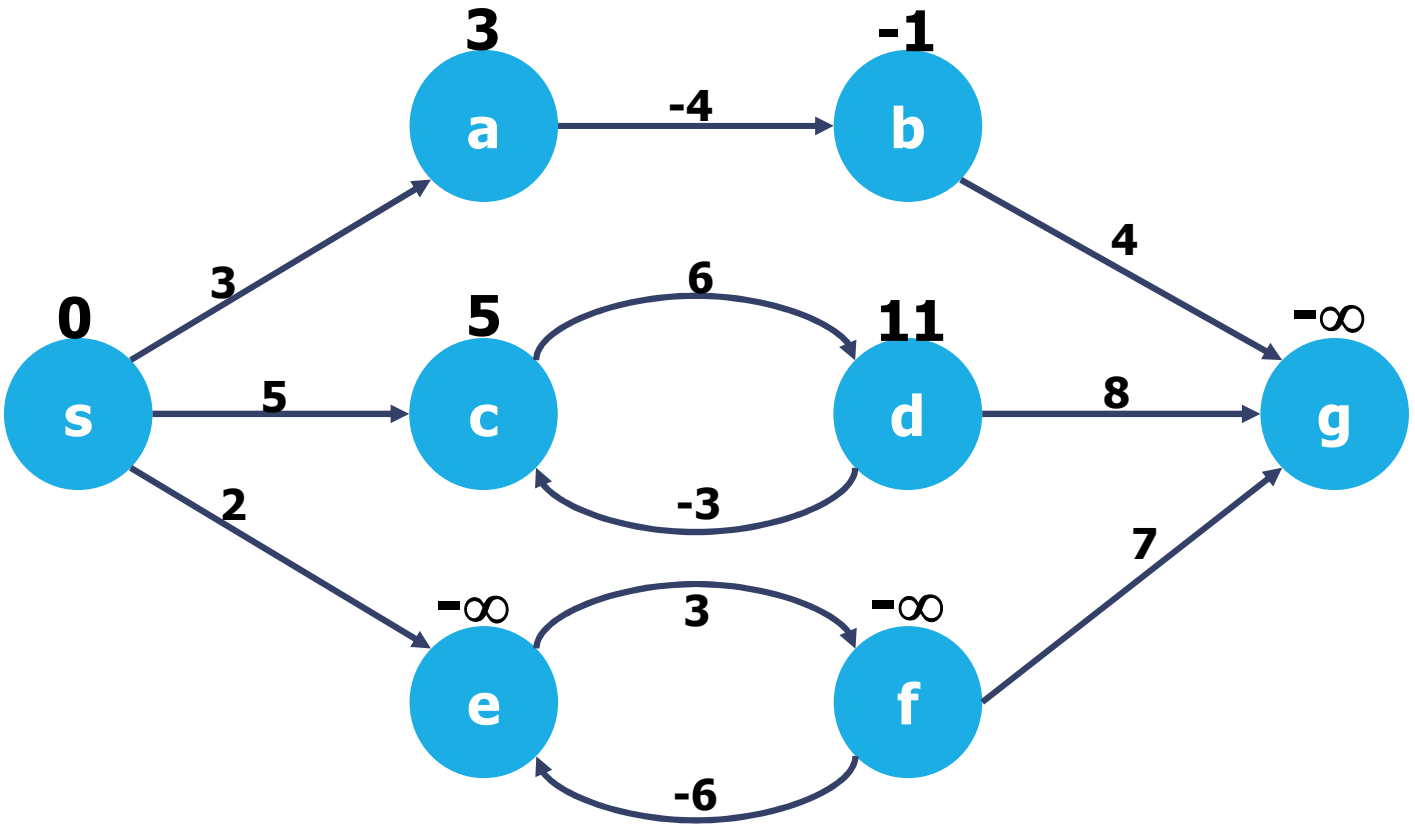
- da sorgente singola: cammino minimo e suo peso da s a ogni altro vertice v
  - algoritmo di Dijkstra
  - algoritmo di Bellman-Ford
- con destinazione singola
- tra una coppia di vertici
- tra tutte le coppie di vertici.

Archi con pesi negativi

- $\exists$ (u,v) $\in$ E per cui w(u,v) < 0 ma $\nexists$ ciclo a peso < 0:
  - algoritmo di Djikstra: soluzione ottima non garantita
  - algoritmo di Bellman-Ford: soluzione ottima garantita
- $\exists$ ciclo a peso < 0: problema privo di significato, $\nexists$ soluzione:
  - algoritmo di Djikstra: risultato senza significato
  - algoritmo di Bellman-Ford: rileva ciclo <0.

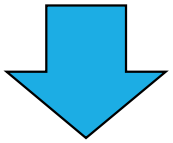
Esempio




3
-1
a
-4
b
4
3
6
0
5
11
-∞
s
5
c
d
8
g
-3
2
7
-∞
3
-∞
e
f
-6

Cammini minimi e cicli

- I cammini minimi non possono contenere cicli a peso negativo in quanto il problema perderebbe di significato
- I cammini minimi non possono contenere cicli a peso positivo, in quanto non sarebbero minimi (eliminando il ciclo si avrebbe peso inferiore)



- I cammini minimi sono semplici
- I cammini minimi constano di al più |V|-1 archi

Approccio brute-force

In assenza di cicli a peso negativo:

- enumerare i sottoinsiemi di archi di cardinalità tra 1 e |V|-1
- l'ordine conta, quindi powerset con disposizioni semplici
- accettabilità: il sottoinsieme di archi forma un cammino
- ottimalità: selezionare quello a peso minimo.

Costo esponenziale.

Sottostruttura ottima di un cammino minimo

Lemma: un sottocammino di un cammino minimo è un cammino minimo.

G = (V,E): grafo orientato, pesato w: $E \rightarrow \boldsymbol{R}$.

$p = \langle v_1, v_2, \ldots, v_k \rangle$: un cammino minimo da v_1 a v_k.

$\forall i, j\ 1 \leq i \leq j \leq k$, $p_{ij} = \langle v_i, v_{i+1}, \ldots, v_j \rangle$ (sottocammino di p da v_i a v_j)

p_{ij} è un cammino minimo da v_i a v_j.

Dimostrazione (per assurdo):

Scomponiamo p in p_{1i}, p_{ij} e p_{jk} con $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$

Se p_{ij} non fosse minimo, esisterebbe un p'_{ij} minimo con peso $w'(p_{ij}) < w(p_{ij})$.

Sostituendo p'_{ij} a p_{ij}, w(p) non sarebbe minimo, contraddicendo l'ipotesi.

Rappresentazione dei cammini minimi

1 - Vettore dei predecessori st[v]:

$$\forall v \in V \quad st[v] = \begin{cases} \text{parent(v) se } \exists \\ \\ \text{-1 altrimenti} \end{cases}$$

2- Sottografo dei predecessori:

$G_\pi=(V_\pi,E_\pi)$, dove

- $V_\pi = \{v \in V: st[v] \text{ != } -1\} \cup \{s\}$
- $E_\pi = \{(st[v], v) \in E : v \in V_\pi - \{s\}\}$

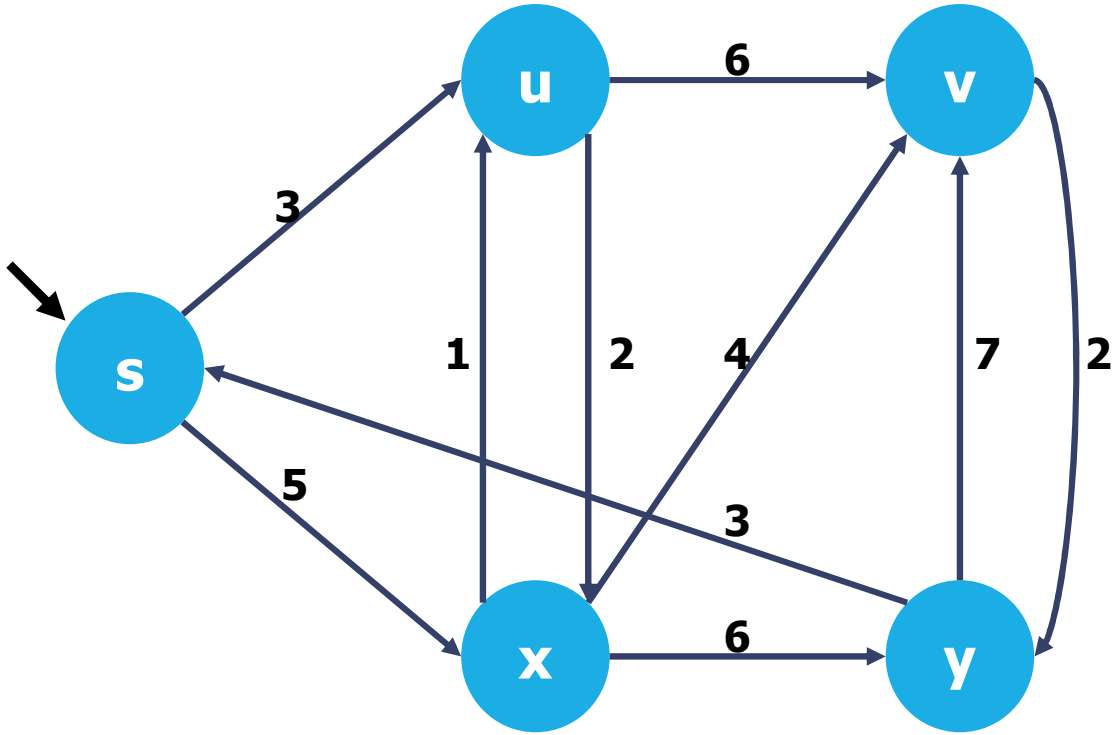
3- Albero dei cammini minimi:

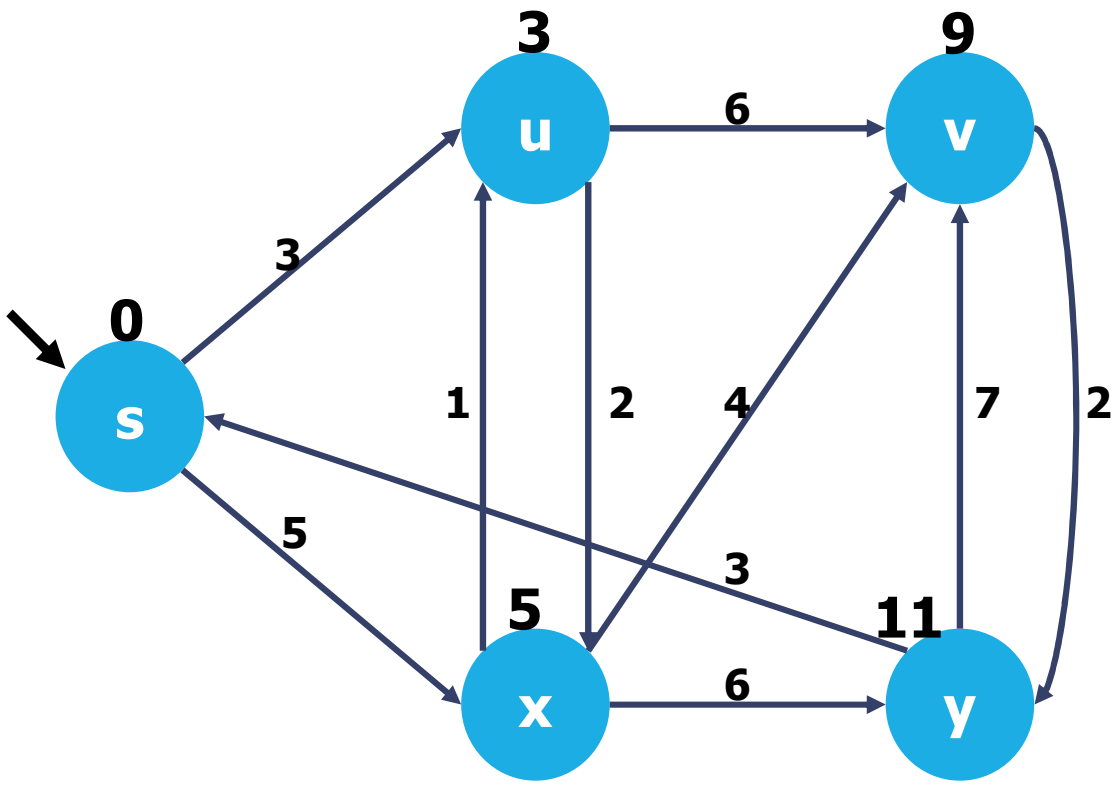
G’ = (V’, E’) dove $V' \subseteq V$ && $E' \subseteq E$

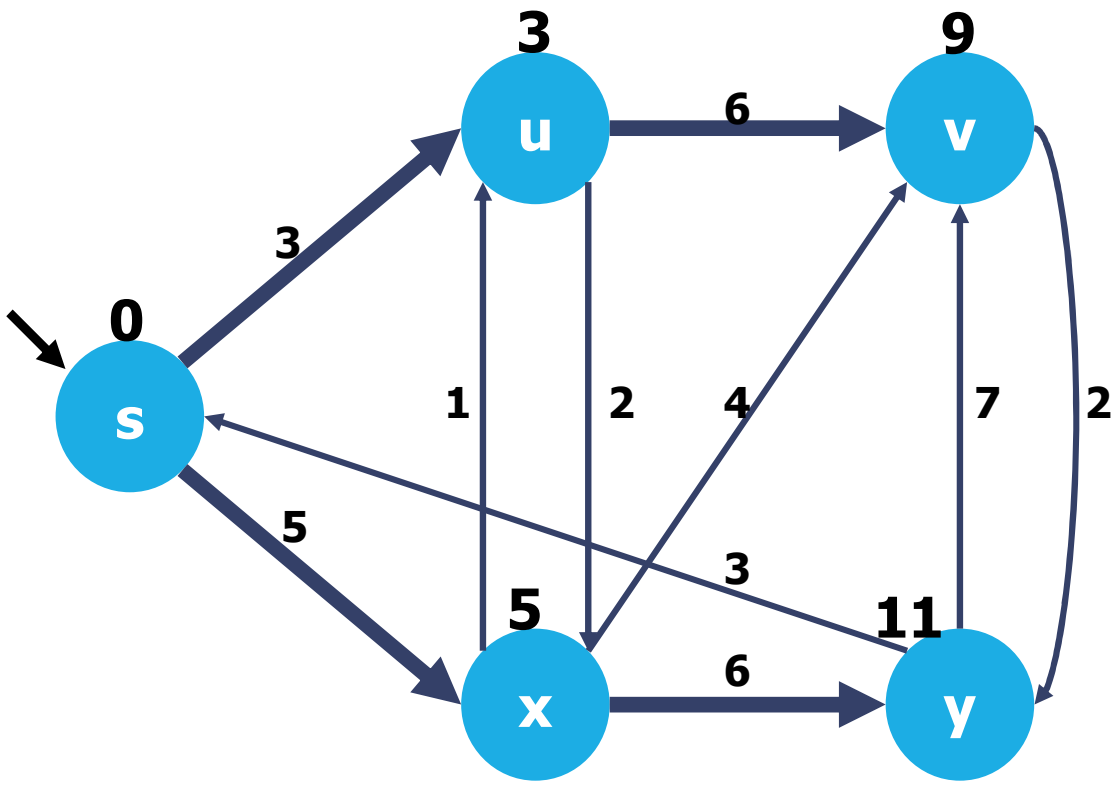
- V’: insieme dei vertici raggiungibili da s
- s radice dell’albero
- $\forall v \in V'$ l’unico cammino semplice da s a v in G’ è un cammino minimo da s a v in G.

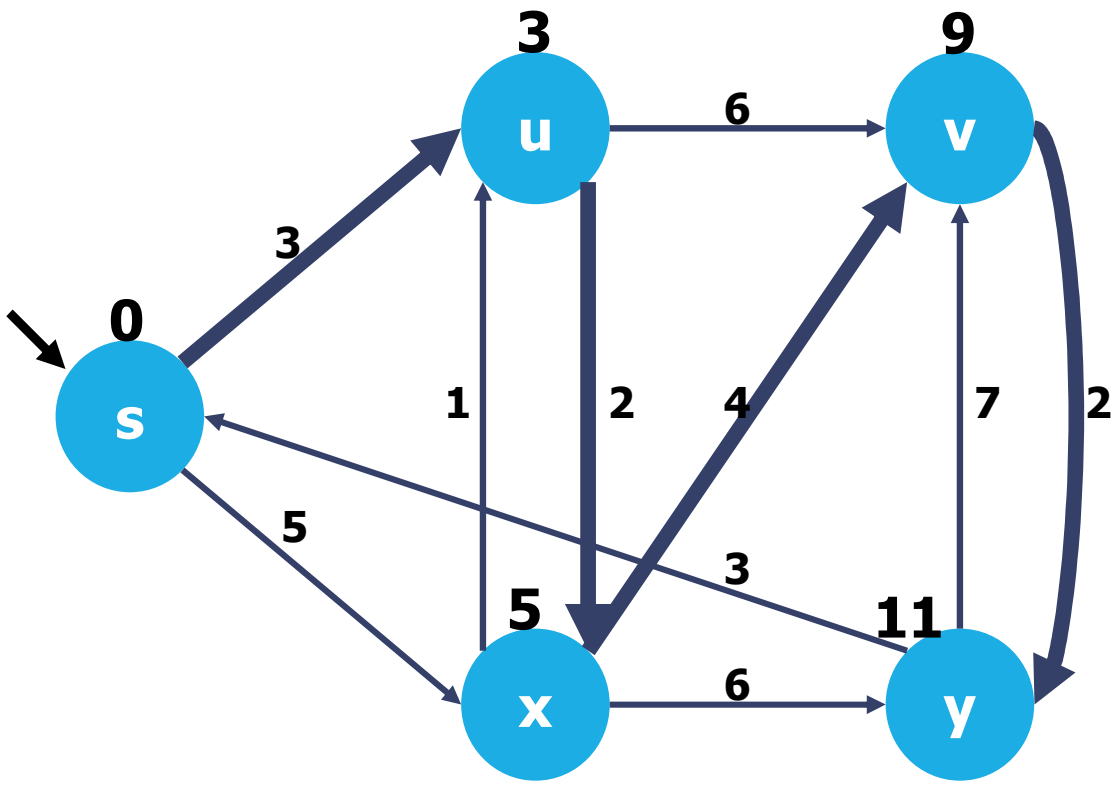
Nei grafi non pesati: si ottiene con una visita in ampiezza.

Esempio




3
9
u
6
v
3
0
s
1
2
4
7
2
5
3
5
11
x
6
y

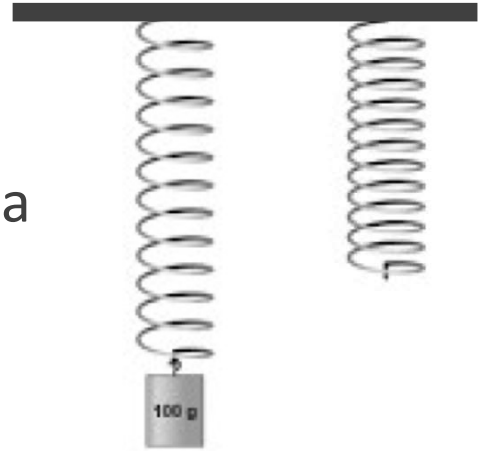

3
9
u
6
v
3
0
s
1
2
4
7
2
5
3
5
11
x
6
y


3
9
u
6
v
3
0
s
1
2
4
7
2
5
3
5
11
x
6
y

Rilassamento (relaxation)

In generale: lasciare che una soluzione violi un vincolo temporaneamente e poi eliminare la violazione.

Analogia: molla elicoidale di tensione



sovrastima del cammino minimo sta a molla tesa
come
cammino minimo sta a molla a riposo

- d[v]: stima (limite superiore) del peso del cammino minimo da s a v

  inizialmente:

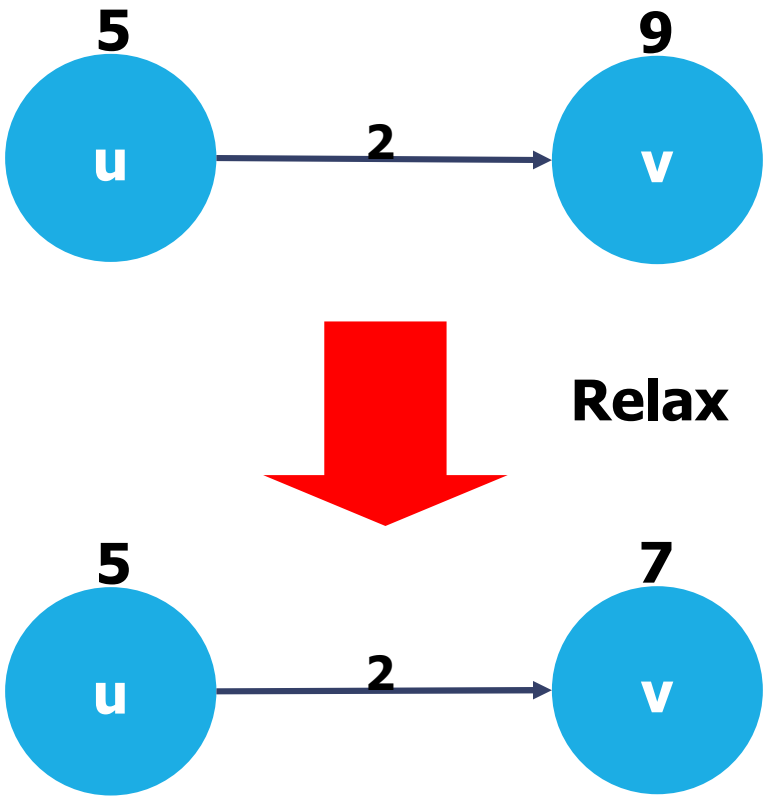
  $\forall v \in V$ d[v]= maxWT, st[v] = -1;

  d[s] = 0; st[s] = 0;

- rilassare: d[v] e st[v] verificando se conviene il cammino da s a u e l' arco e = (u,v), dove w(u,v) è il peso dell'arco:

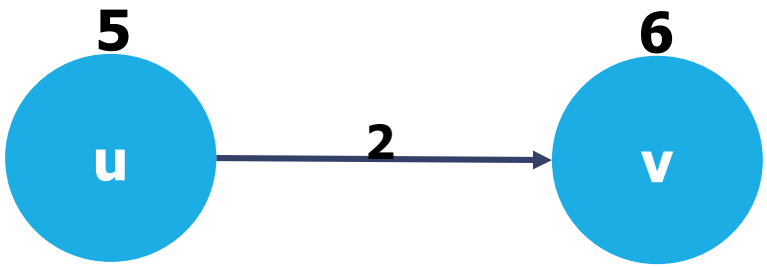
```
if (d[v]>d[u]+w(u,v)) {
  d[v] = d[u]+w(u,v);
  st[v] = u;
}
```

Esempio

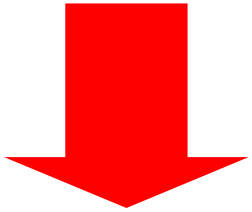


d[v] =9
d[u] = 5
w(u,v) = 2
d[v] > d[u] + w(u,v)

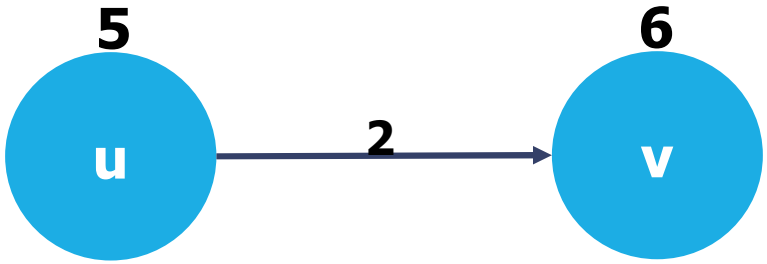
d[v] = 7
st[v] = u
cammino minimo da s a v =
cammino minimo da s a u +
arco (u,v)



d[v] =6
d[u] = 5
w(u,v) = 2
d[v] < d[u] + w(u,v)



Relax



Il rilassamento non ha avuto effetto.

Proprietà

Lemma (disuguaglianza triangolare):

G=(V,E): grafo orientato, pesato w: E$\rightarrow$***R*** con sorgente s.

$\forall(u,v) \in E$

$$\delta(s,v) \leq \delta(s,u) + w(u,v)$$

Un cammino minimo da s a v non può avere peso maggiore del cammino formato da un cammino minimo da s a u e da un arco (u, v).

Lemma (proprietà del limite superiore):

G=(V,E): grafo orientato, pesato w: E$\rightarrow$***R*** con sorgente s e stime per i vertici inizializzate.

$\forall v \in V$

$$d(v) \geq \delta(s,v)$$

Una volta che il limite superiore d(v) assume il valore $\delta(s,v)$ esso non cambia più.

Corollario: (proprietà dell'assenza di cammino):

G=(V,E): grafo orientato, pesato w: E$\rightarrow$***R*** con stime per i vertici inizializzate.

Se non esiste un cammino tra s e v, allora si ha sempre

$$d(v) = \delta(s,v) = \infty$$

Lemma:

G=(V,E): grafo orientato, pesato w: E$\rightarrow$***R***.
e = (u,v) $\in$ E

Dopo il rilassamento di e = (u,v) si ha che

$$d[v] \leq d[u] + w(u,v)$$

A seguito del rilassamento d[v] non può essere aumentato, ma

- o è rimato invariato (rilassamento senza effetto)
- o è diminuito per effetto del rilassamento.

Lemma (proprietà della convergenza):

G=(V,E): grafo orientato, pesato w: E$\rightarrow$***R*** con sorgente s e stime per i vertici inizializzate.

Sia il cammino minimo da s a v composto da

- cammino da s a u
- arco e = (u,v)

A seguito dell'applicazione del rilassamento su e = (u,v)

se prima del rilassamento d[u] = δ(s,u)

dopo il rilassamento d[v] = δ(s,v).

Lemma (proprietà del rilassamento):

G=(V,E): grafo orientato, pesato w: E$\rightarrow$***R*** con sorgente s e stime per i vertici inizializzate

Se p=<v_1, v_2, …, v_k> è un cammino minimo da v_1 a v_k

dopo tutti i passi di rilassamento sugli archi

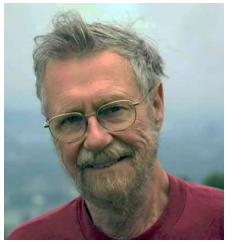
- $d(v_k) = \delta(s,v_k)$
- e $d(v_k)$ non cambia più.

Applicazione

Rilassamento:

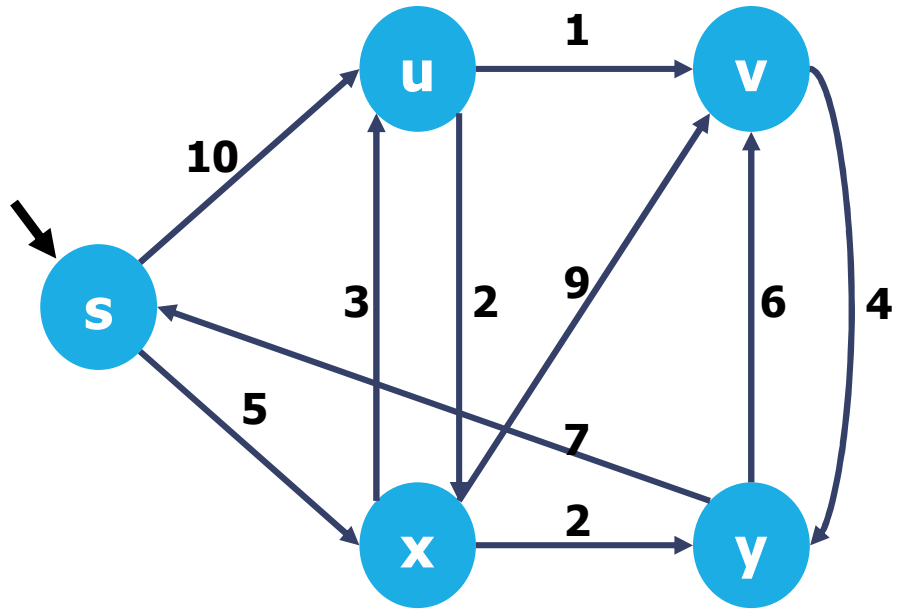
- applicato 1 volta ad ogni arco (Dijkstra) o più volte (Bellman-Ford)
- ordine con cui si rilassano gli archi.

Algoritmo di Dijkstra



- Ipotesi: $\nexists$ archi a peso < 0
- Strategia: greedy
- S: insieme dei vertici il cui peso sul cammino minimo da s è già stato determinato
- V-S: coda a priorità `PQ` dei vertici ancora da stimare. Termina per `PQ` vuota:
  - estrae u da V-S (d[u] minimo)
  - inserisce u in S
  - rilassa tutti gli archi uscenti da u.

Esempio

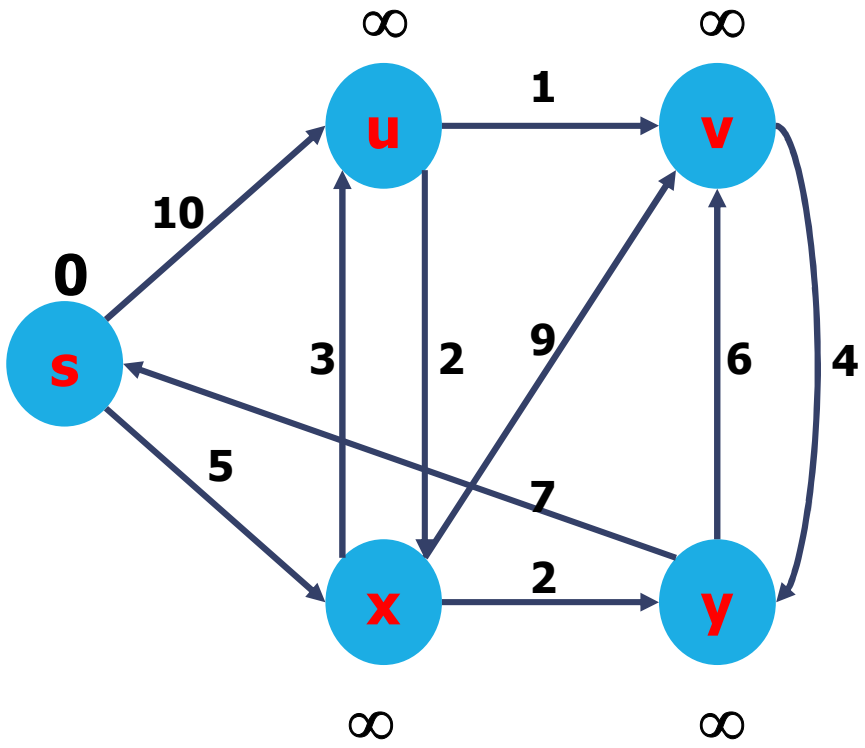


ST

0	s
1	u
2	x
3	v
4	y

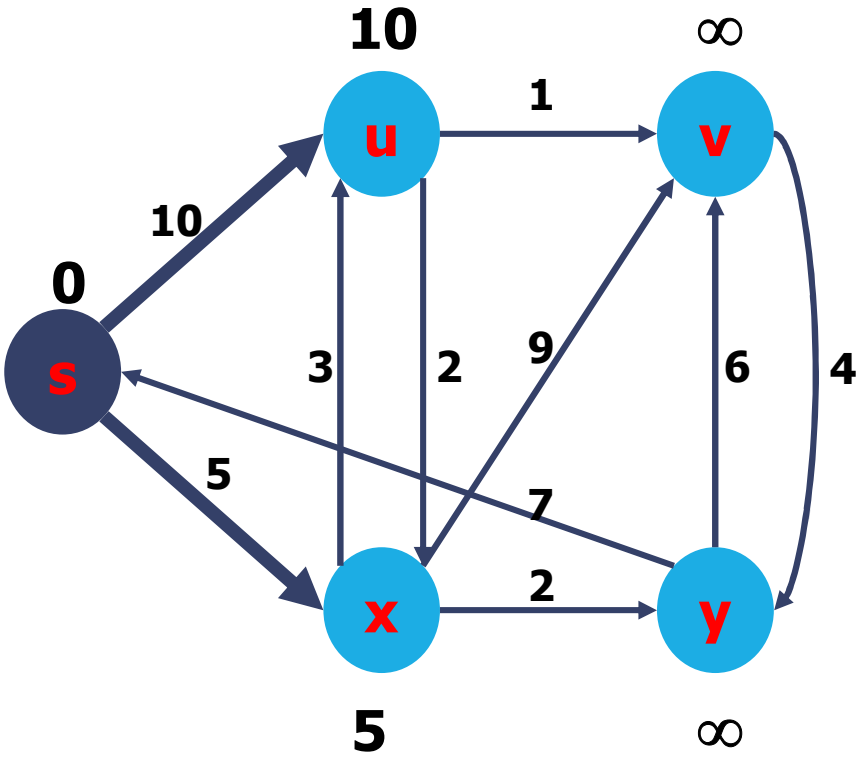
Coda a priorità visualizzata per semplicità come vettore. I nodi compaiono con il loro nome originale per leggibilità

S={}
PQ={s/0, u/∞,v/∞, x/∞, y/∞}



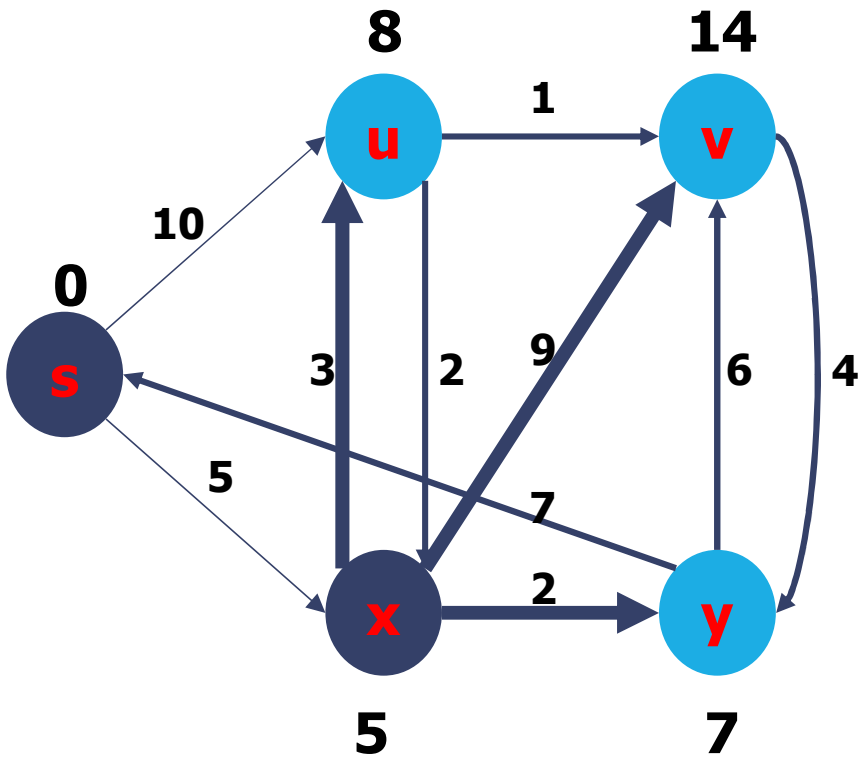
st	0	-1	-1	-1	-1
	0	1	2	3	4

d	0	∞	∞	∞	∞
	0	1	2	3	4



S={s}
relax (s,u), (s,x)
PQ={x/5, u/10,v/∞, y/∞}

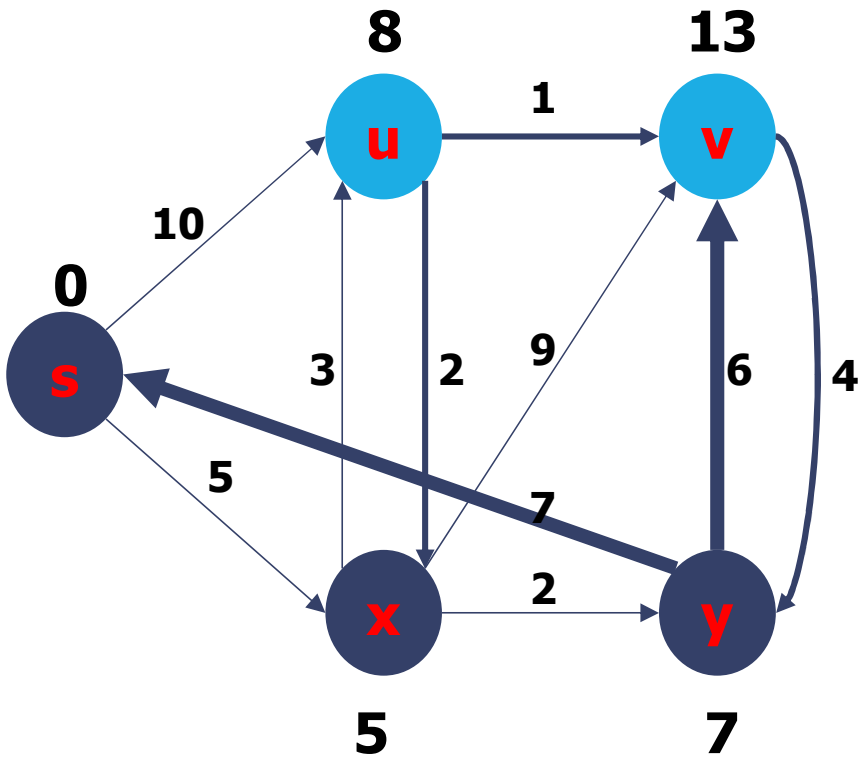
	0	1	2	3	4
st	0	0	0	-1	-1
d	0	10	5	∞	∞



S={s, x}
relax (x,u), (x,v), (x,y)
PQ={y/7, u/8,v/14}

st	0	2	0	2	2
	0	1	2	3	4

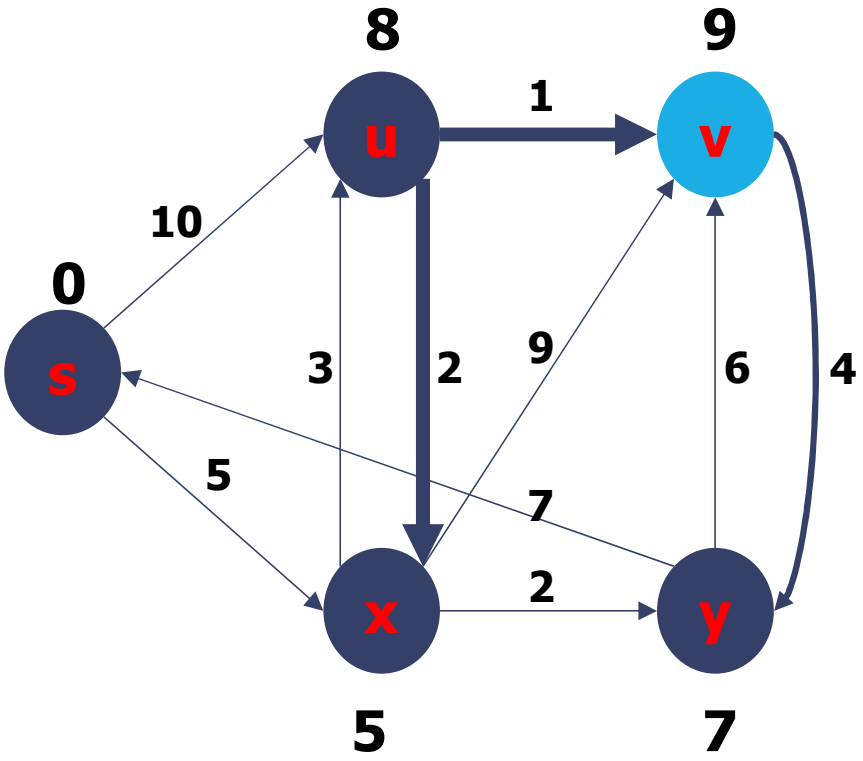
d	0	8	5	14	7
	0	1	2	3	4



S={s, x, y}
relax (y,s), (y,v)
PQ={u/8,v/13}

st	0	2	0	4	2
	0	1	2	3	4

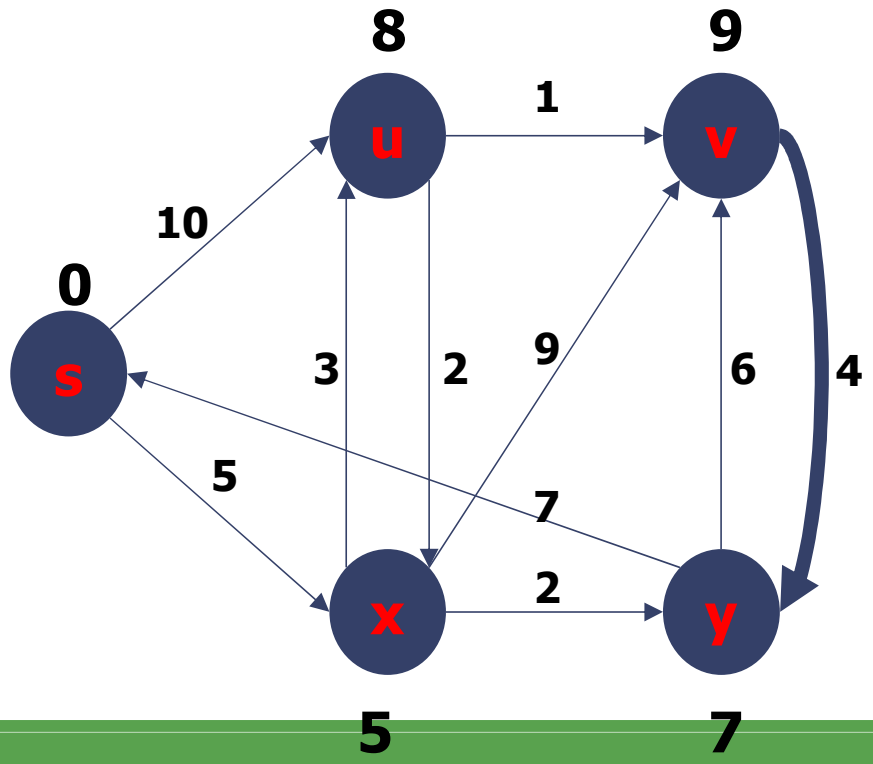
d	0	8	5	13	7
	0	1	2	3	4



S={s, x, y, u}
relax (u,v), (u,x)
PQ={v/9}

st	0	2	0	1	2
	0	1	2	3	4

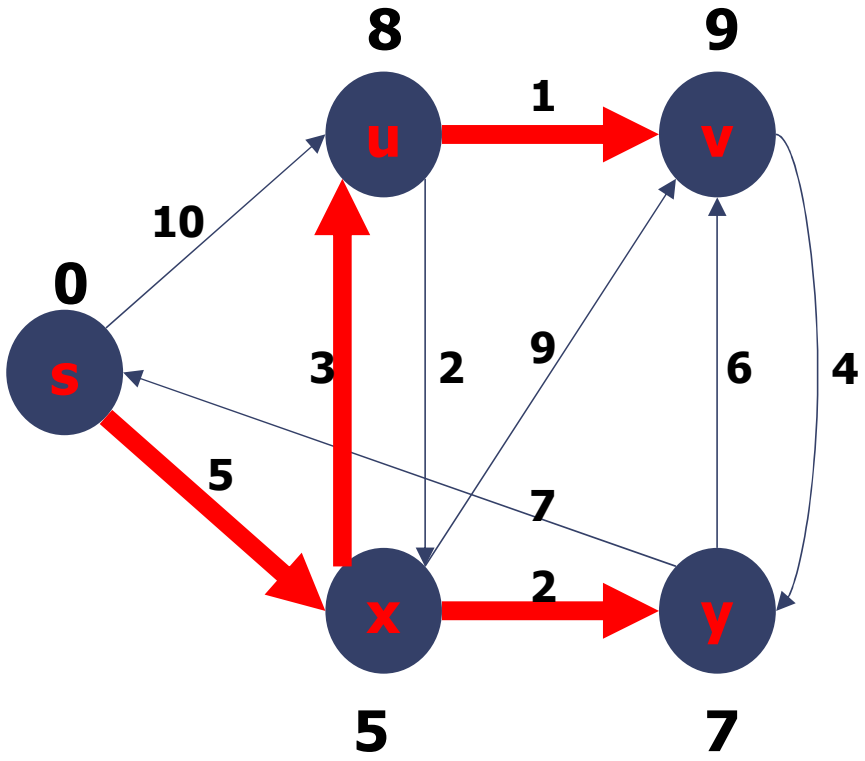
d	0	8	5	9	7
	0	1	2	3	4



S={s, x, y, u, v}
relax (v,y)
PQ={}

st	0	2	0	1	2
	0	1	2	3	4

d	0	8	5	9	7
	0	1	2	3	4



S={s, x, y, u, v}
PQ={}

st	0	2	0	1	2
	0	1	2	3	4

d	0	8	5	9	7
	0	1	2	3	4

```
void GRAPHspD(Graph G, int id) {
  int v;
  link t;
  PQ pq = PQinit(G->V);
  int *st, *d;
  st = malloc(G->V*sizeof(int));
  d = malloc(G->V*sizeof(int));

  for (v = 0; v < G->V; v++){
    st[v] = -1;
    d[v] = maxWT;
    PQinsert(pq, d, v);
  }

  d[id] = 0;
  st[id] = id;
  PQchange(pq, d, id);
```

PQ con priorità in d

```
  while (!PQempty(pq)) {
    if (d[v = PQextractMin(pq, d)] != maxWT)
      for (t=G->ladj[v]; t!=G->z ; t=t->next)
        if (d[v] + t->wt < d[t->v]) {
          d[t->v] = d[v] + t->wt;
          PQchange(pq, d, t->v);
          st[t->v] = v;
        }
  }
  printf("\n Shortest path tree\n");
  for (v = 0; v < G->V; v++)
    printf("parent of %s is %s \n", STsearchByIndex(G->tab, v),
            STsearchByIndex (G->tab, st[v]));
  printf("\n Min.dist. from %s\n", STsearchByIndex(G->tab, s));
  for (v = 0; v < G->V; v++)
    printf("%s: %d\n", STsearchByIndex(G->tab, v), d[v]);
  Pqfree(pq);
}
```

Complessità

Θ(|V|)

V-S: coda a priorità pq dei vertici ancora da stimare. Termina per pq vuota. Implementando la pq con uno heap:

- estrae u da V-S (d[u] minimo)
- inserisce u in S
- rilassa tutti gli archi uscenti da u.

O(lg|V|)

O(lg|V|)

O(|E|)

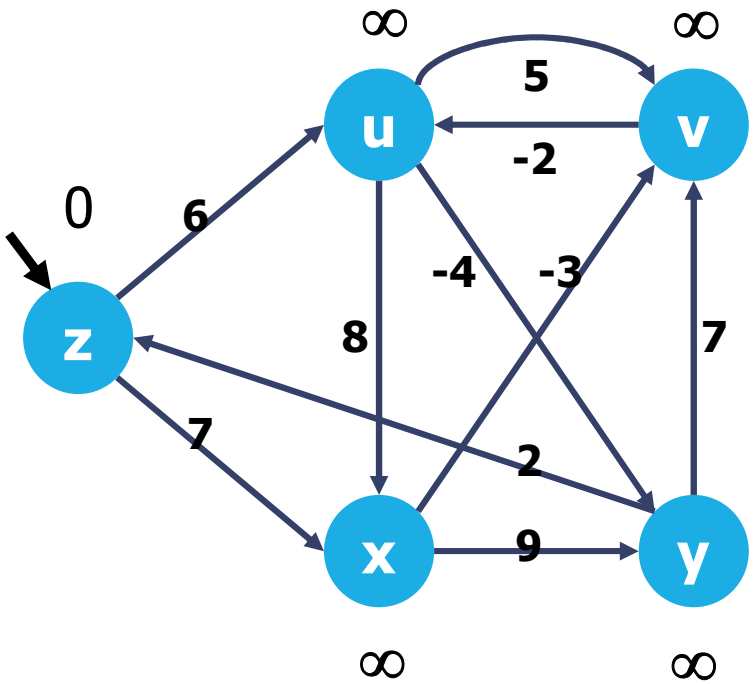
T(n) = O((|V|+|E|) lg |V|)

T(n) = O(|E| lg |V|) se tutti i vertici sono raggiungibili da s

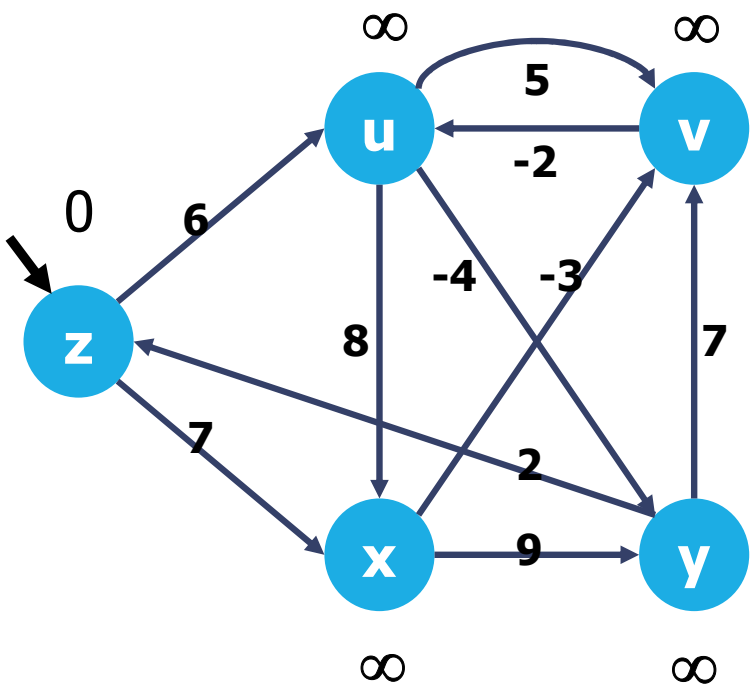
Dijkstra e grafi con archi a peso negativo

$\exists$ archi a peso negativo

$\nexists$ ciclo a peso negativo



ST	
0	z
1	u
2	x
3	v
4	y



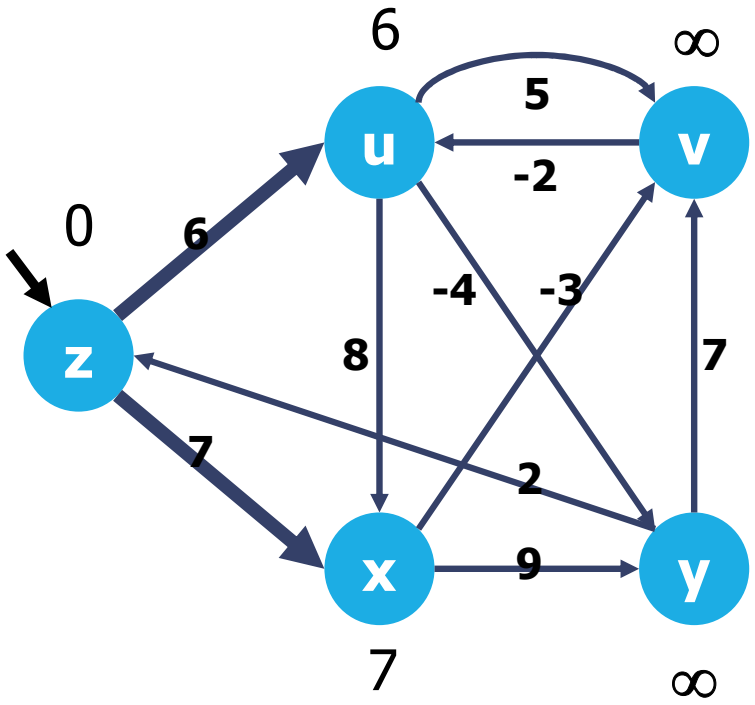
Coda a priorità visualizzata per semplicità come vettore. I nodi compaiono con il loro nome originale per leggibilità.

S={}

PQ={z/0, u/∞, v/∞, x/∞, y/∞}

st	0	-1	-1	-1	-1
	0	1	2	3	4

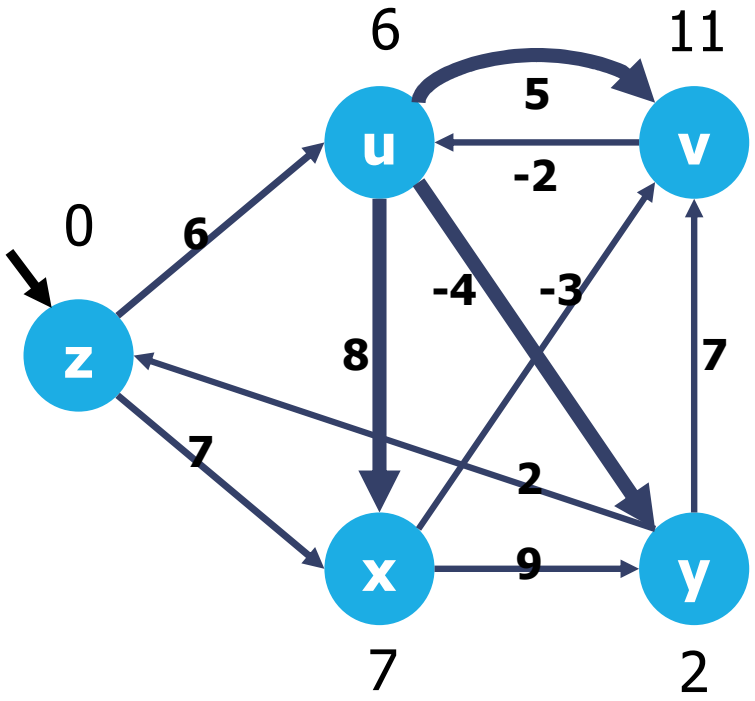
d	0	∞	∞	∞	∞
	0	1	2	3	4



S={z}
relax (z,u), (z,x)
PQ={u/6, x/7, v/∞, y/∞}

st	0	0	0	-1	-1
	0	1	2	3	4

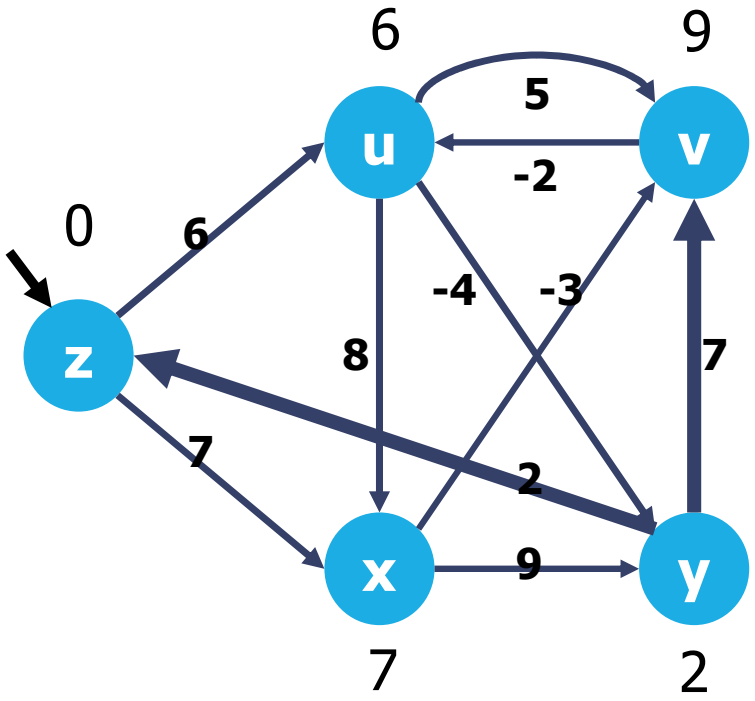
d	0	6	7	∞	∞
	0	1	2	3	4



S={z, u}
relax (u,v), (u,x), (u,y)
PQ={y/2, x/7, v/11}

st	0	0	0	1	1
	0	1	2	3	4

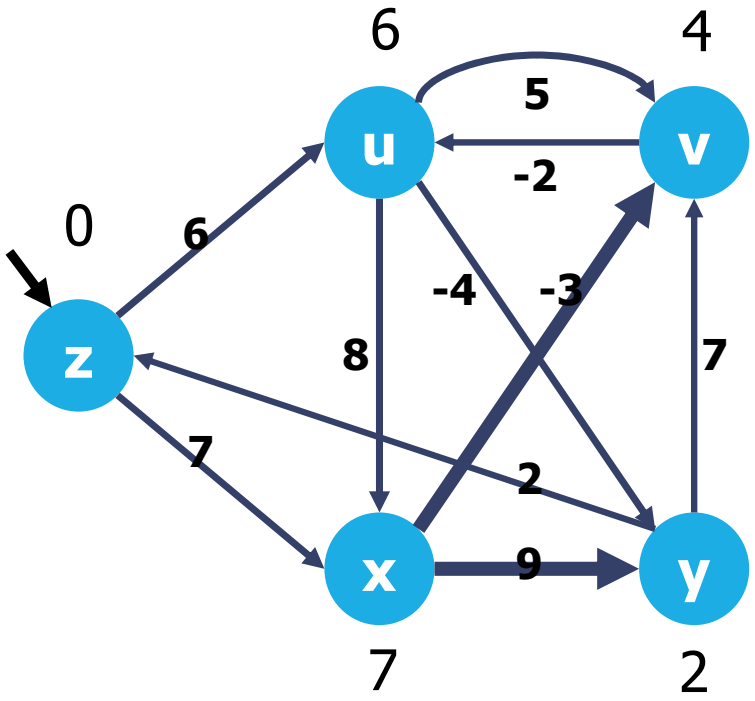
d	0	6	7	11	2
	0	1	2	3	4



S={z, u, y}
relax (y,z), (y,v)
PQ={x/7, v/9}

st	0	0	0	4	1
	0	1	2	3	4

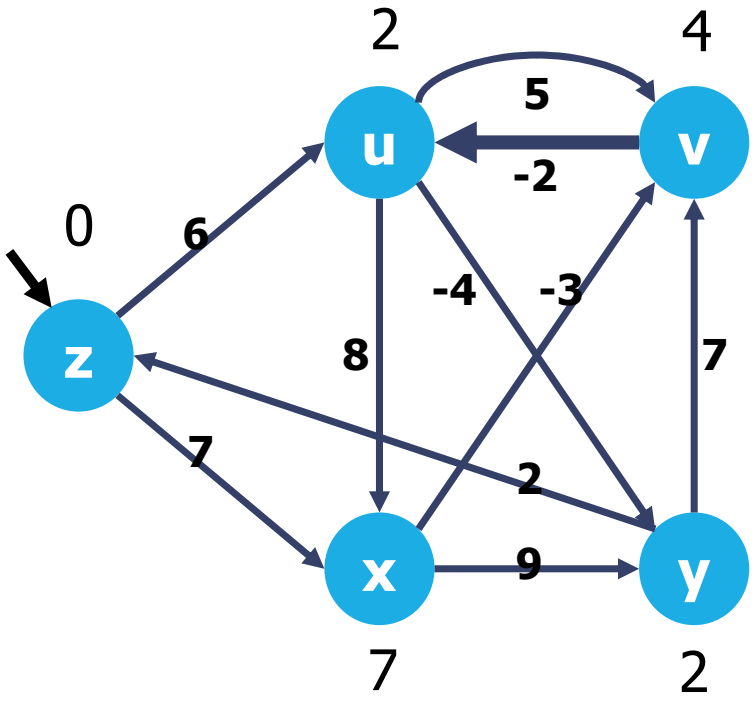
d	0	6	7	9	2
	0	1	2	3	4



S={z, u, y, x}
relax (x,v), (x,y)
PQ={v/4}

st	0	0	0	2	1
	0	1	2	3	4

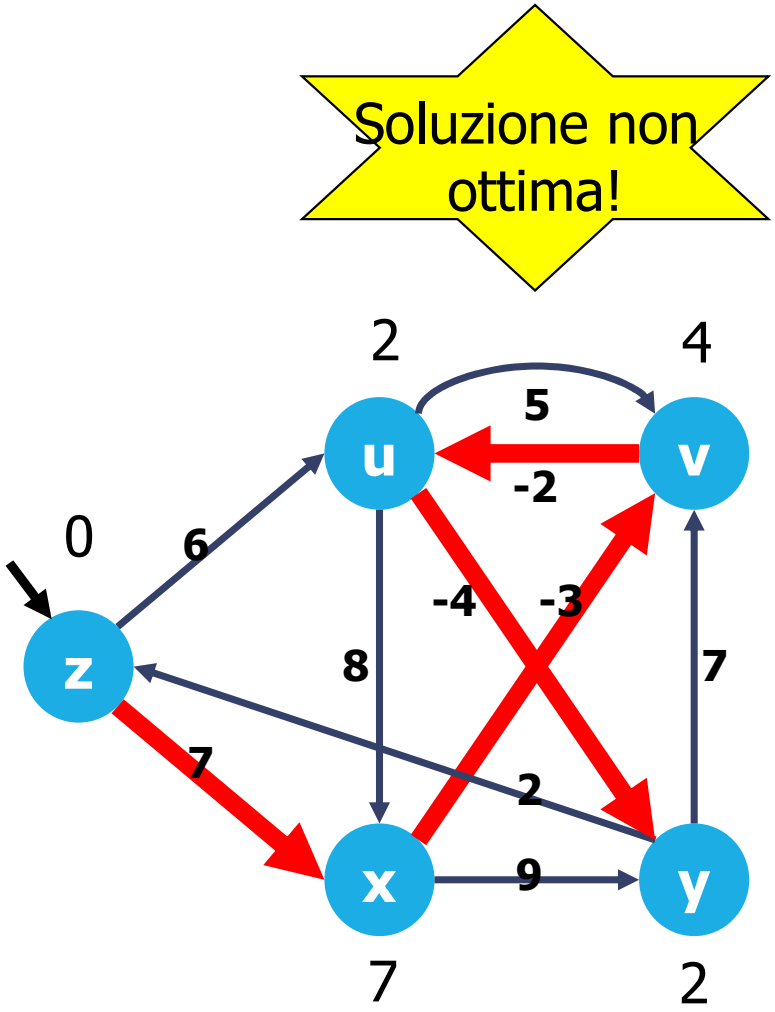
d	0	6	7	4	2
	0	1	2	3	4



S={z, u, x, y, v}
relax (v,u)
PQ={}

st	0	3	0	2	1
	0	1	2	3	4

d	0	2	7	4	2
	0	1	2	3	4



S={z, u, x, y, v}
relax (v,u)
PQ={}

st	0	3	0	2	1
	0	1	2	3	4

d	0	2	7	4	2
	0	1	2	3	4

Se si riconsiderasse l'arco (u,y) la stima di y scenderebbe a -2 (Soluzione ottima).

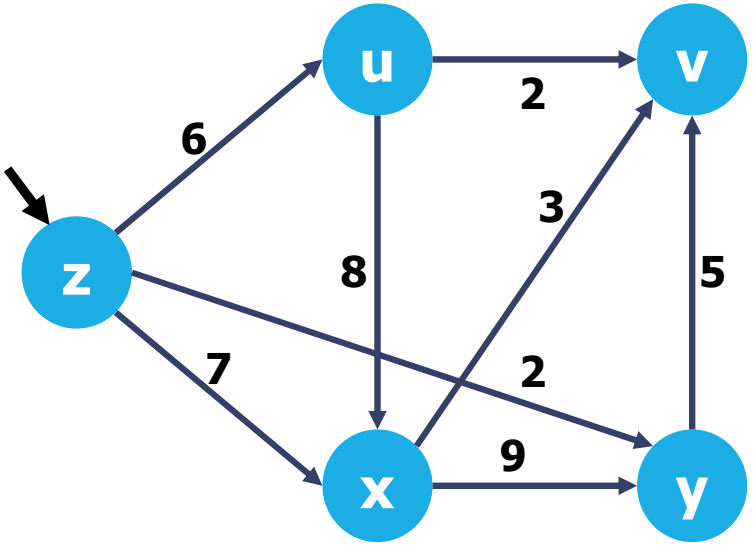
Cammini minimi su DAG pesati

L’assenza di cicli semplifica l’algoritmo:

- ordinamento topologico del DAG
- per tutti i vertici ordinati:
- applica la relaxation da quel vertice.

Esempio

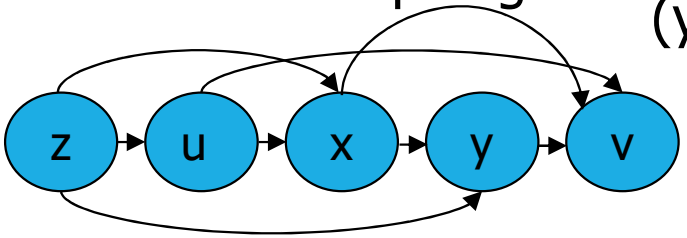
I nodi compaiono con il loro nome originale per leggibilità

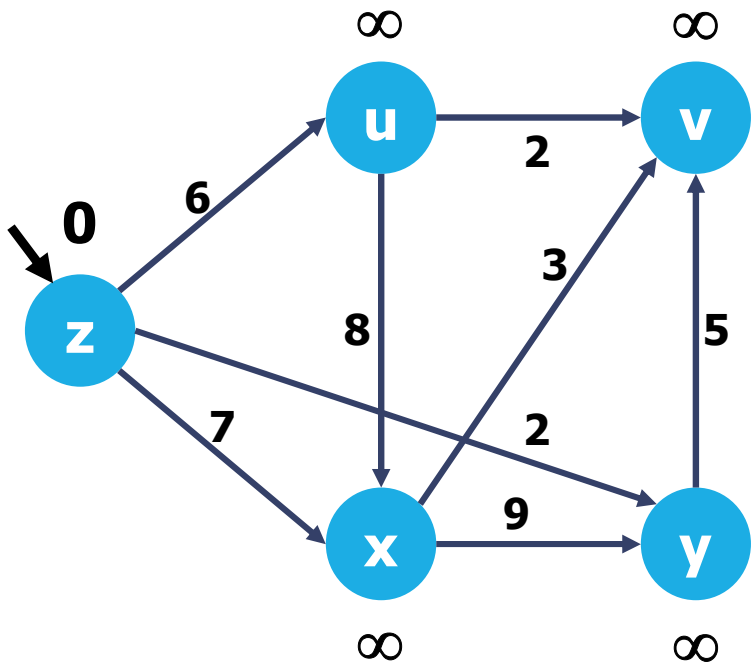


Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v)
(u, x)
(x, v)
(x, y)
(y, v)

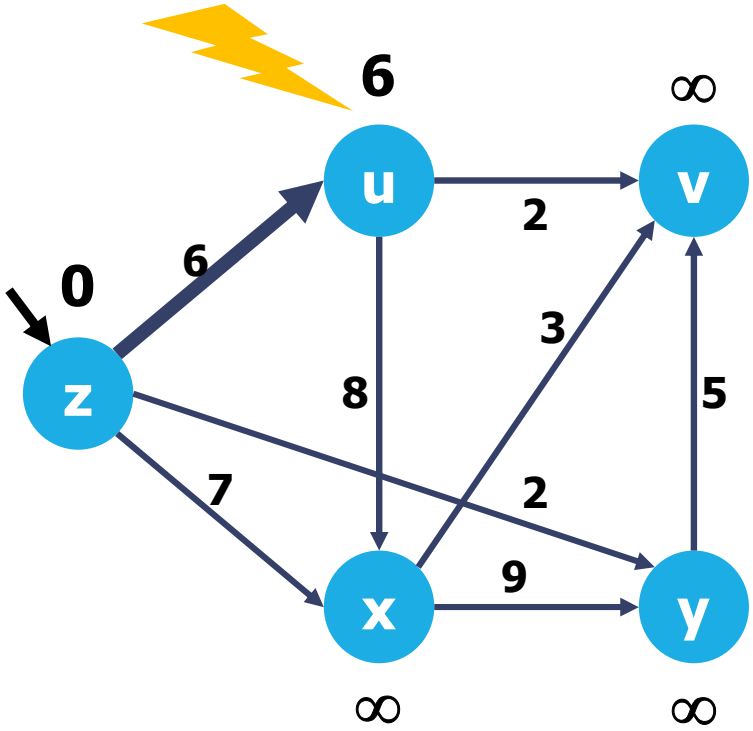
ordinamento topologico





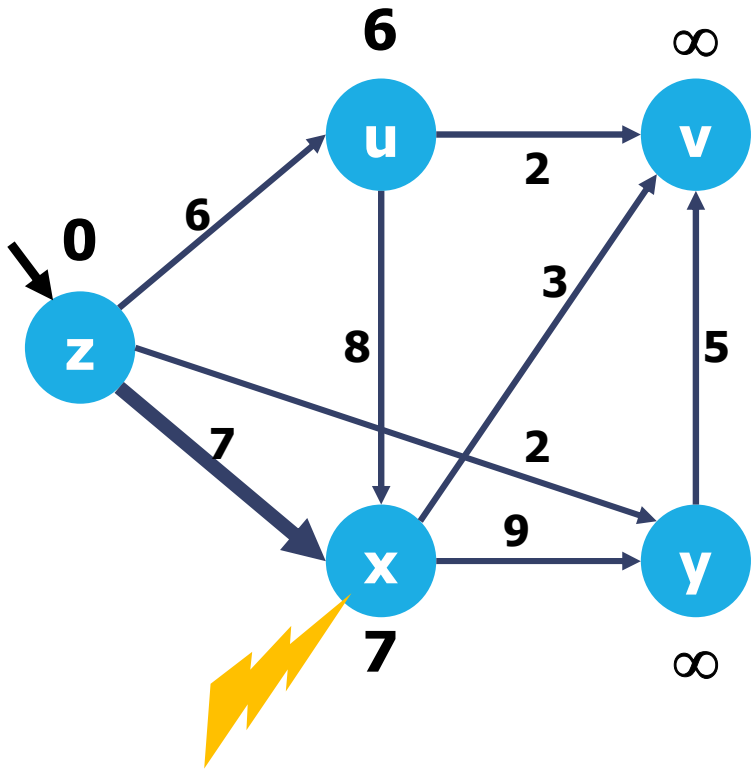
Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v)
(u, x)
(x, v)
(x, y)
(y, v)



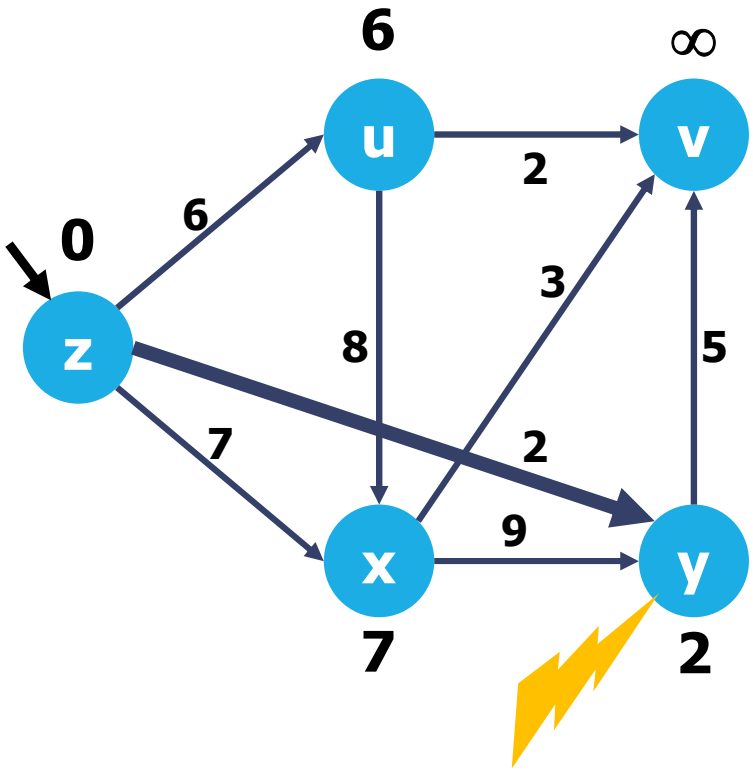
Ordine di applicazione della relaxation

(z, u) ←
(z, x)
(z, y)
(u, v)
(u, x)
(x, v)
(x, y)
(y, v)



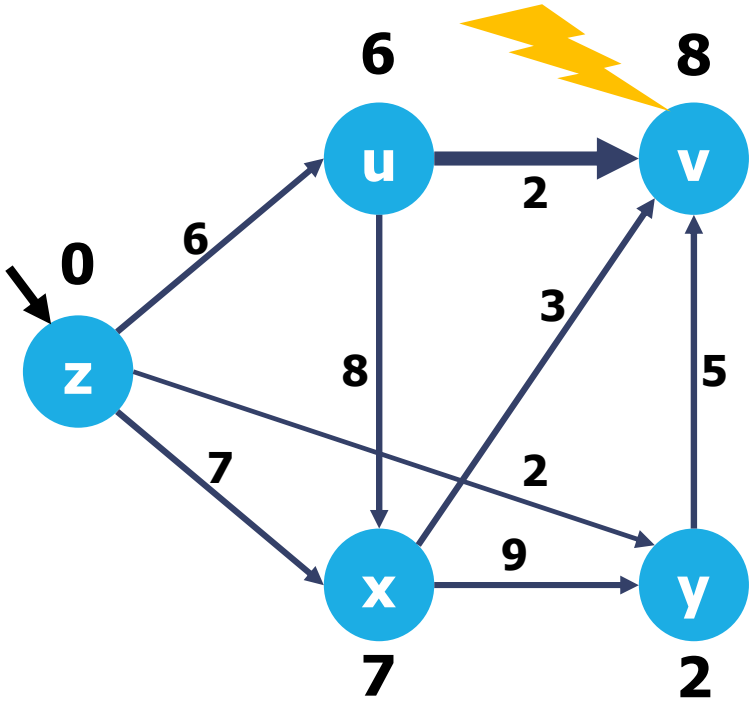
Ordine di applicazione della relaxation

(z, u)
(z, x) ←
(z, y)
(u, v)
(u, x)
(x, v)
(x, y)
(y, v)



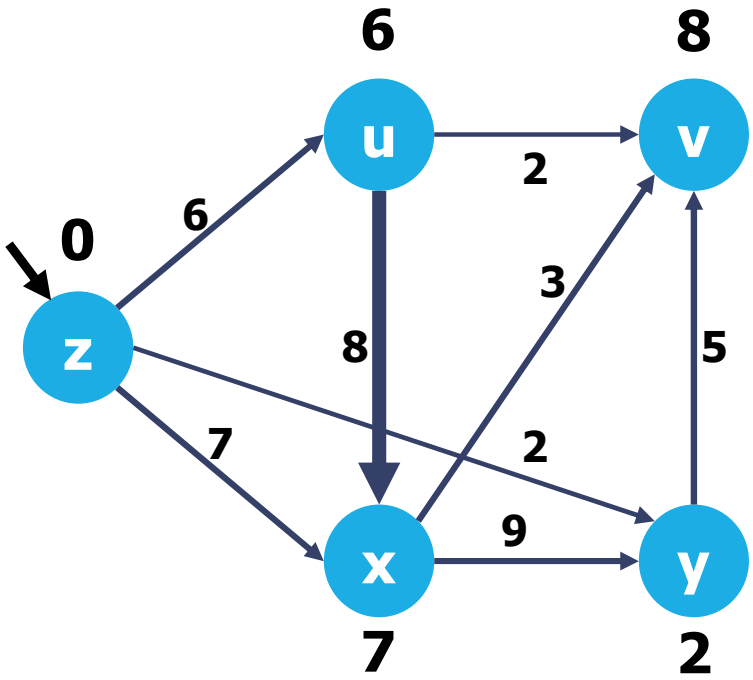
Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y) ←
(u, v)
(u, x)
(x, v)
(x, y)
(y, v)



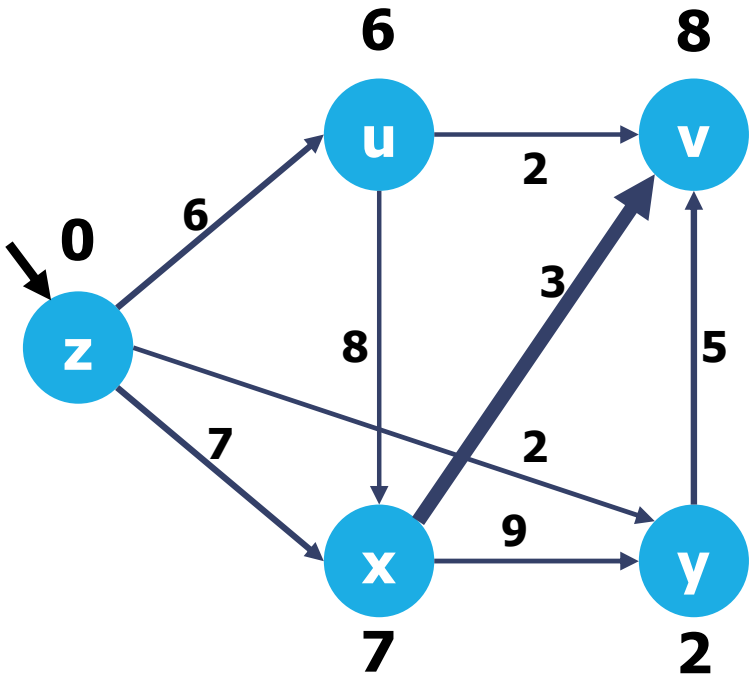
Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v) ←
(u, x)
(x, v)
(x, y)
(y, v)



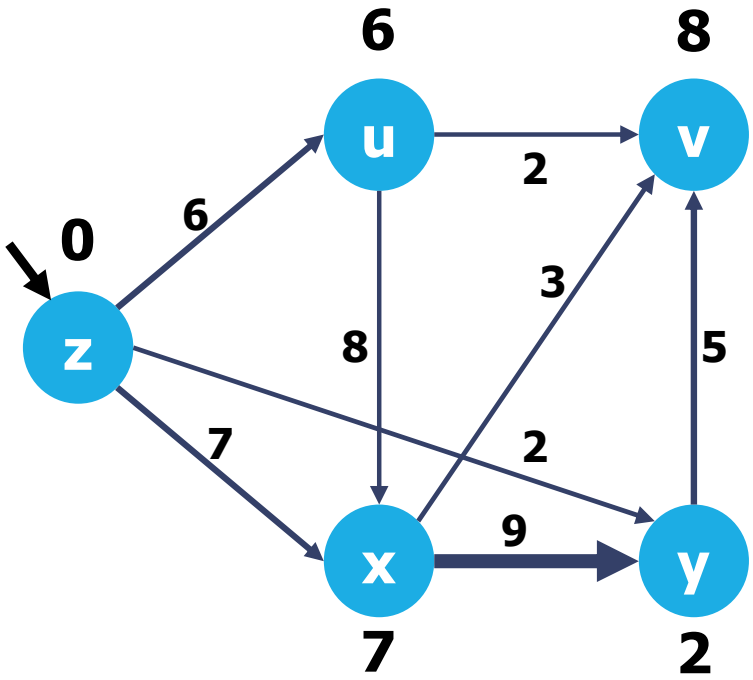
Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v)
(u, x) ←
(x, v)
(x, y)
(y, v)



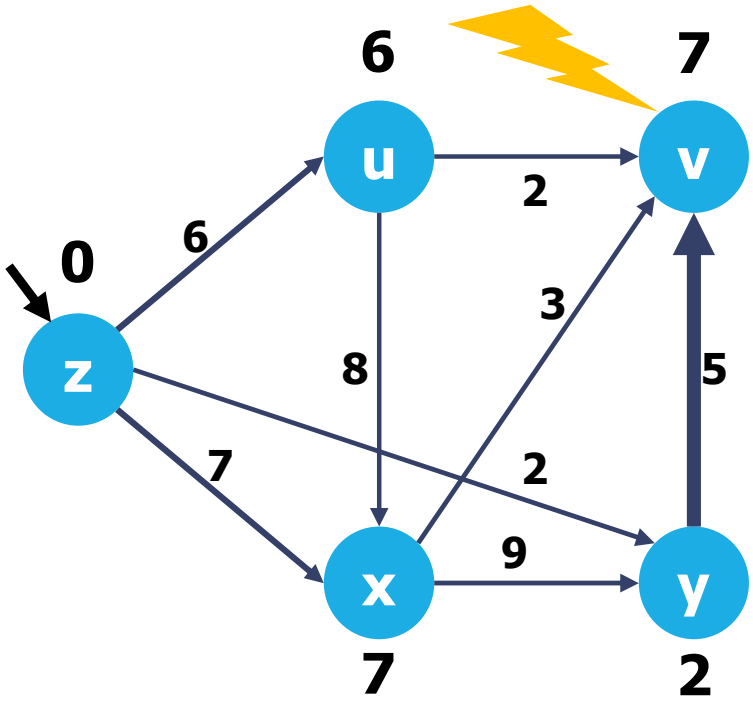
Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v)
(u, x)
(x, v) ←
(x, y)
(y, v)



Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v)
(u, x)
(x, v)
(x, y) ←
(y, v)



Ordine di applicazione della relaxation

(z, u)
(z, x)
(z, y)
(u, v)
(u, x)
(x, v)
(x, y)
(y, v) ←

Complessità

- Applicabile a DAG anche con archi negativi
- T(n) = $O(|V|+|E|)$.

Applicazione: Seam Carving

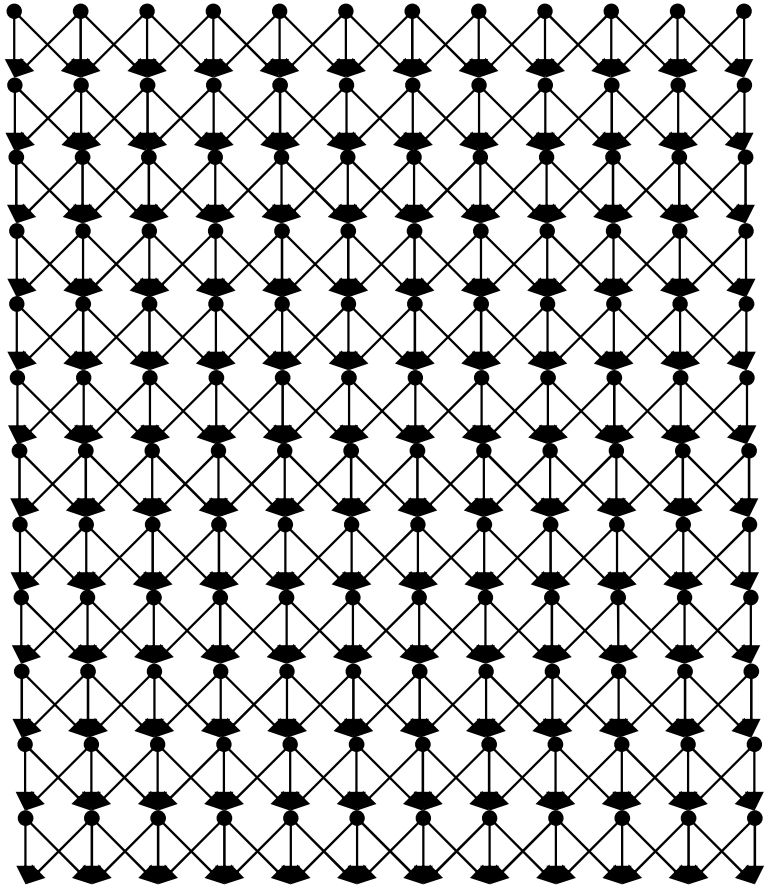
Algoritmo di image resizing per minimizzare la distorsione (Avidan, Shamir).

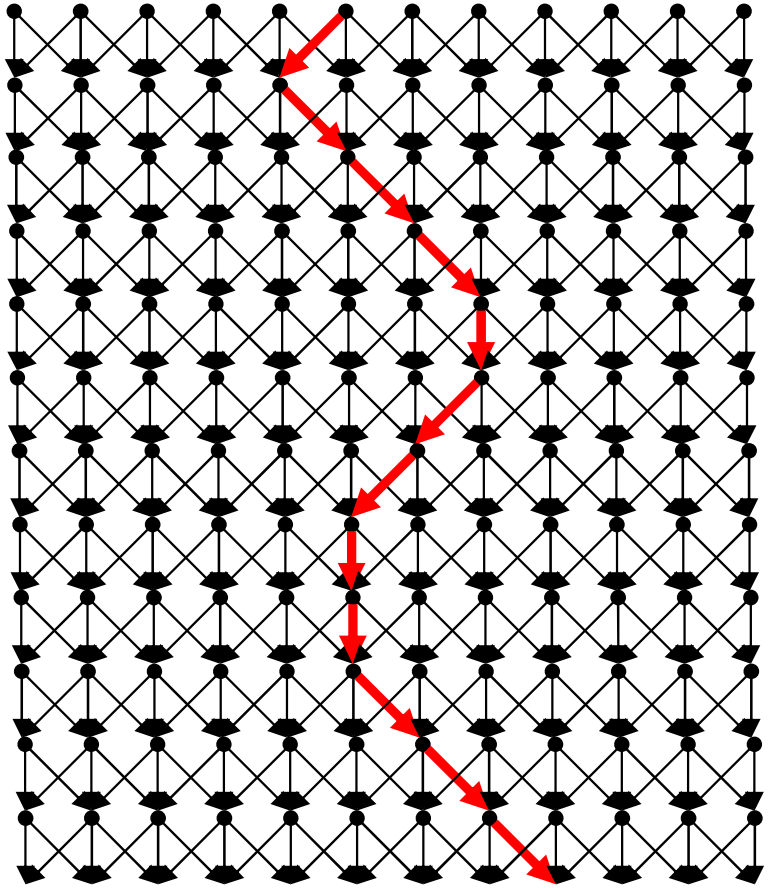
Modello: immagine come DAG pesato di pixel.

Peso dell'arco: misura del contrasto tra 2 pixel.

Algoritmo: determinazione di un cammino minimo da una sorgente (seam), eliminazione dei pixel su di esso.

http://en.wikipedia.org
Sedgewick, Wayne, Algorithms Part I & II, www.coursera.org





seam

Cammini massimi su DAG pesati

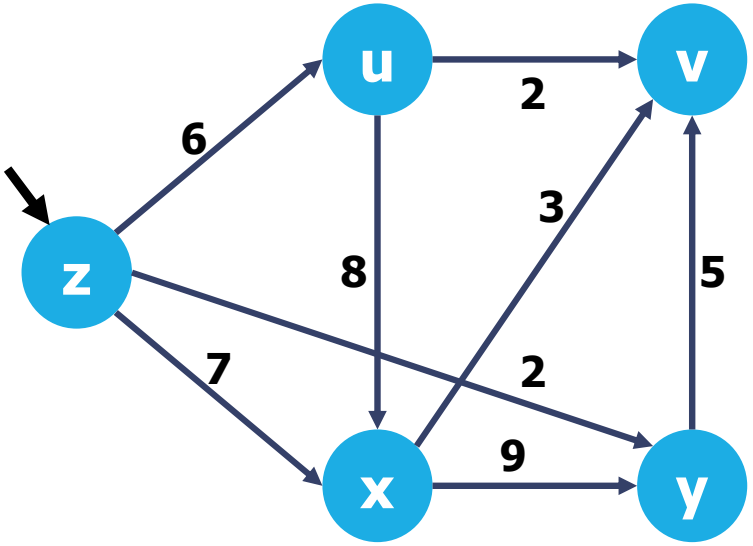
Problema non trattabile su grafi pesati qualsiasi.

L'assenza di cicli tipica dei DAG rende facile il problema:

- ordinamento topologico del DAG
- per tutti i vertici ordinati:
- applica la relaxation «invertita» da quel vertice:

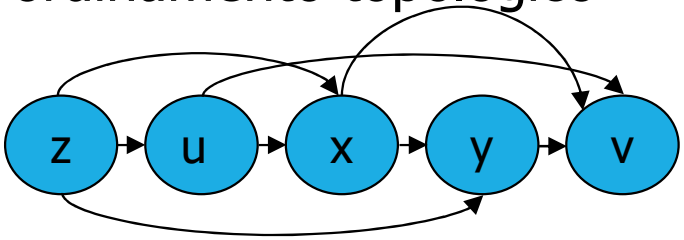
```
if (d[v]< d[u] + w(u,v)) {
  d[v] = d[u] + w(u,v);
  st[v] = u;
}
```

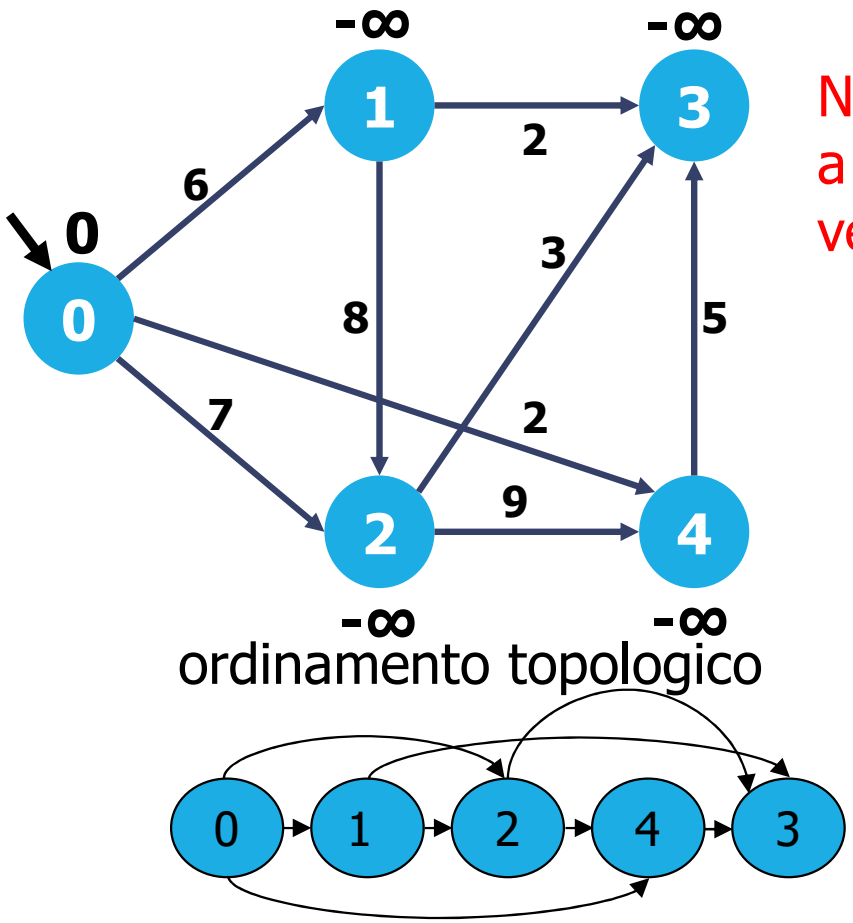
Esempio



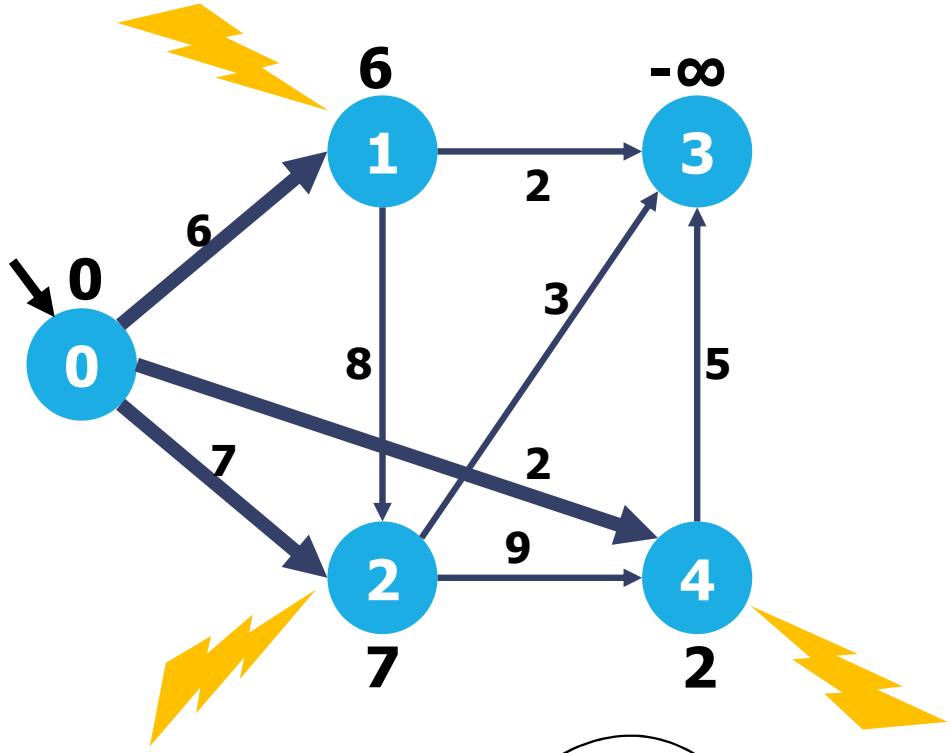
ST	
0	z
1	u
2	x
3	v
4	y

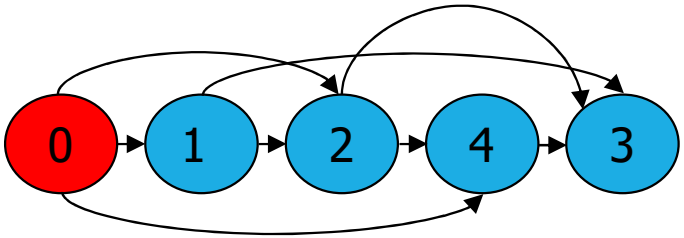
ordinamento topologico

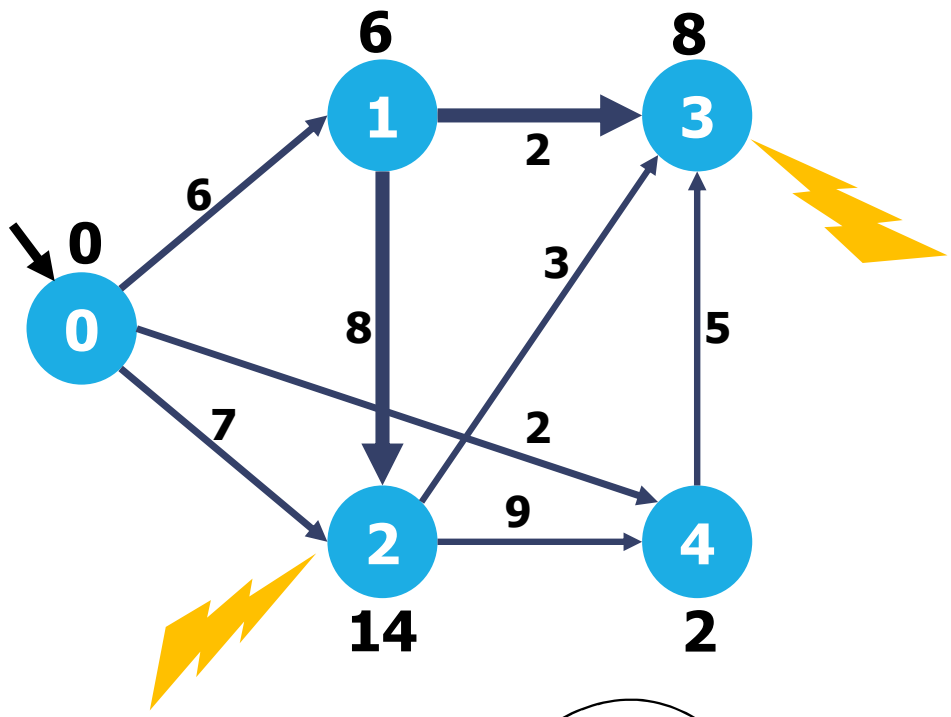


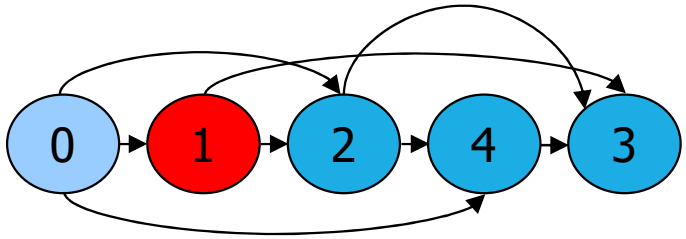


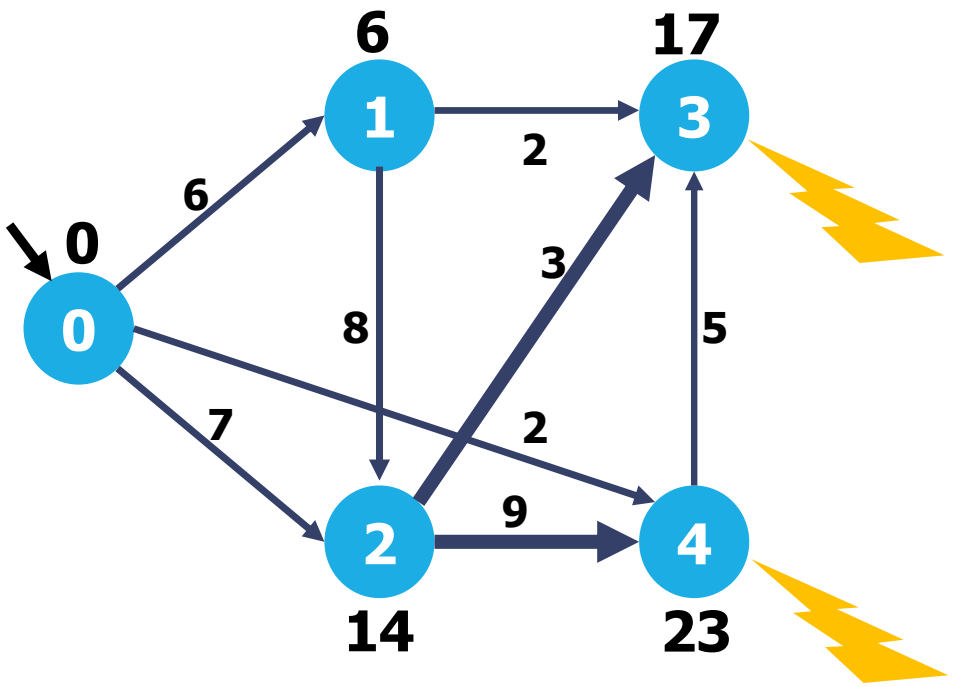
Notare le stime iniziali a -∞ tranne che del vertice di partenza

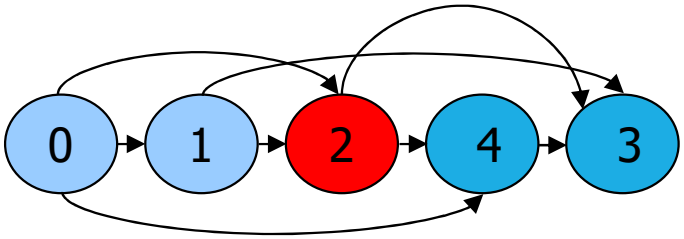

6
-∞
1
3
2
6
0
0
3
8
5
7
2
9
4
7
2

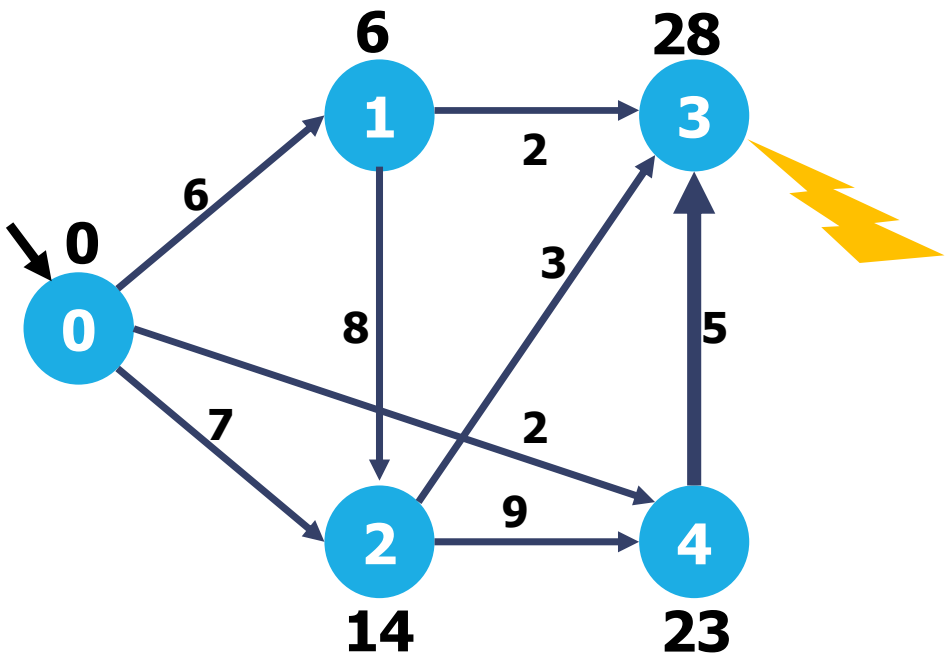

0
1
2
4
3

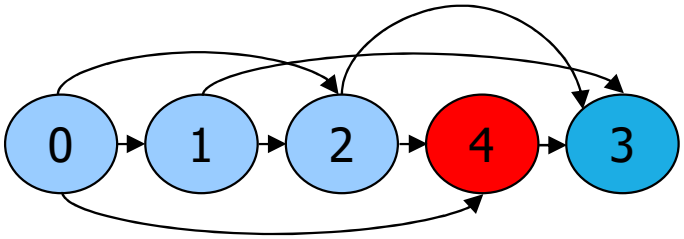

6
8
1
3
2
6
0
0
3
8
5
7
2
9
2
4
14
2

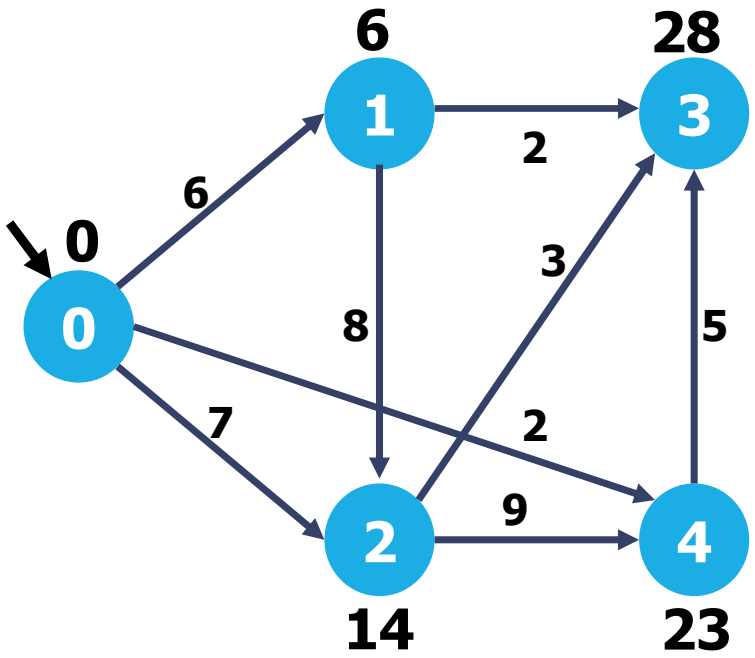

0
1
2
4
3

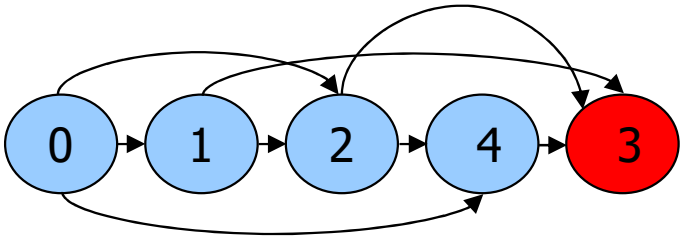

6
17
1
3
2
6
0
0
3
8
5
7
2
2
9
4
14
23


0
1
2
4
3


6
28
1
3
2
6
0
0
3
8
5
7
2
2
9
4
14
23


0
1
2
4
3


6
28
1
3
2
6
0
0
3
8
5
7
2
2
9
4
14
23


0
1
2
4
3

Algoritmo di Bellman-Ford

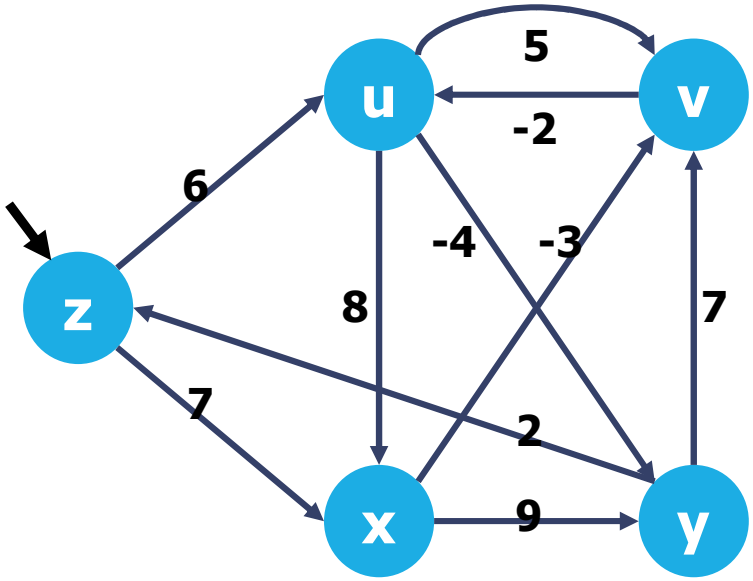
- Ipotesi: possono $\exists$ archi a peso < 0
- Rileva cicli < 0
- Strategia: programmazione dinamica (applicabilità già dimostrata)
- soluzione ricorsiva: $d_w[v]$ è la lunghezza del cammino minimo da s a v con al più w archi
  - $d_0[v]$ vale 0 se v coincide con s, ∞ altrimenti
  - $d_w[v] = \min\{d_w[v], \min_{u \in Inc(v)}(d_{w-1}[u] + w(u,v))\}$
  - Inc(v): insieme dei vertici da cui escono archi incidenti in v

Calcolo bottom-up:

- vettori `d` e `st` delle distanze minime e dei predecessori opportunamente inizializzati
- |V|-1 passi
- al passo i-esimo:
  - ciclo di rilassamento sugli archi «in avanti» (lista di adiacenza e non di incidenza)
- al |V|-esimo passo:
  - diminuisce almeno una stima: ∃ ciclo <0
  - altrimenti soluzione ottima.

Il numero di passi è al massimo |V|-1: se ci si accorge di aver raggiunto un punto fisso (non cambiano le stime da un passo a quello successivo), si può interrompere anticipatamente il ciclo.

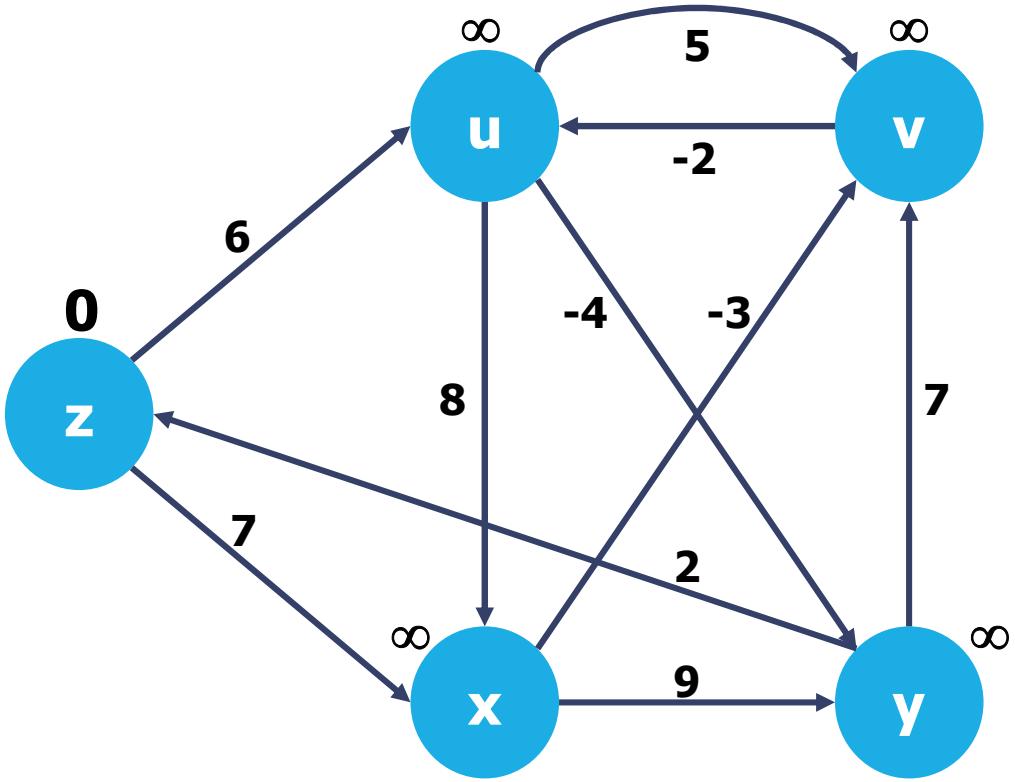
Esempio



ST	
0	z
1	u
2	x
3	v
4	y

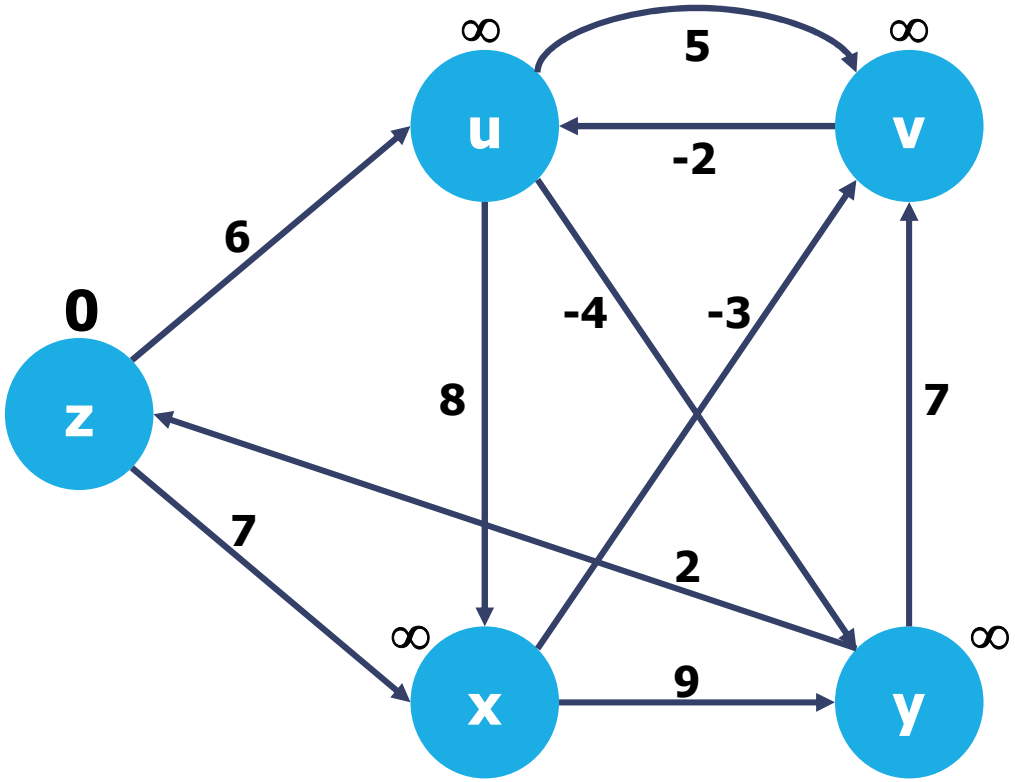
Archi in ordine lessicografico:
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

I nodi compaiono con il loro nome originale per leggibilità



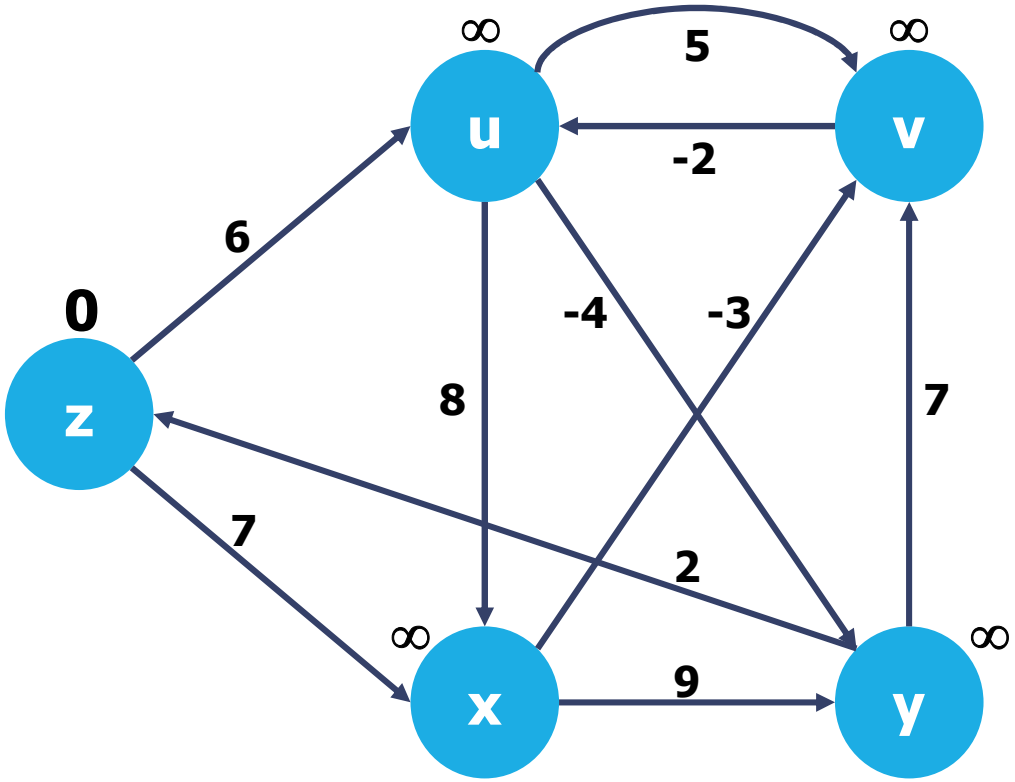
Passo 1

→ **(u,v)**
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



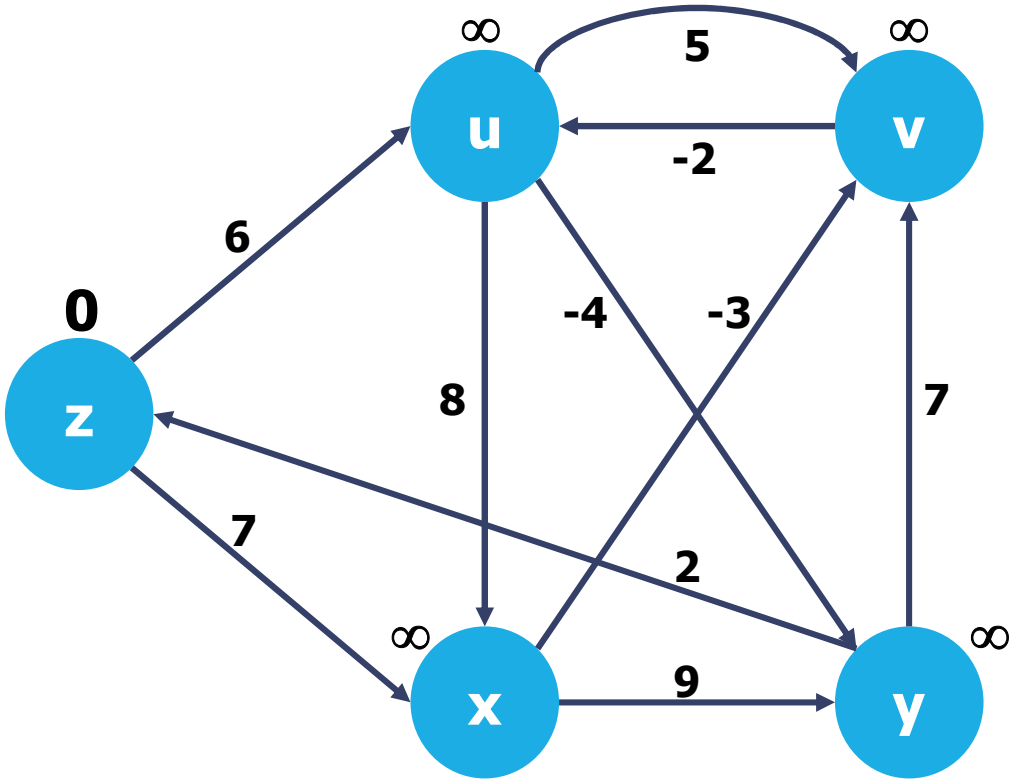
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



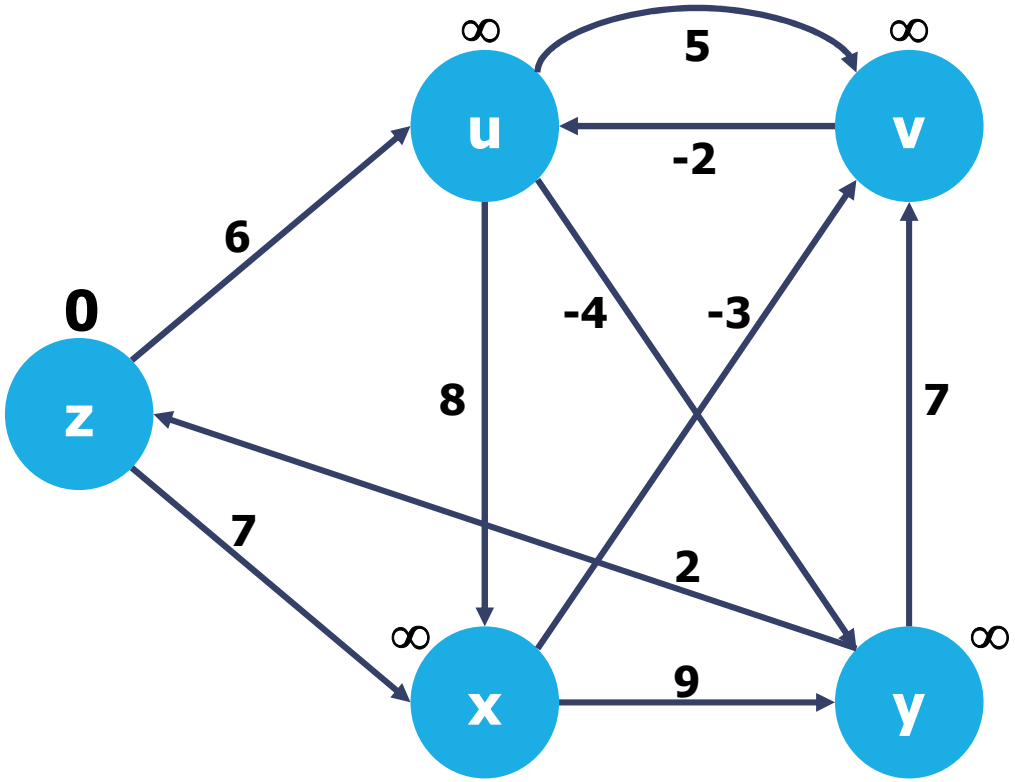
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



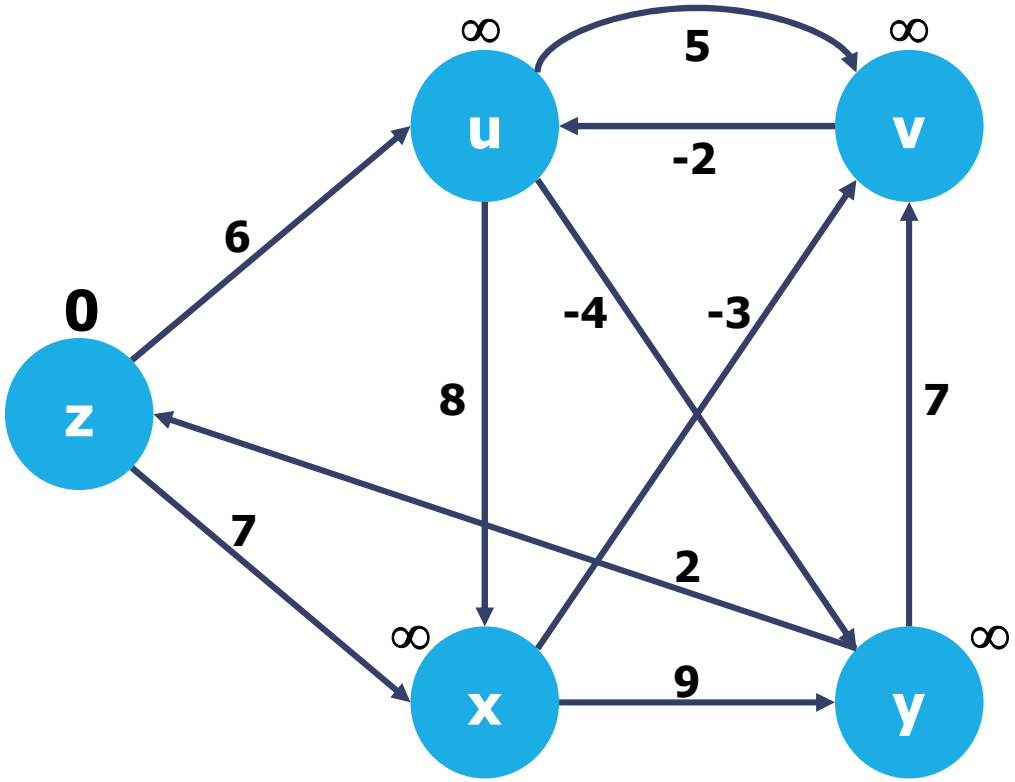
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



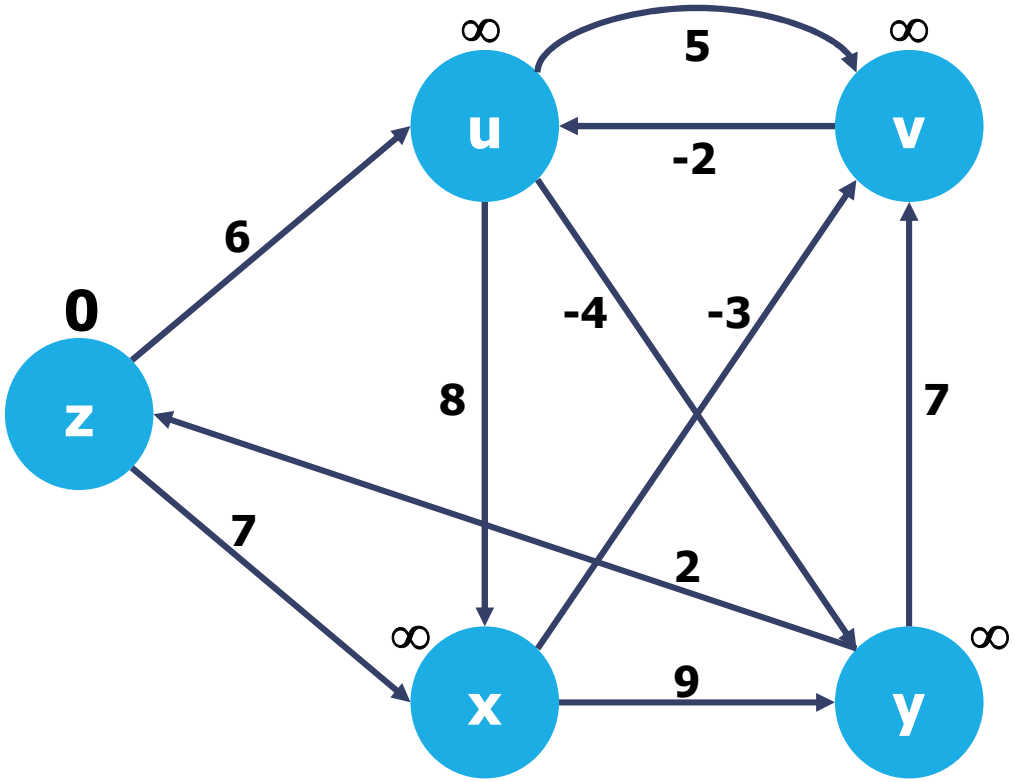
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



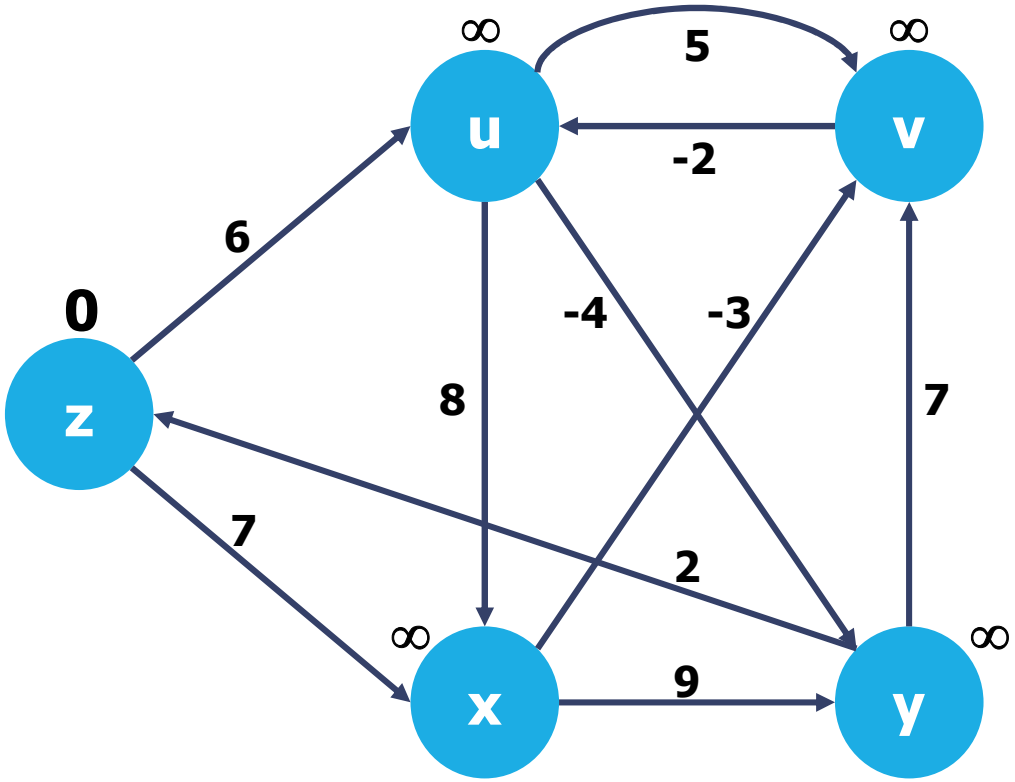
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



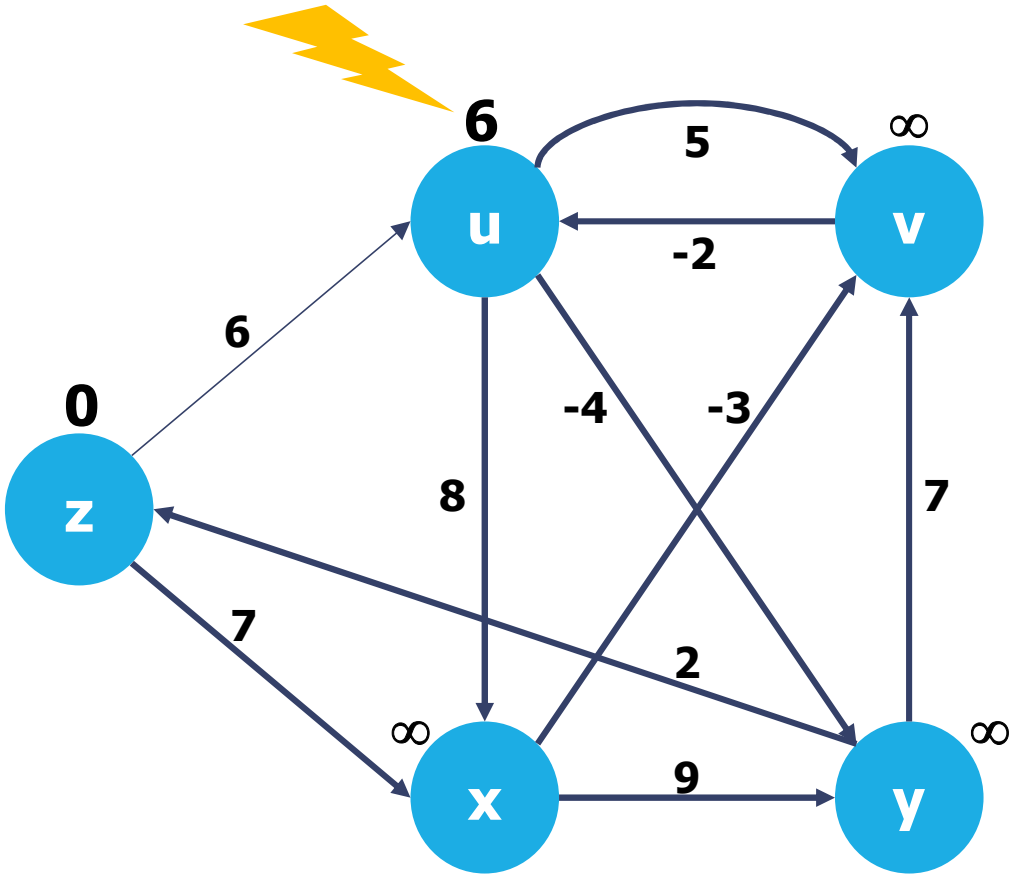
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



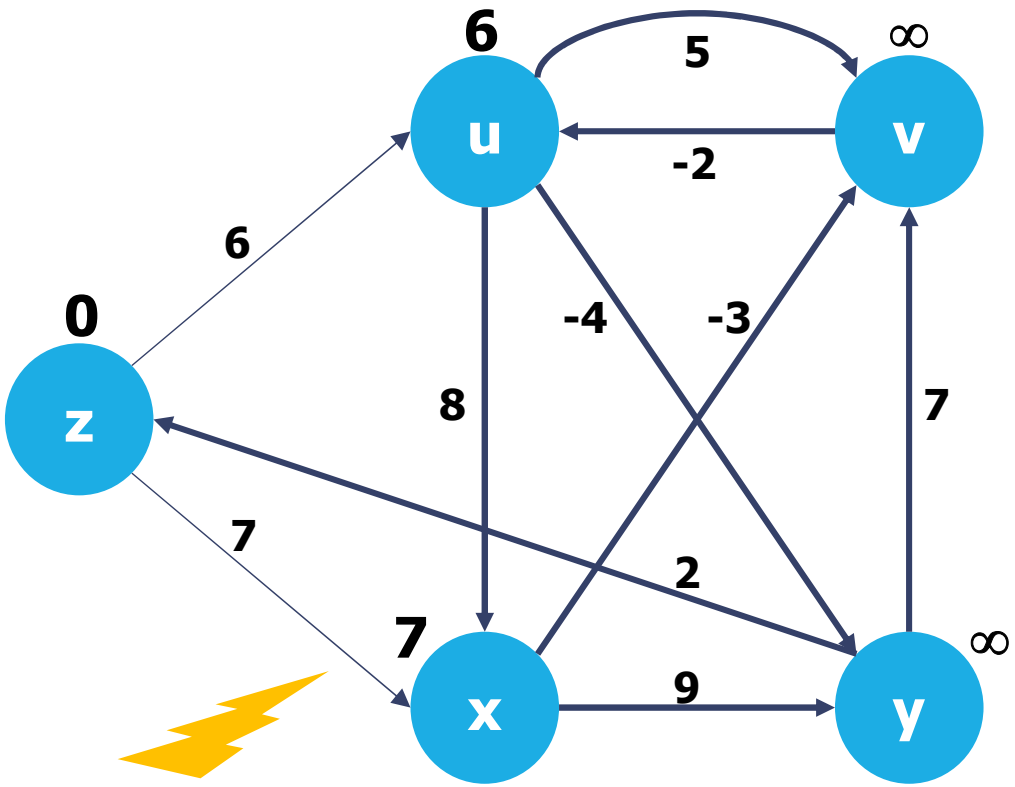
Passo 1

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



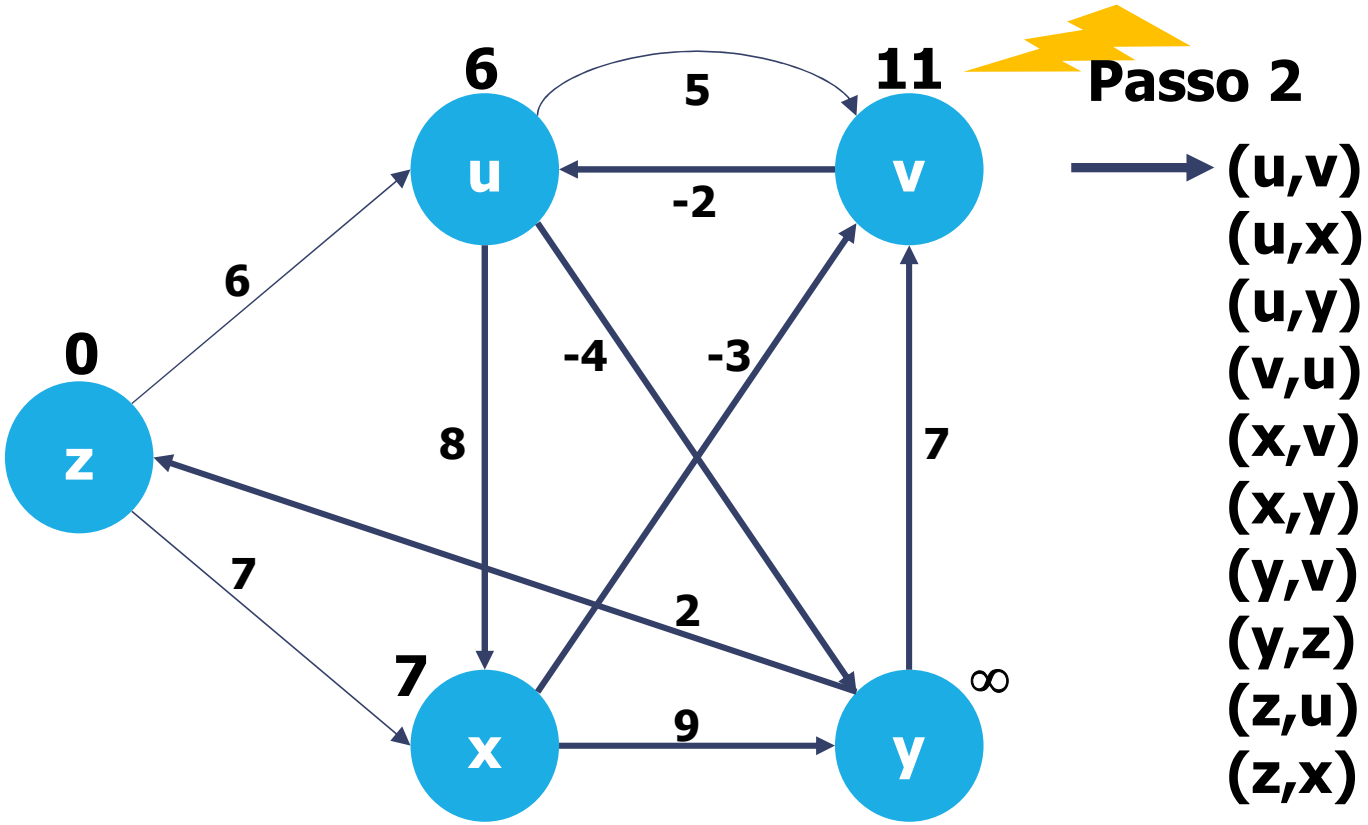
Passo 1

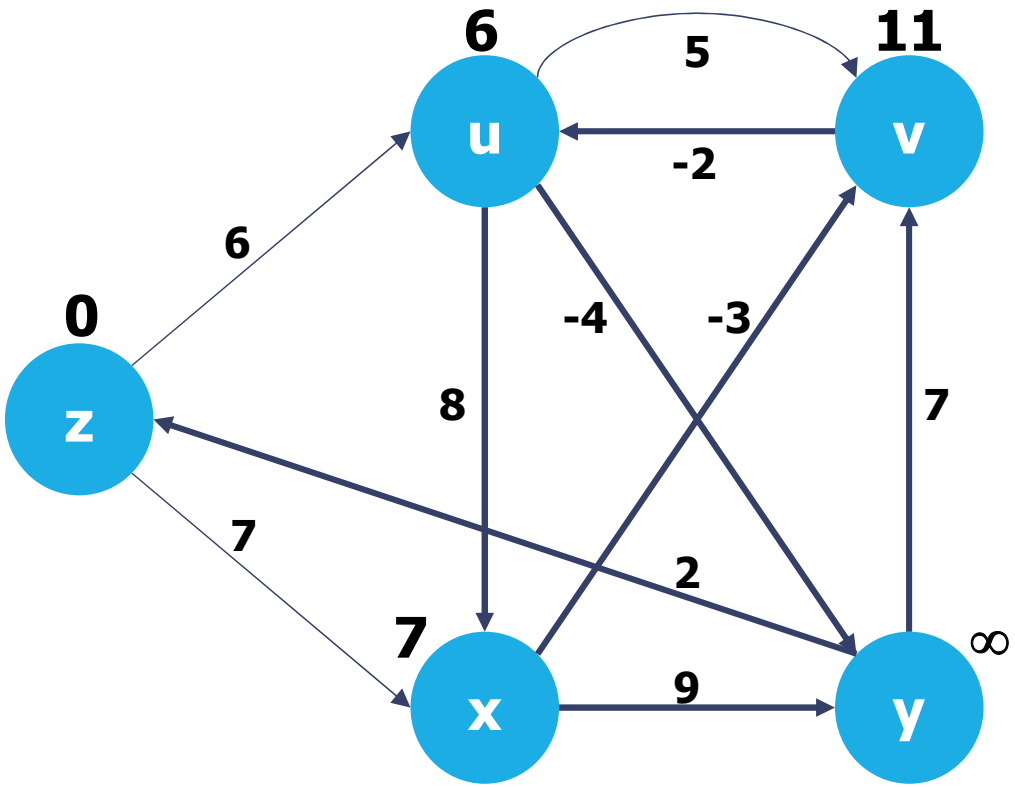
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 1

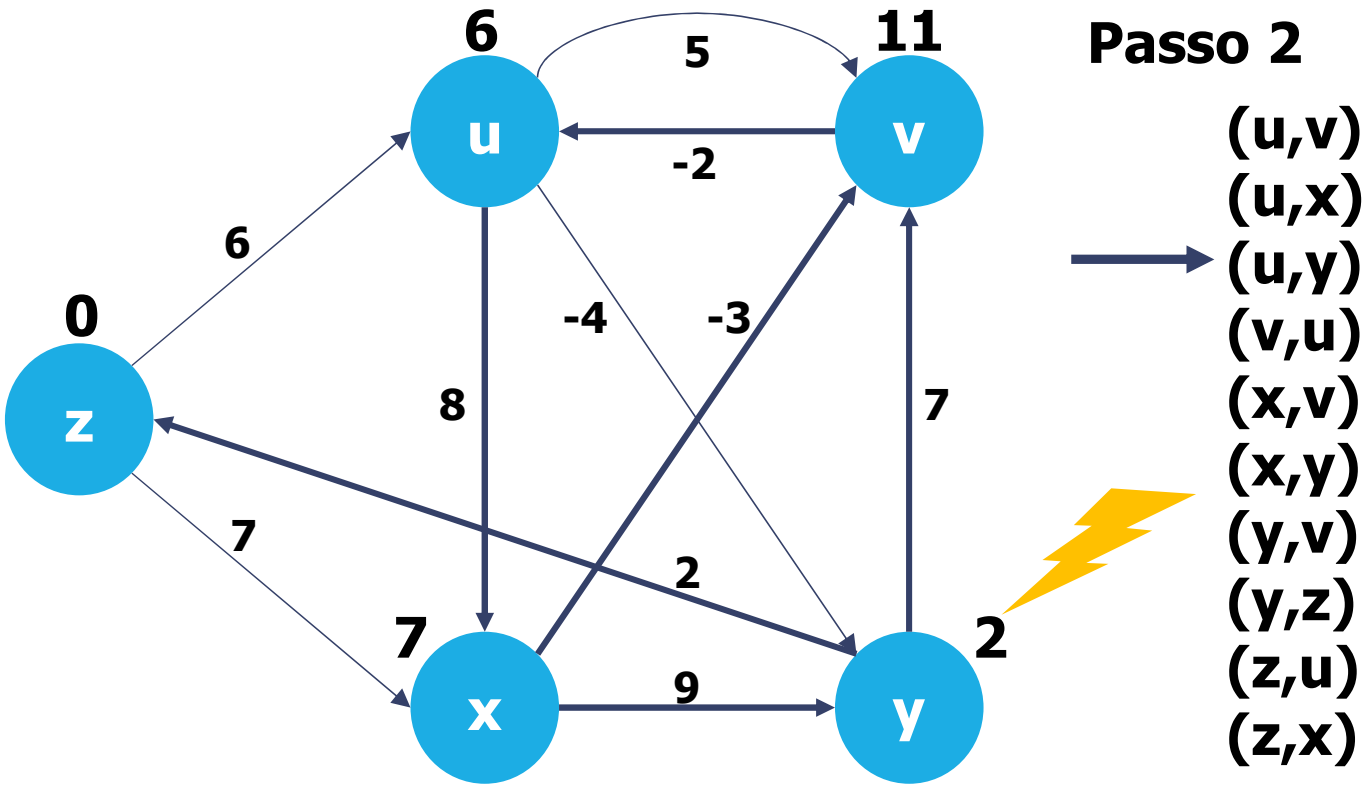
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

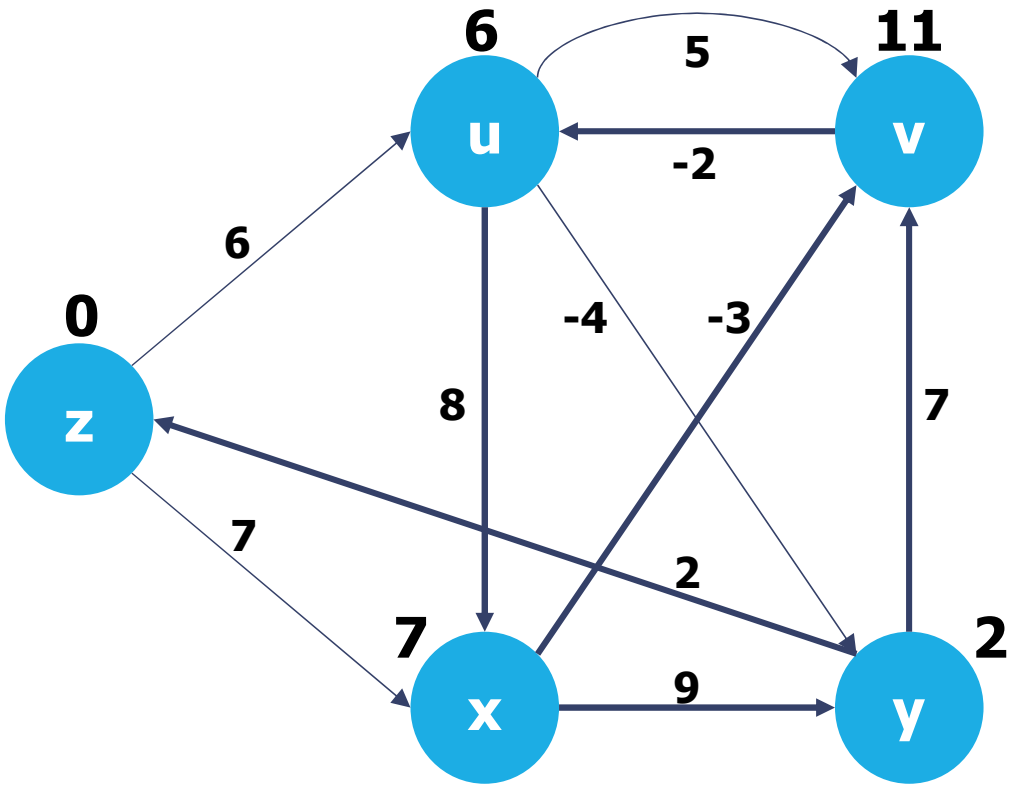

6
11
Passo 2
5
u
v
-2
6
0
-4
-3
z
8
7
7
2
∞
9
x
y
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 2

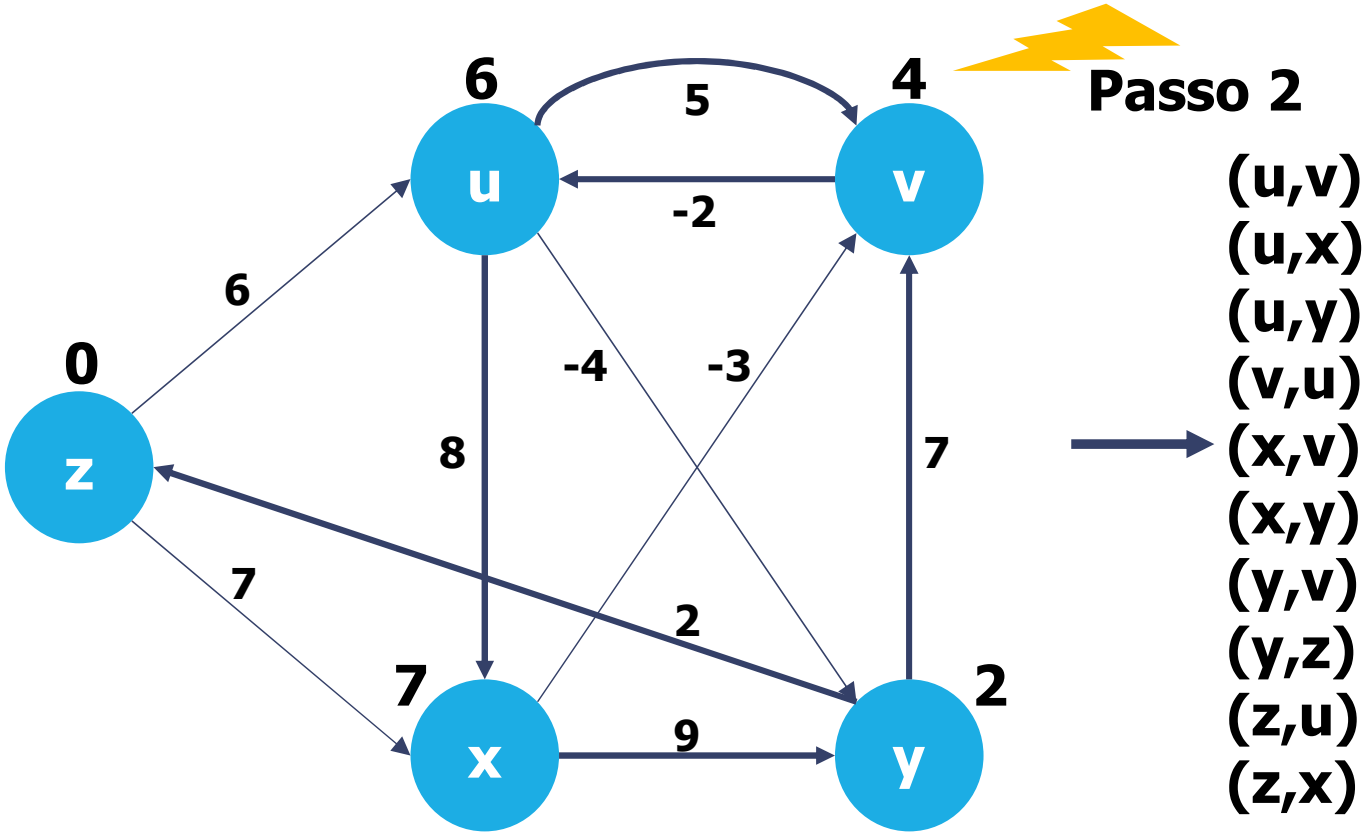
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

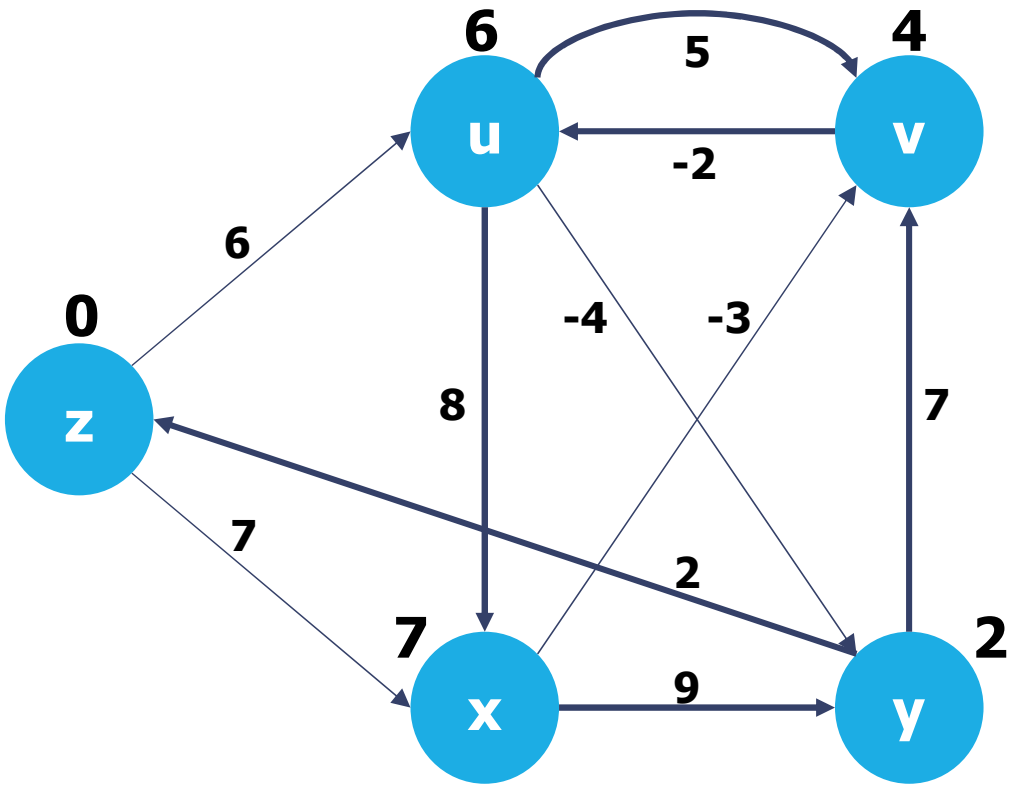

6
11
u
v
5
-2
6
0
z
-4
-3
8
7
7
2
7
x
9
y
2
Passo 2
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 2

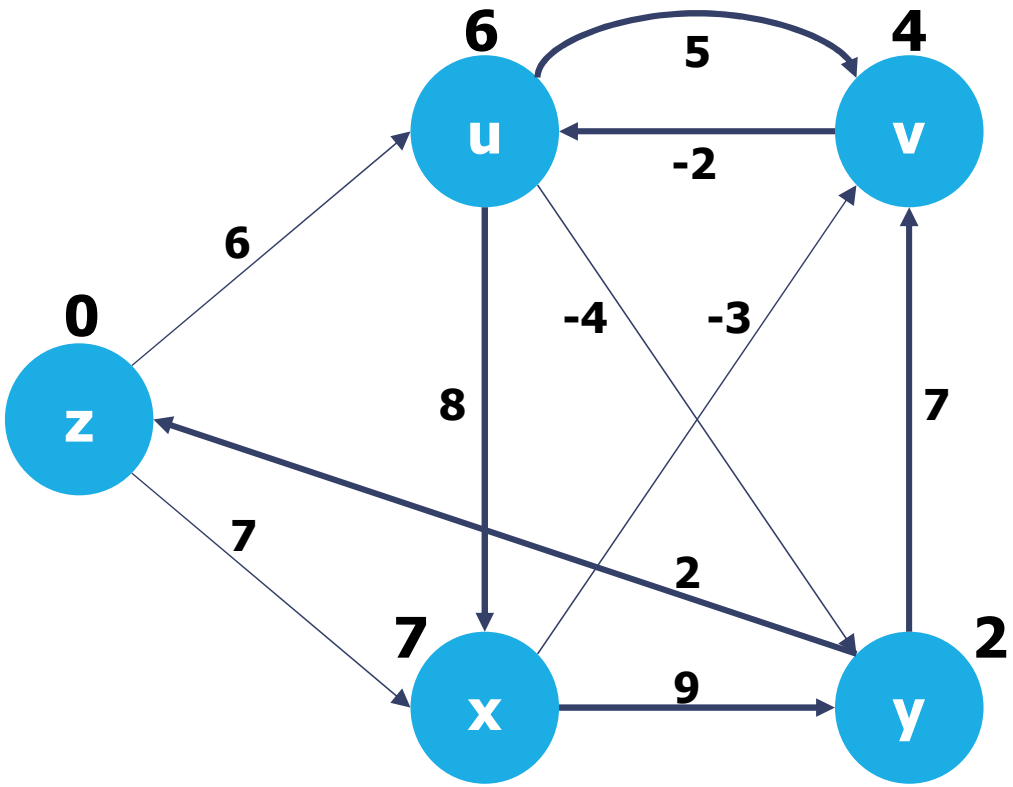
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)


Passo 2
6
4
u
v
5
-2
6
0
-4
-3
z
8
7
7
2
7
2
x
9
y
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



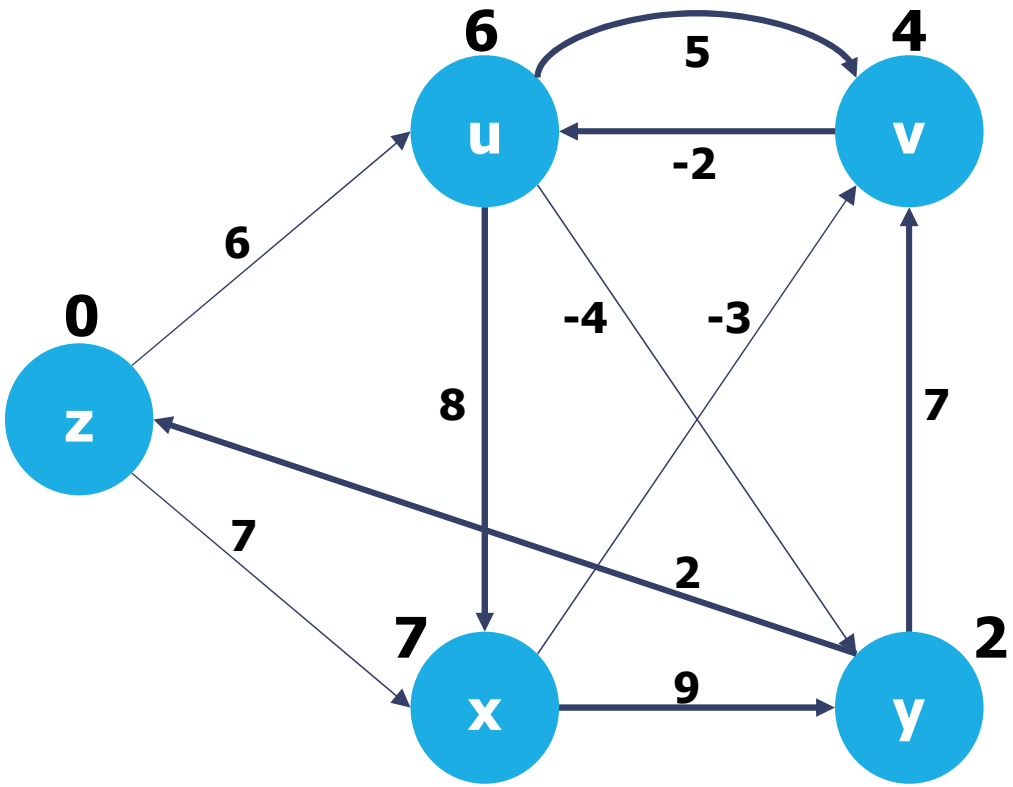
Passo 2

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



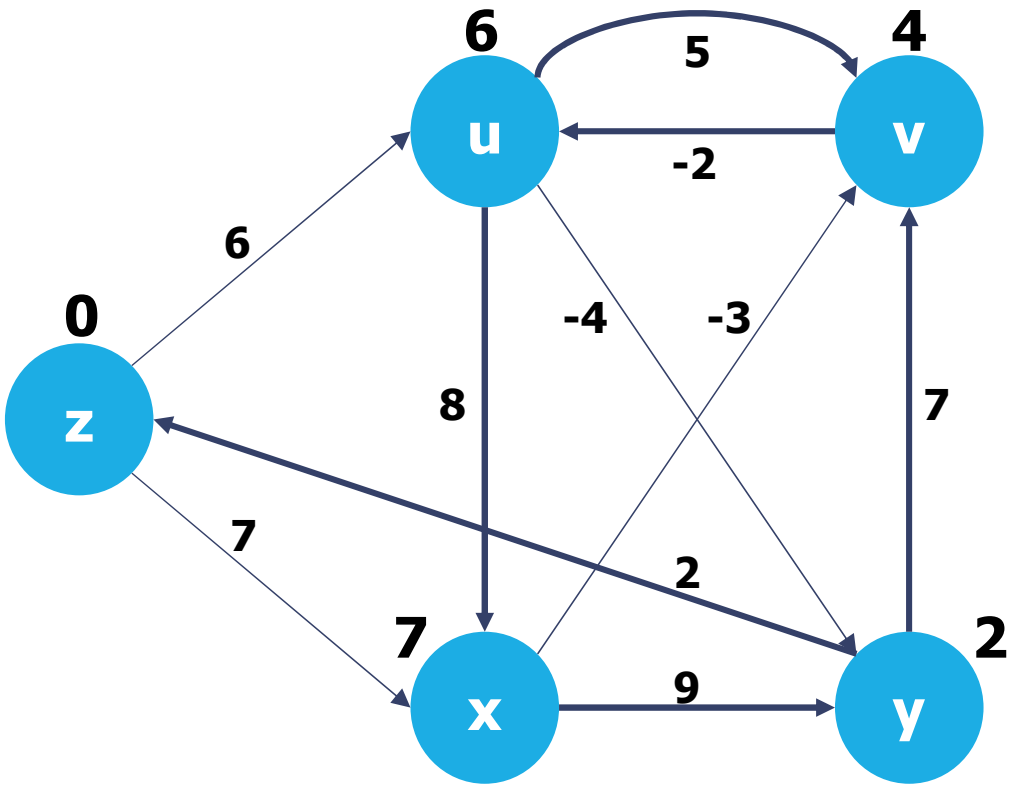
Passo 2

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



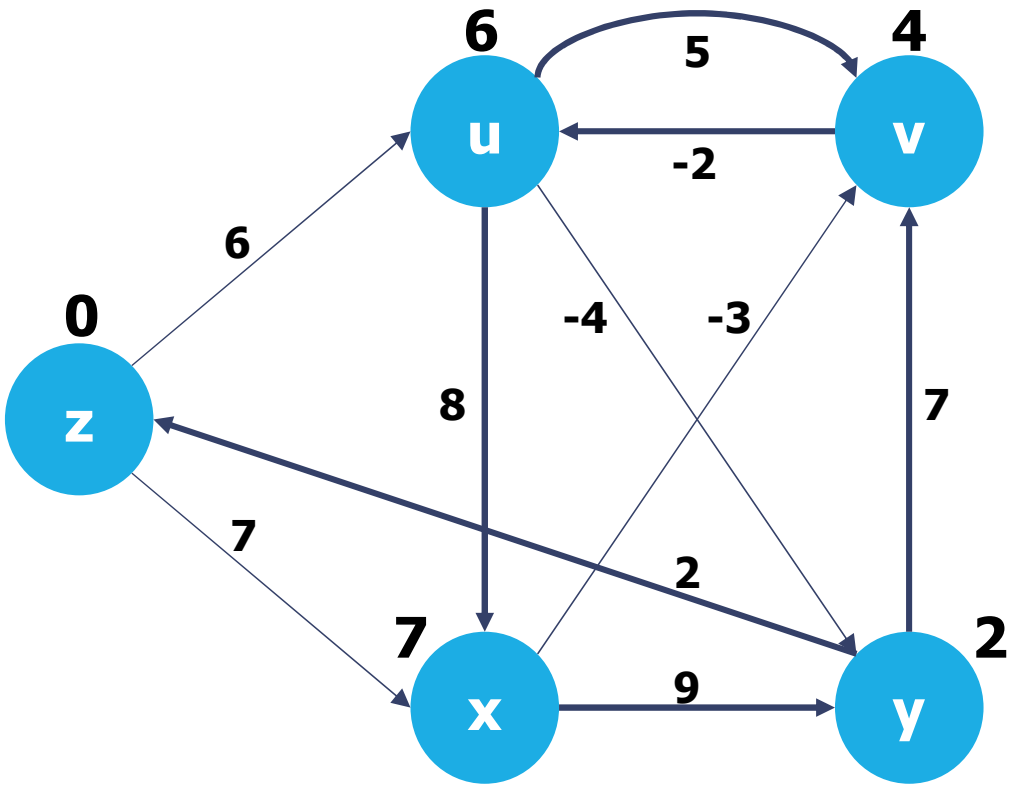
Passo 2

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



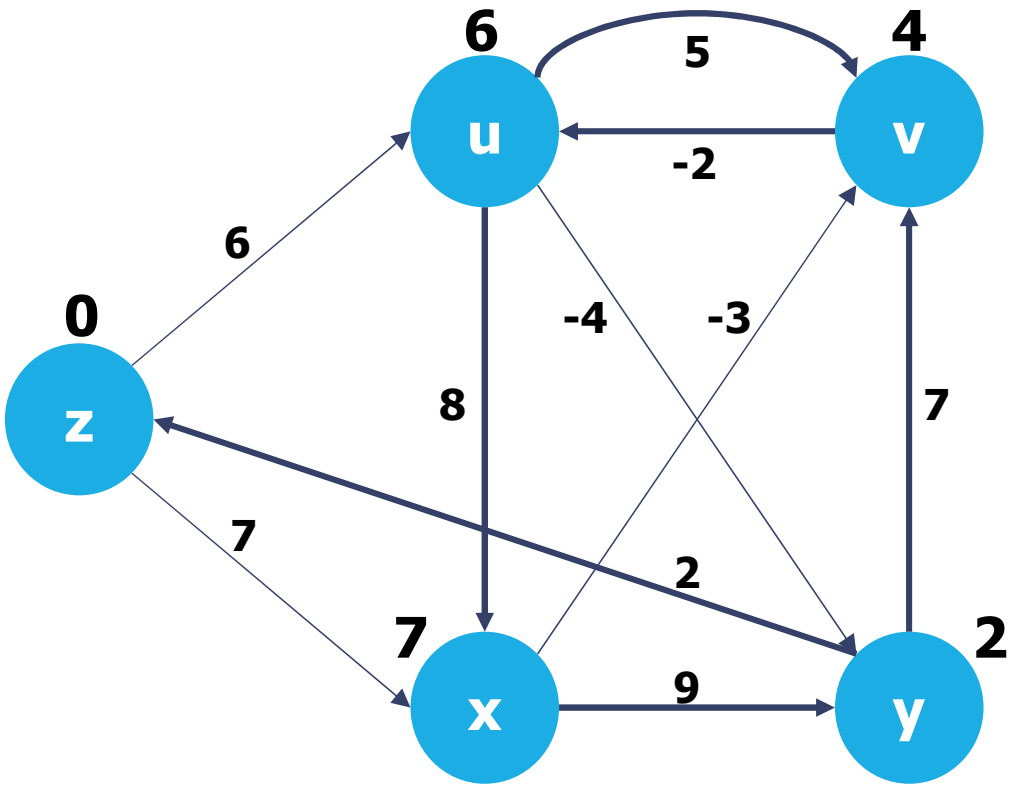
Passo 2

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



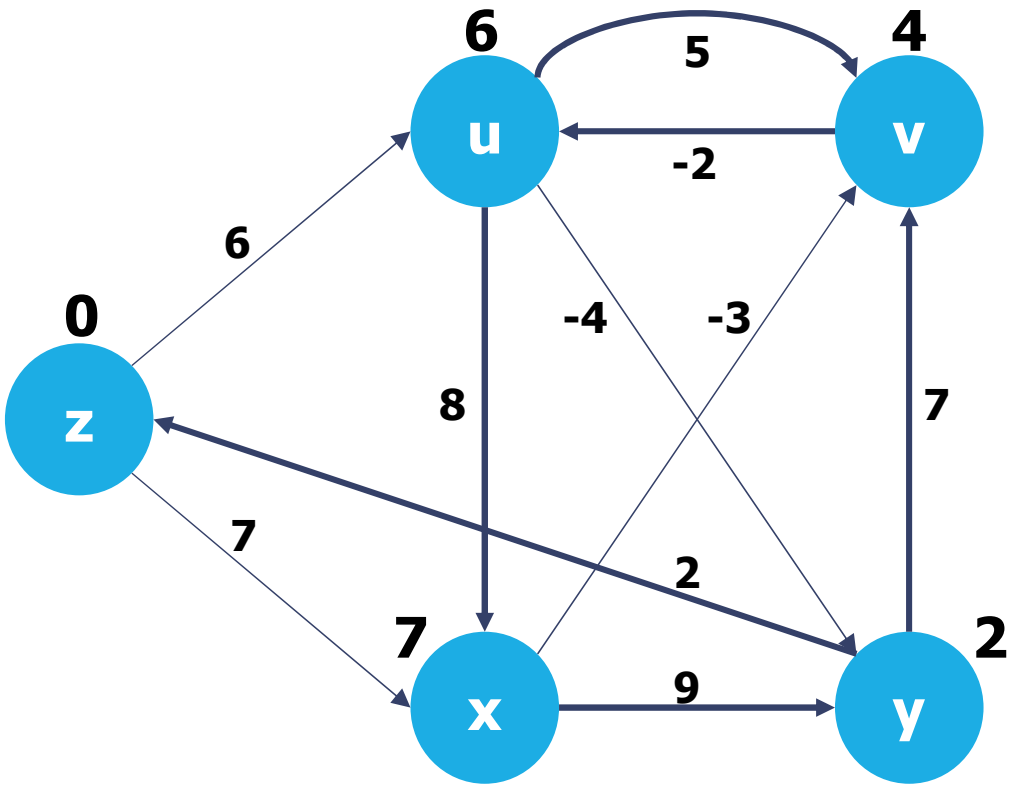
Passo 2

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



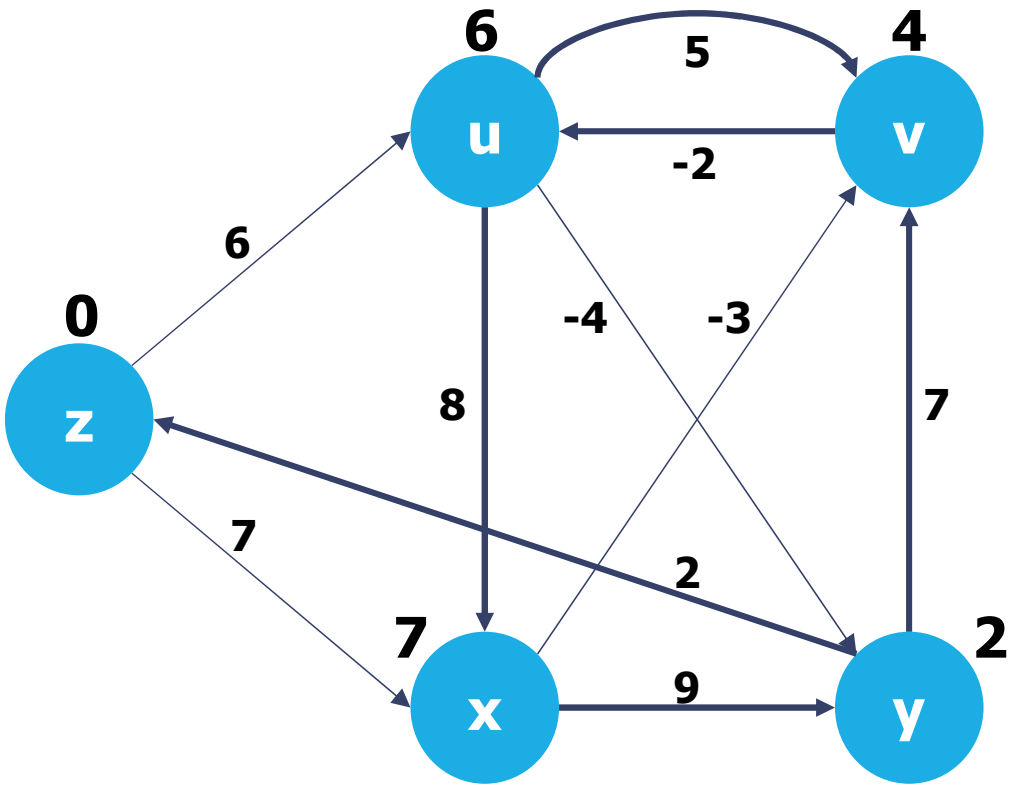
Passo 3

→ (u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



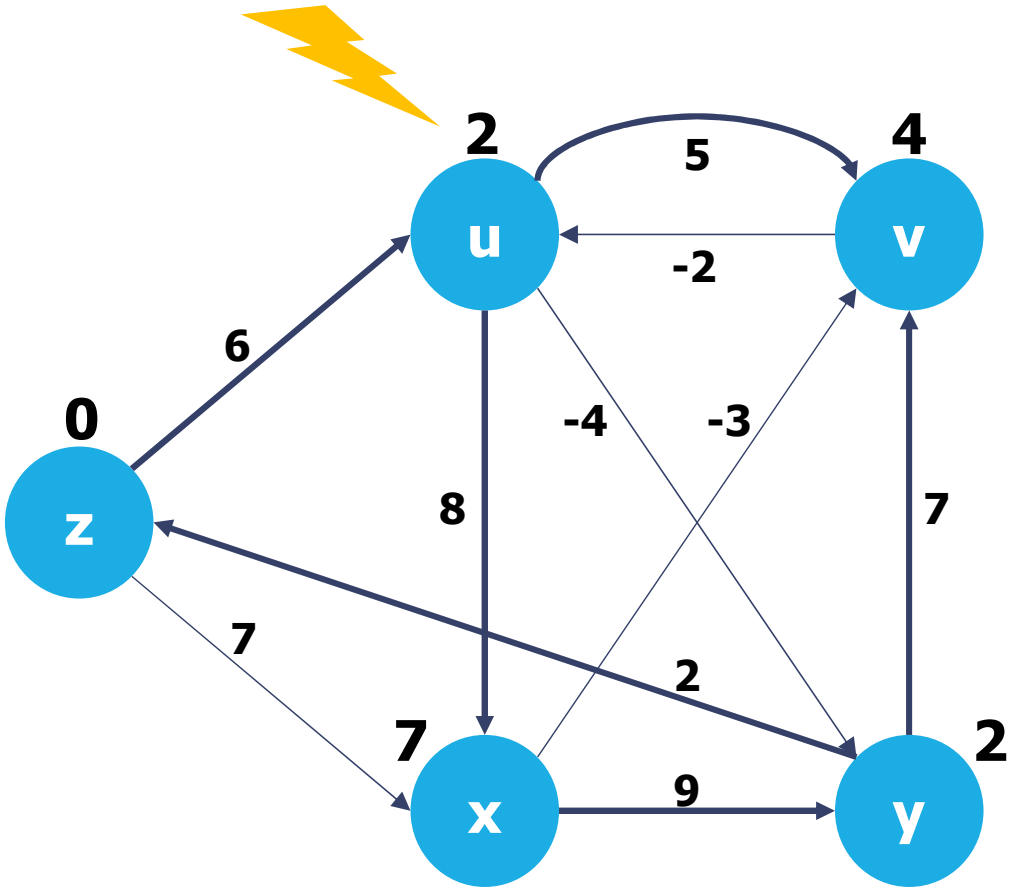
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



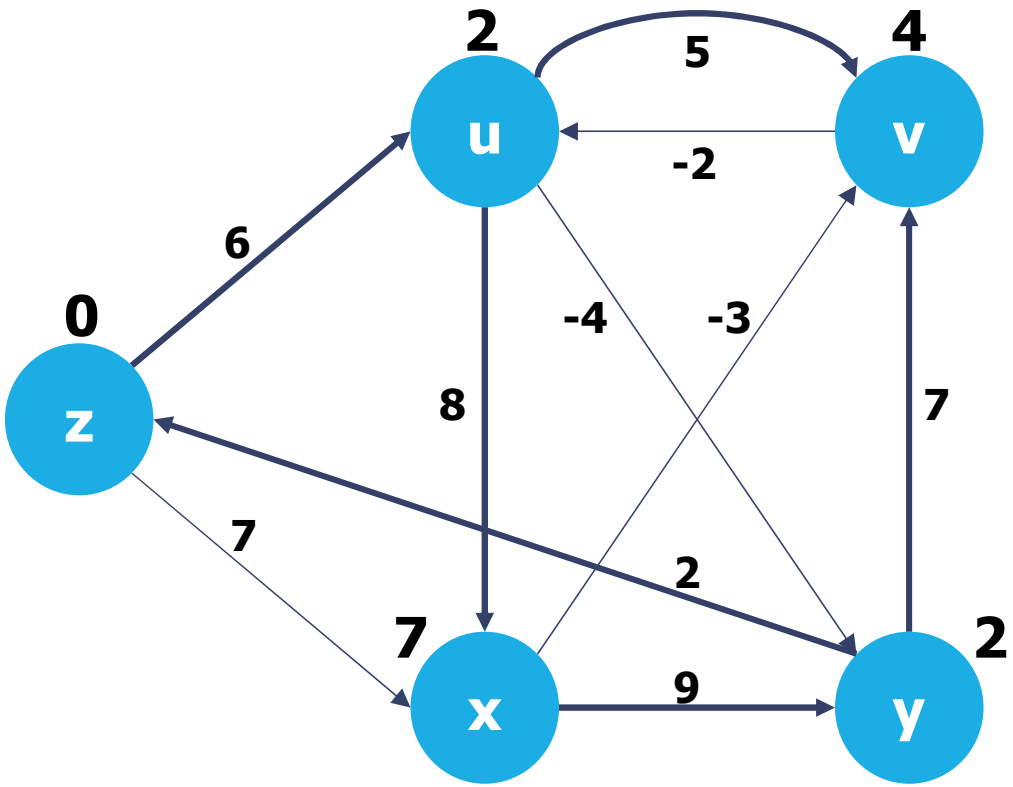
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



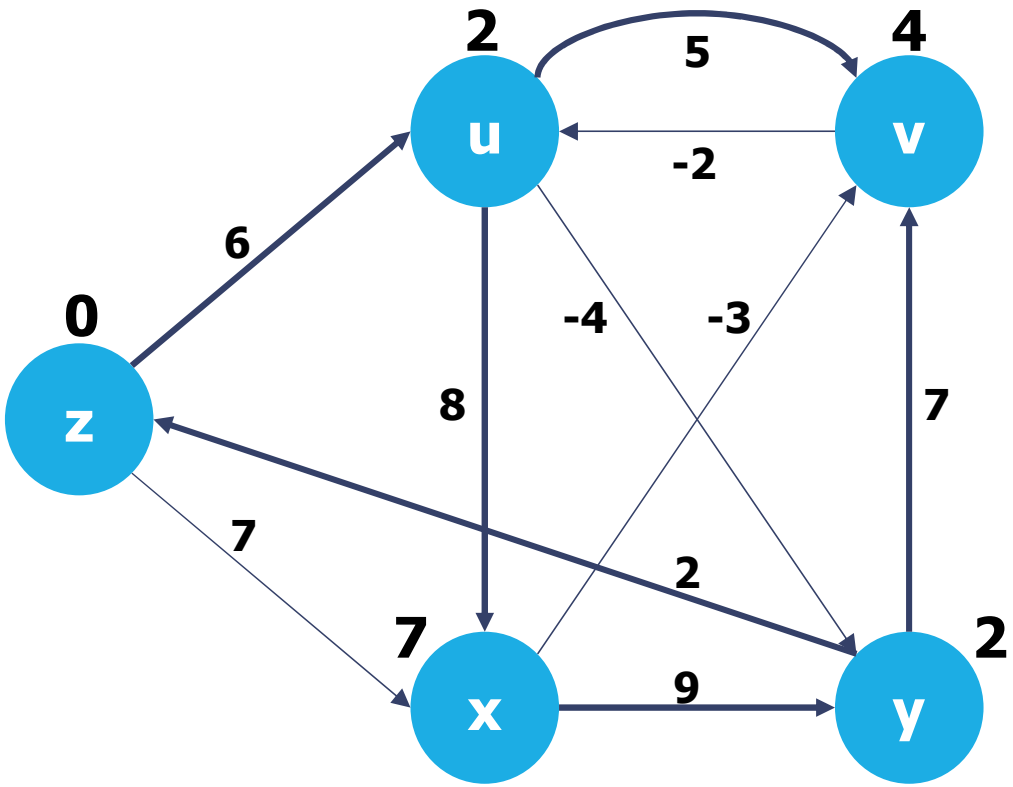
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



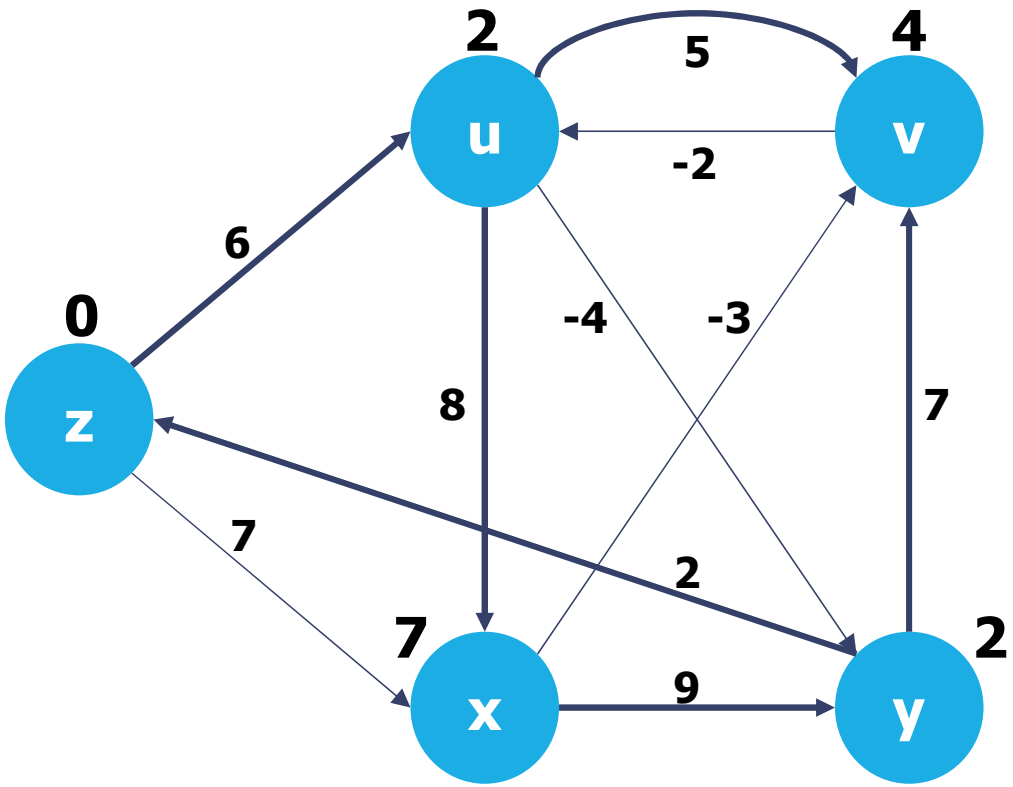
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



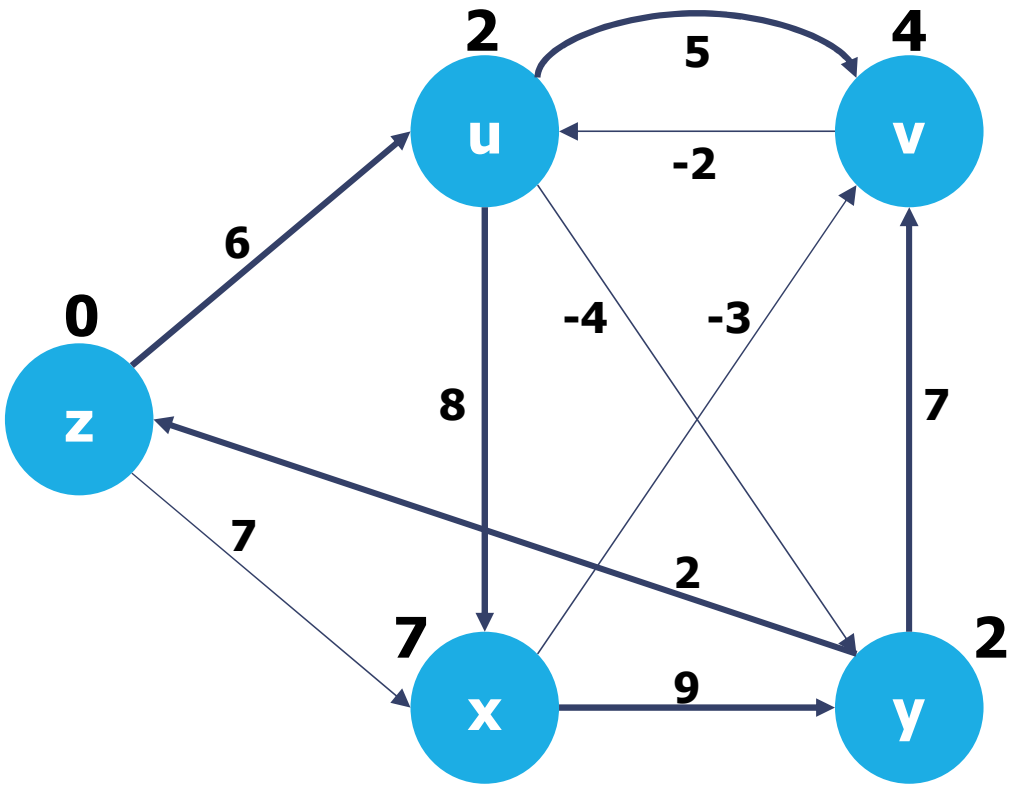
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



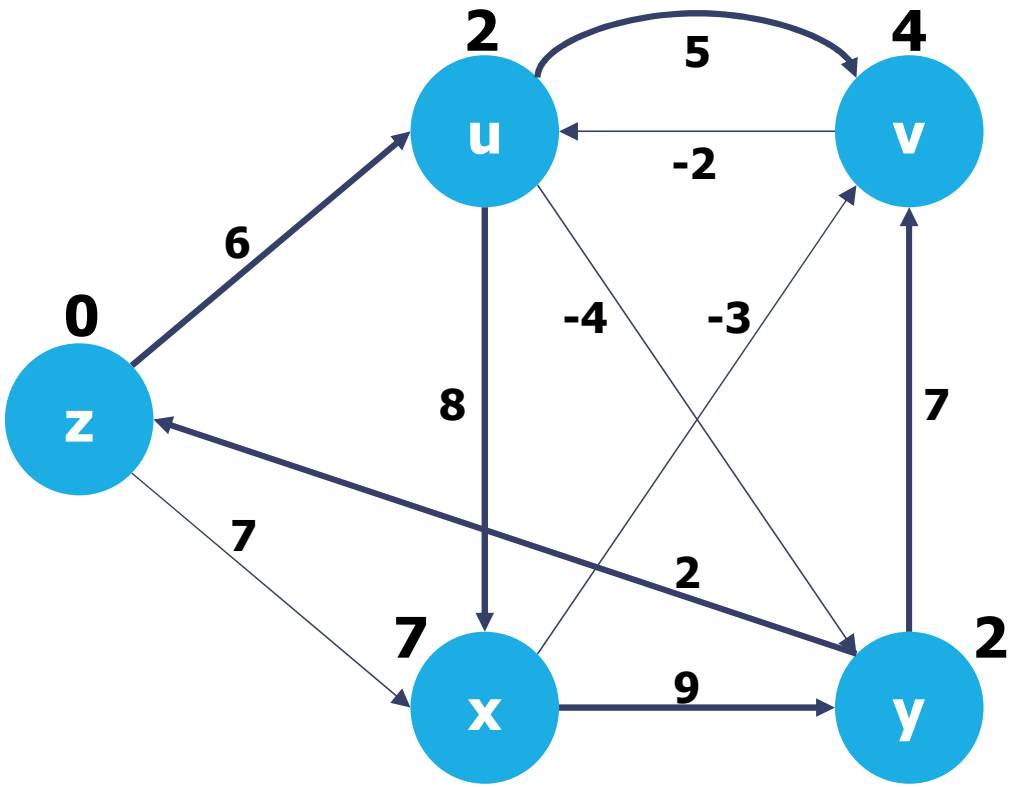
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



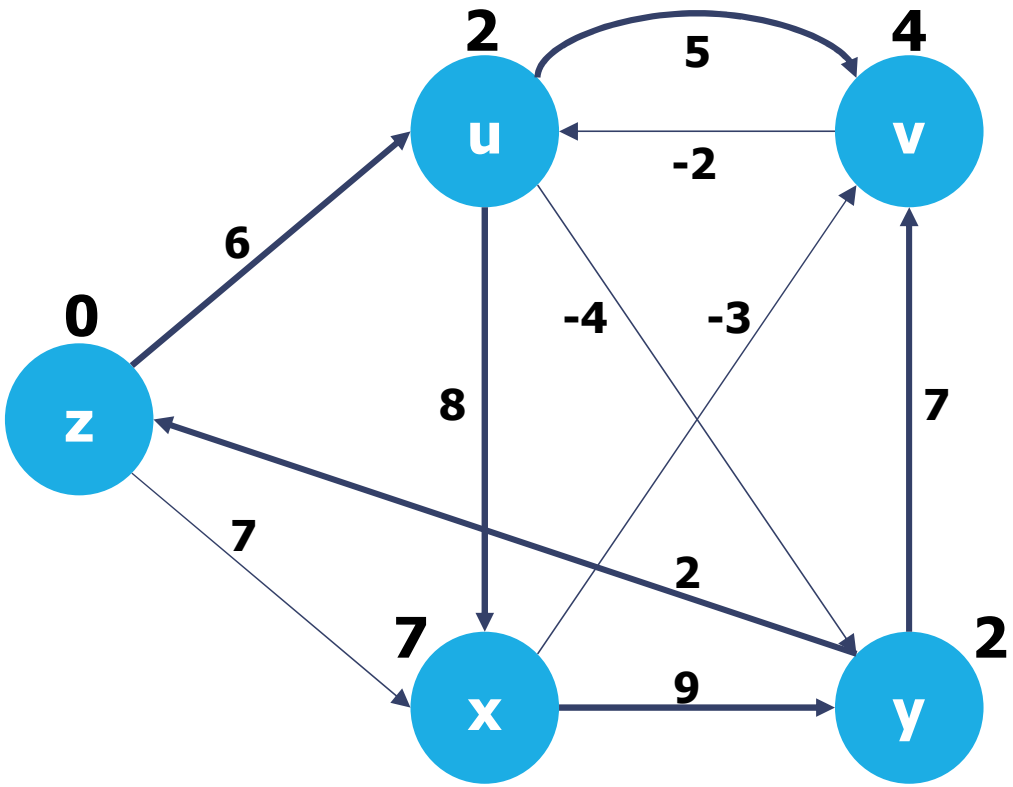
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



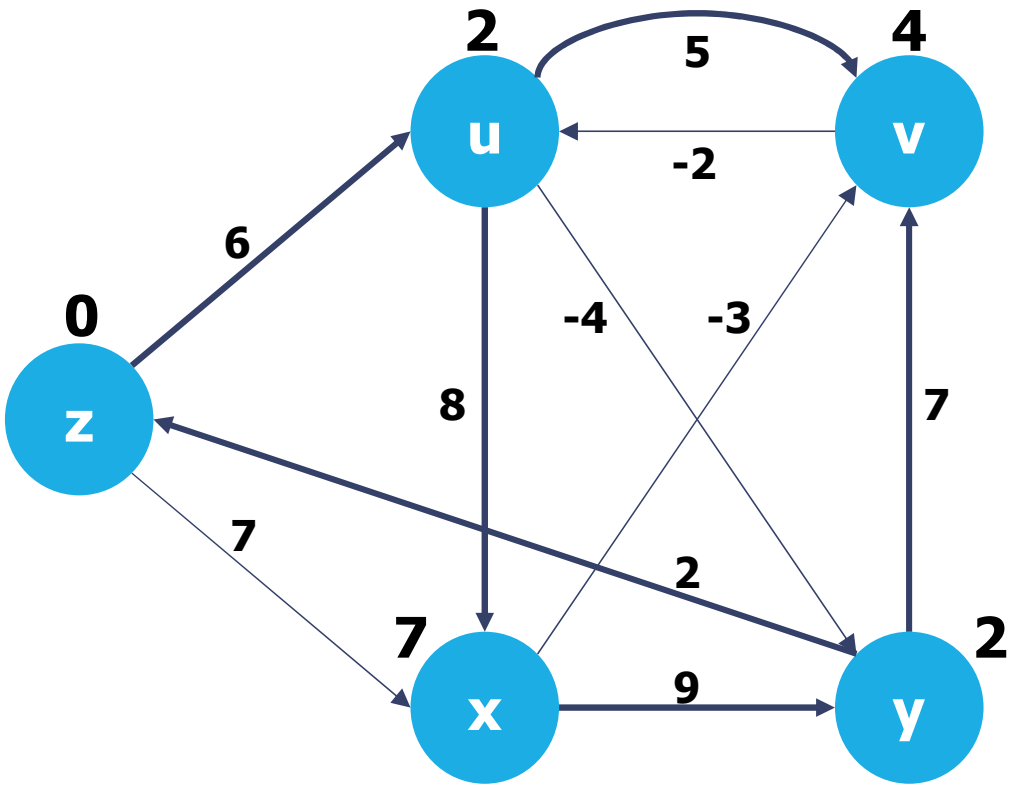
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



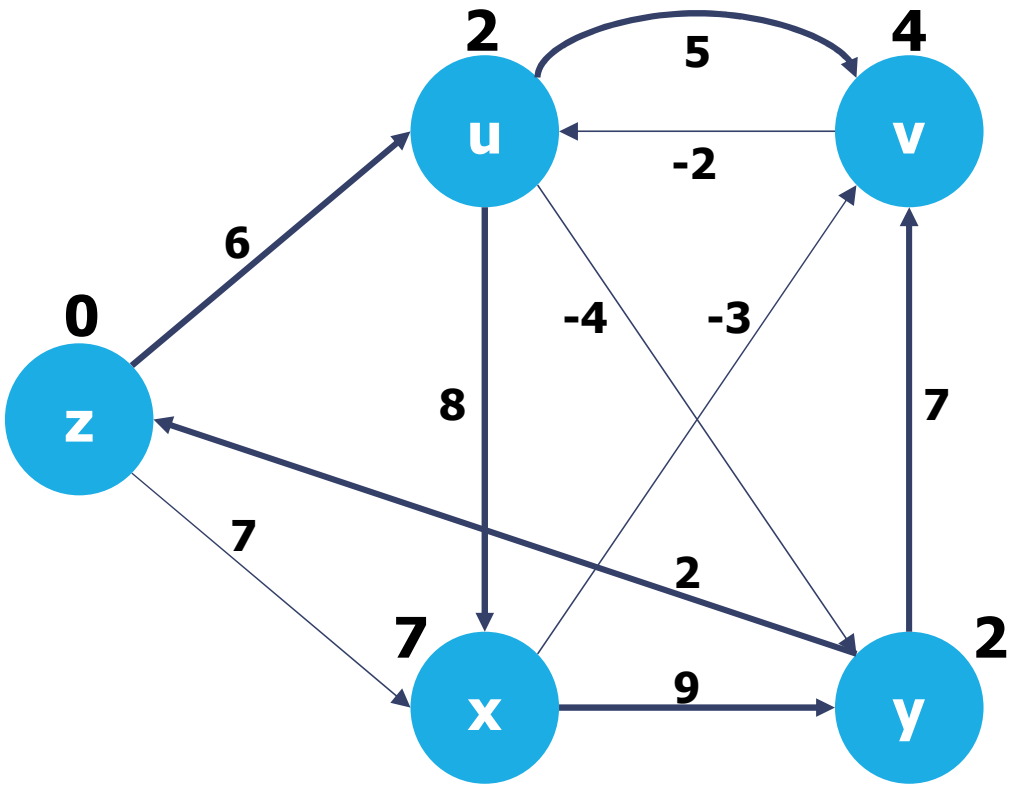
Passo 3

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



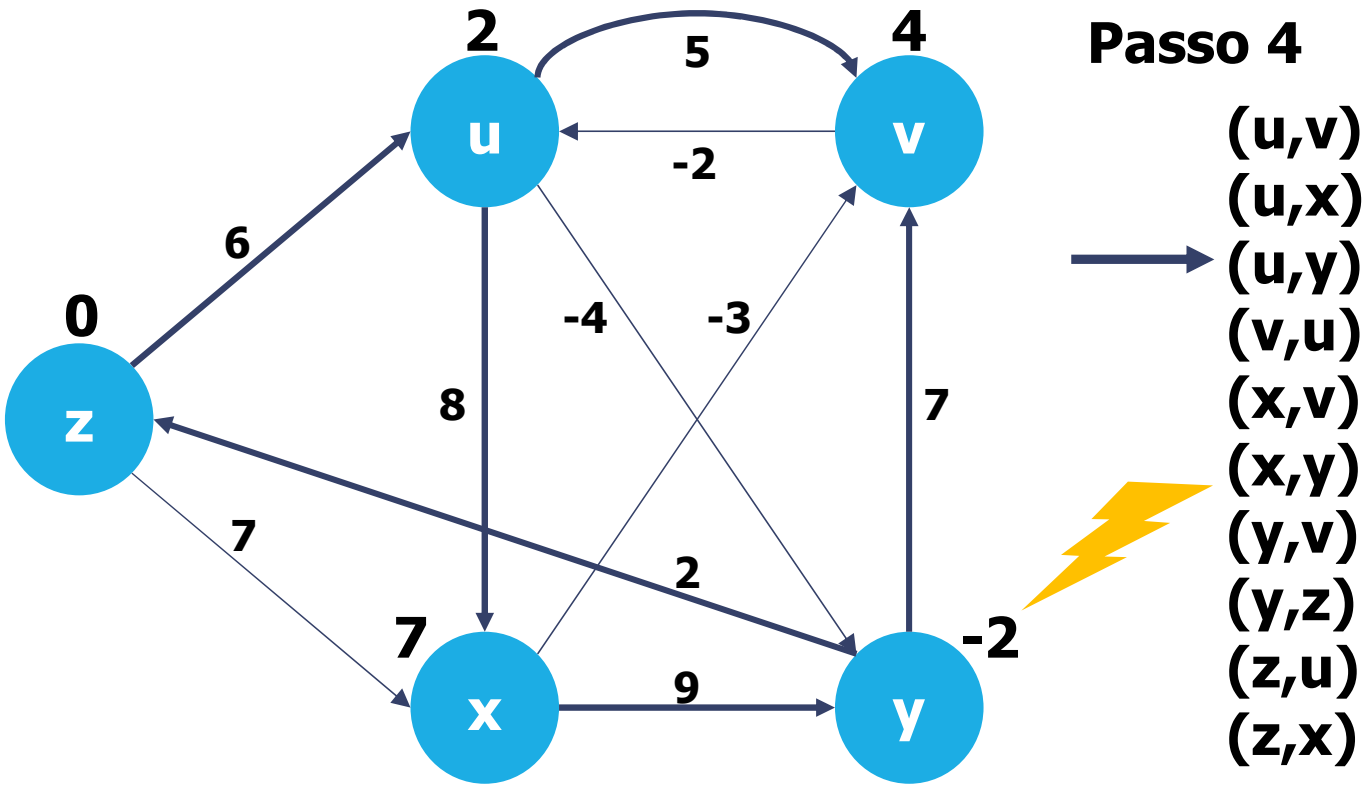
Passo 4

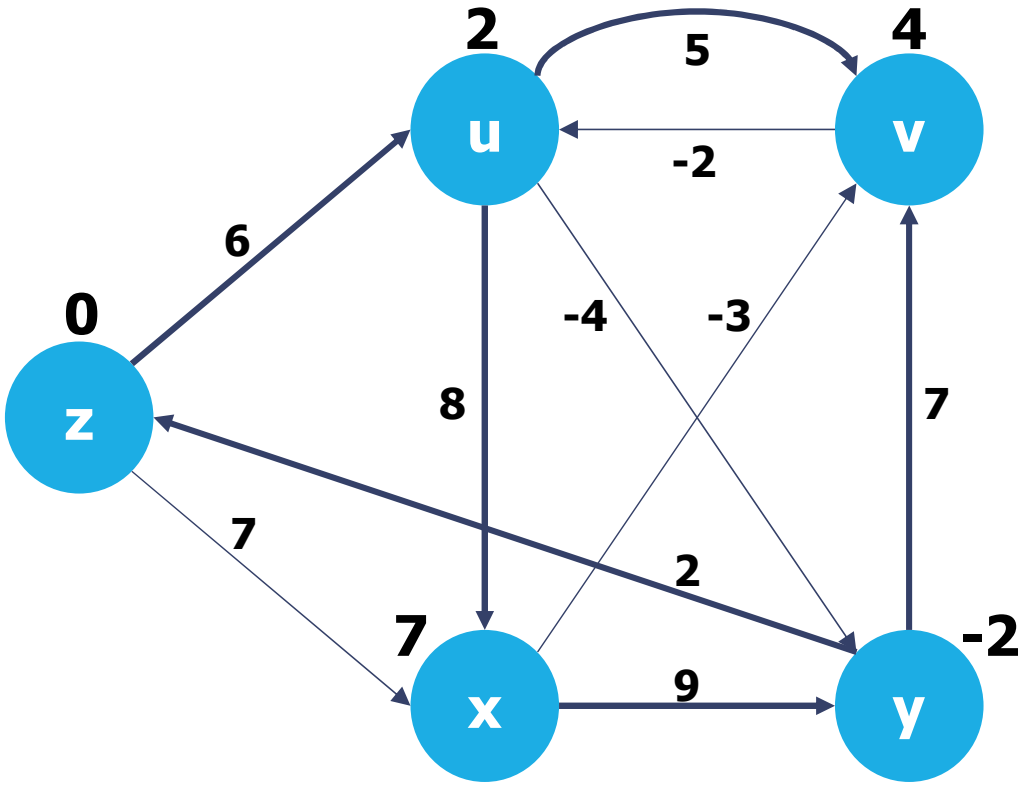
→ (u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 4

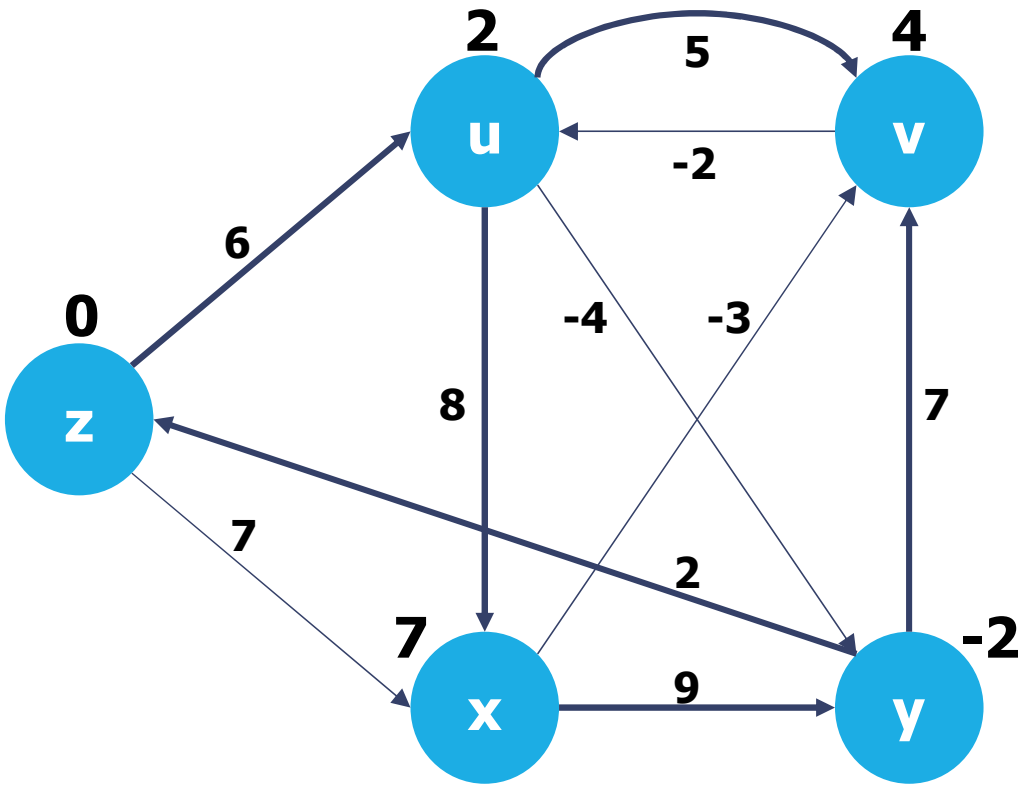
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)


2
4
5
u
v
-2
6
0
-4
-3
z
8
7
7
2
7
x
9
y
-2
Passo 4
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



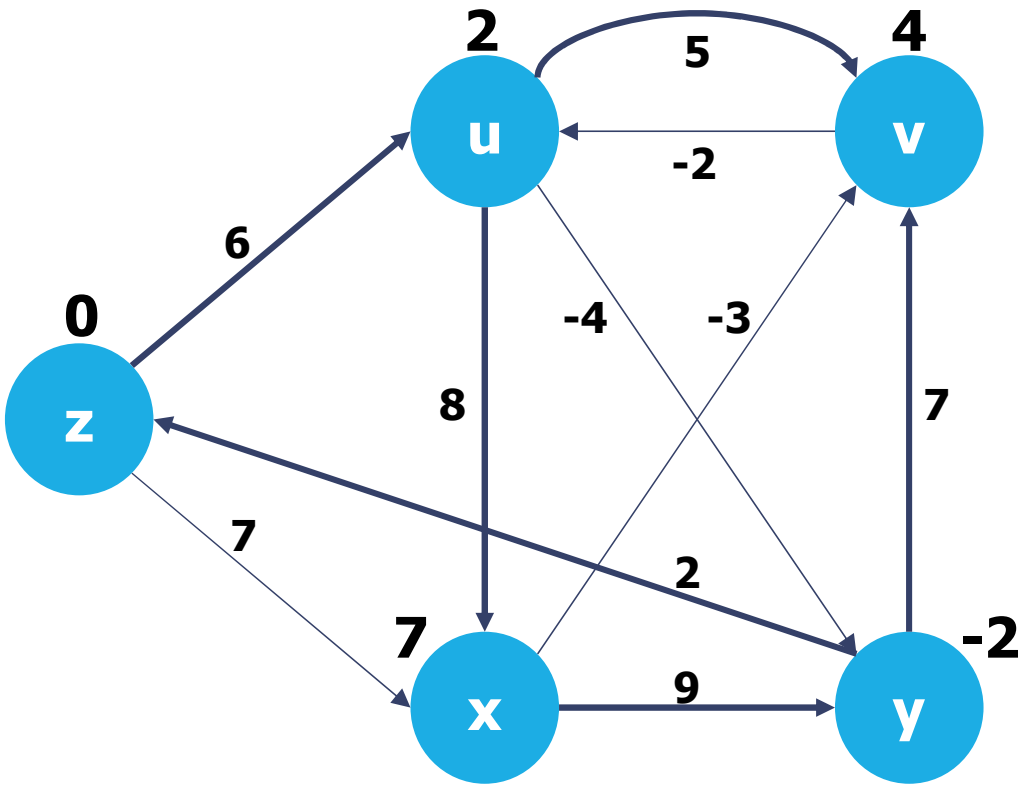
Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



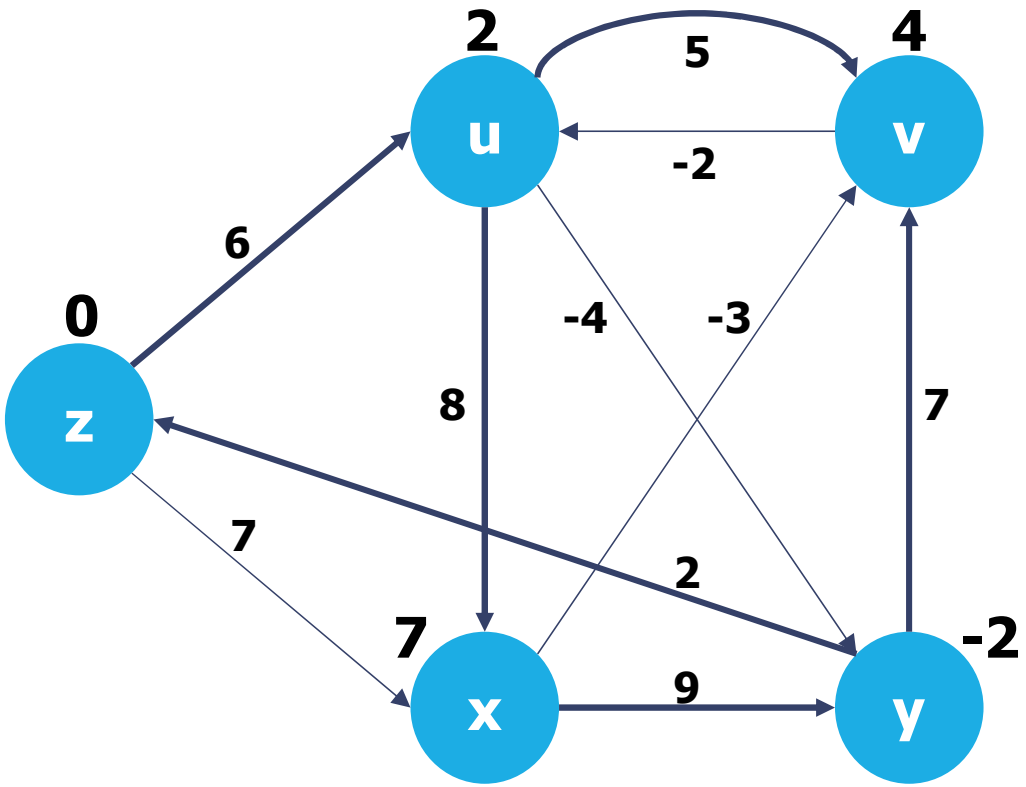
Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



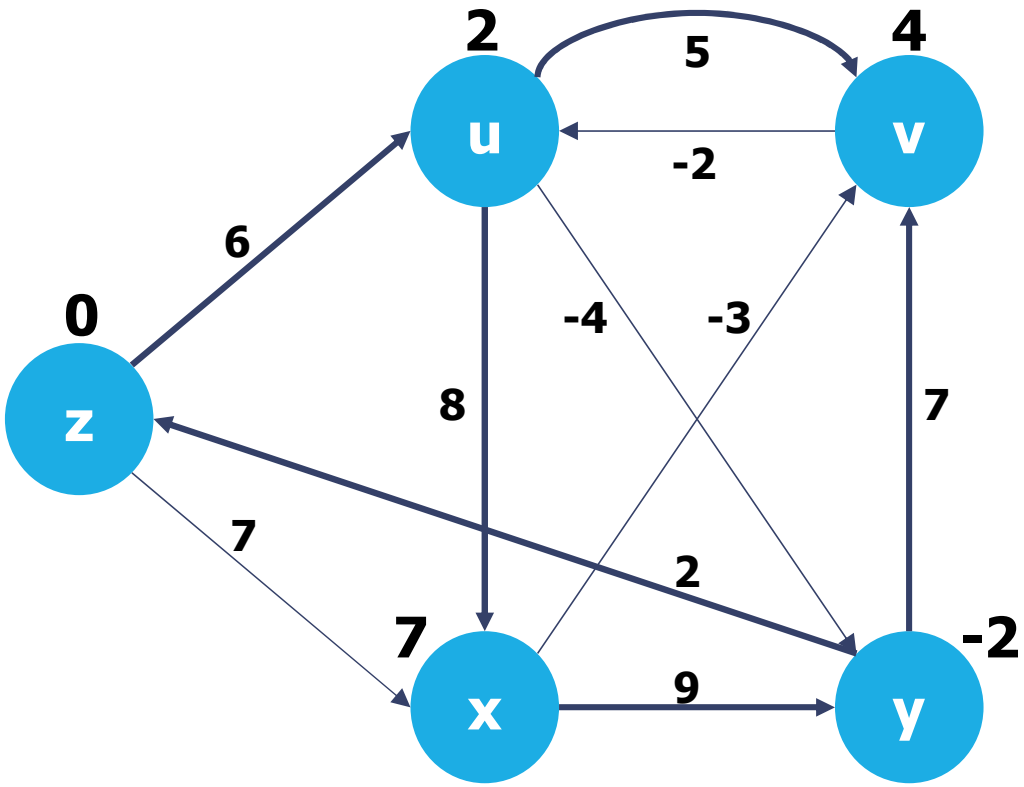
Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



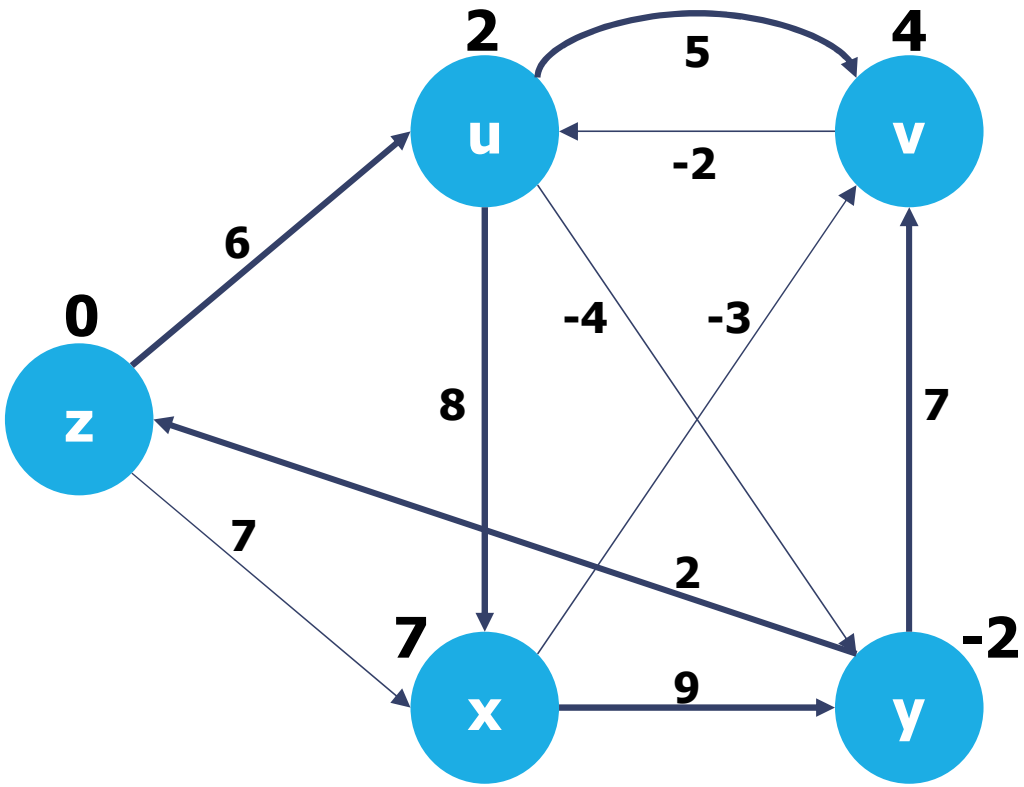
Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



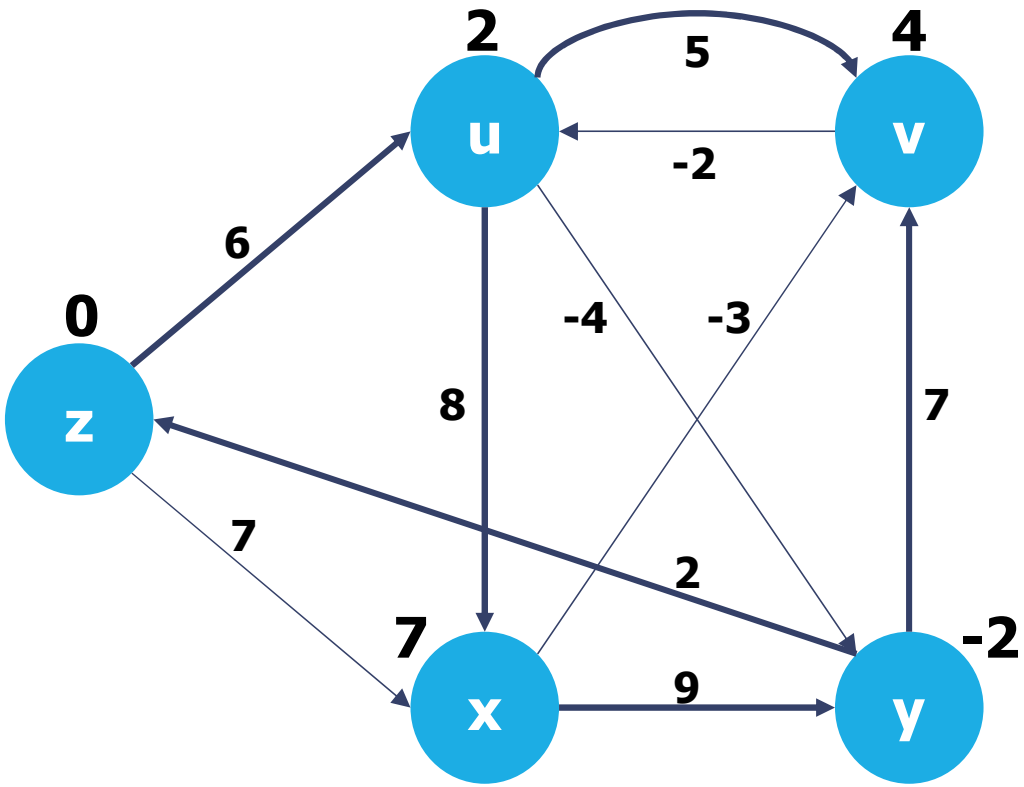
Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 4

(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)

Al |V|-esimo passo di rilassamento non diminuisce alcuna stima: terminazione con soluzione ottima.

Ricordare: la terminazione può anche avvenire prima.

Complessità

- Inizializzazione — O(|V|)
- |V|-1 passi in ciascuno dei quali si rilassano tutti gli archi — O(|V||E|)
- |V|esimo passo di controllo in cui si rilassano tutti gli archi — O(|E|)

T(n) = O(|V| |E|).

```
void GRAPHspBF(Graph G, int id){
  int v, i, negcycfound;
  link t;
  int *st, *d;

  st = malloc(G->V*sizeof(int));
  d = malloc(G->V*sizeof(int));

  for (v = 0; v < G->V; v++) {
    st[v]= -1;
    d[v] = maxWT;
  }

  d[id] = 0;
  st[id] = id;
```

```
for (i=0; i<G->V-1; i++)
  for (v=0; v<G->V; v++)
    if (d[v] < maxWT)
      for (t=G->ladj[v]; t!=G->z ; t=t->next)
        if (d[t->v] > d[v] + t->wt) {
          d[t->v] = d[v] + t->wt;
          st[t->v] = v;
        }
negcycfound = 0;
for (v=0; v<G->V; v++)
  if (d[v] < maxWT)
    for (t=G->ladj[v]; t!=G->z ; t=t->next)
      if (d[t->v] > d[v] + t->wt)
        negcycfound = 1;
```

```
  if (negcycfound == 0) {
    printf("\n Shortest path tree\n");
    for (v = 0; v < G->V; v++)
      printf("Parent of %s is %s \n",
              STsearchByIndex(G->tab, v),
              STsearchByIndex (G->tab, st[v]));
    printf("\n Min.dist. from %s\n",
              STsearchByIndex (G->tab, s));
    for (v = 0; v < G->V; v++)
      printf("%s: %d\n", STsearchByIndex (G->tab, v), d[v]);
  }
  else
    printf("\n Negative cycle found!\n");
}
```

Applicazione: arbitrage

In Economia e Finanza si definisce «arbitrage» l'acquisto e la vendita in simultanea di beni (asset) su mercati diversi per sfruttare piccole differenze nei prezzi.

Si applica ad asset quali: azioni (stocks), materie prime (commodities), valute (currencies).

L'arbitrageur ha la possibilità di guadagno a costo zero e senza rischi.

L'arbitrage offre un meccanismo che garantisce che i prezzi non devino sostanzialmente dal valore corretto per periodi lunghi.

Arbitrage trading program (ATP):

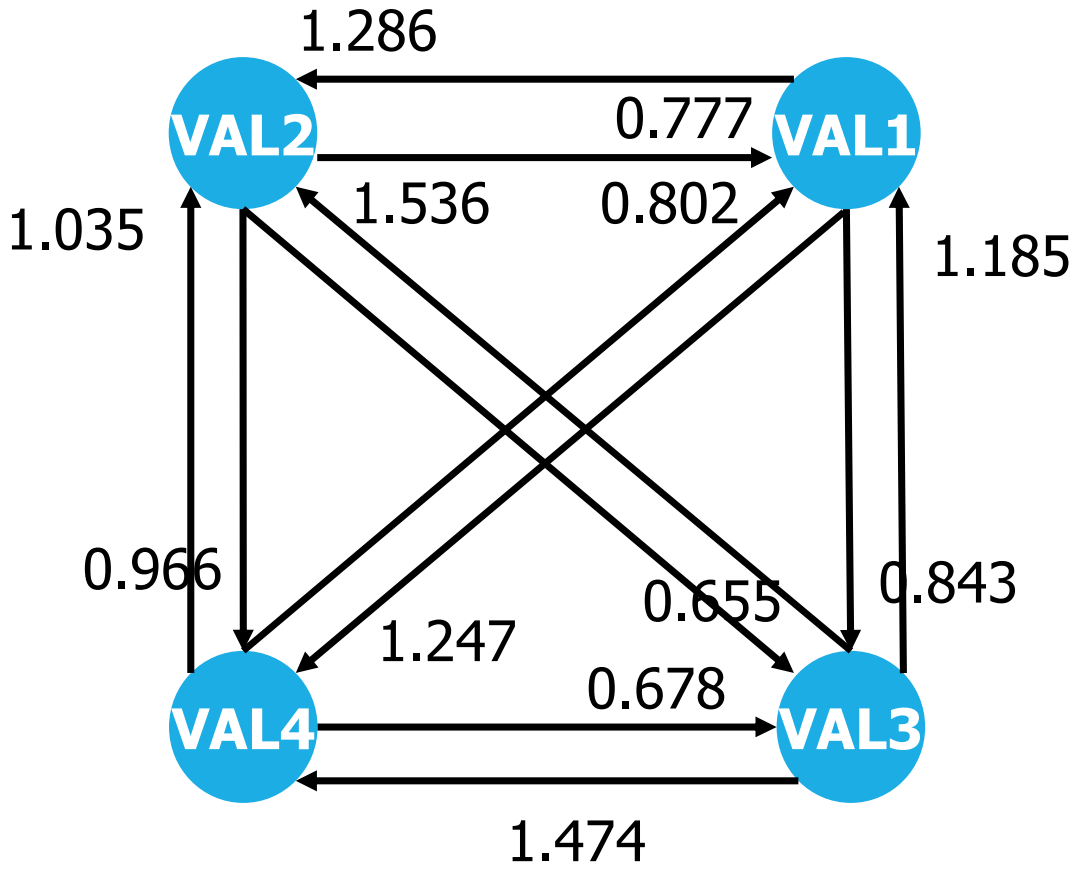
- osserva i prezzi sui mercati
- rileva anomalie di prezzo in millisecondi ("electronic eye")
- applica Bellman-Ford al grafo per rilevare cicli a peso negativo
- effettua le transazioni.

Esempio: mercato delle valute

	VAL1	VAL2	VAL3	VAL4
VAL1	1.000	1.286	0.843	1.247
VAL2	0.777	1.000	0.655	0.966
VAL3	1.185	1.526	1.000	1.474
VAL4	0.802	1.035	0.678	1.000

Tassi di cambio tra 4 valute

$$\forall\ i \neq j\ VAL_i/VAL_j \neq 1/(VAL_j/VAL_i)$$


1.286
VAL2
0.777
VAL1
1.536
0.802
1.035
1.185
0.966
0.655
0.843
1.247
0.678
VAL4
VAL3
1.474

1000 VAL2 = 777 VAL1 = 655,11 VAL3 = 1006,10 VAL2

Guadagno di 6,10 VAL2 $\Rightarrow$ arbitrage

Sul ciclo

VAL2 – VAL1 – VAL3 – VAL2

il prodotto dei tassi di cambio è

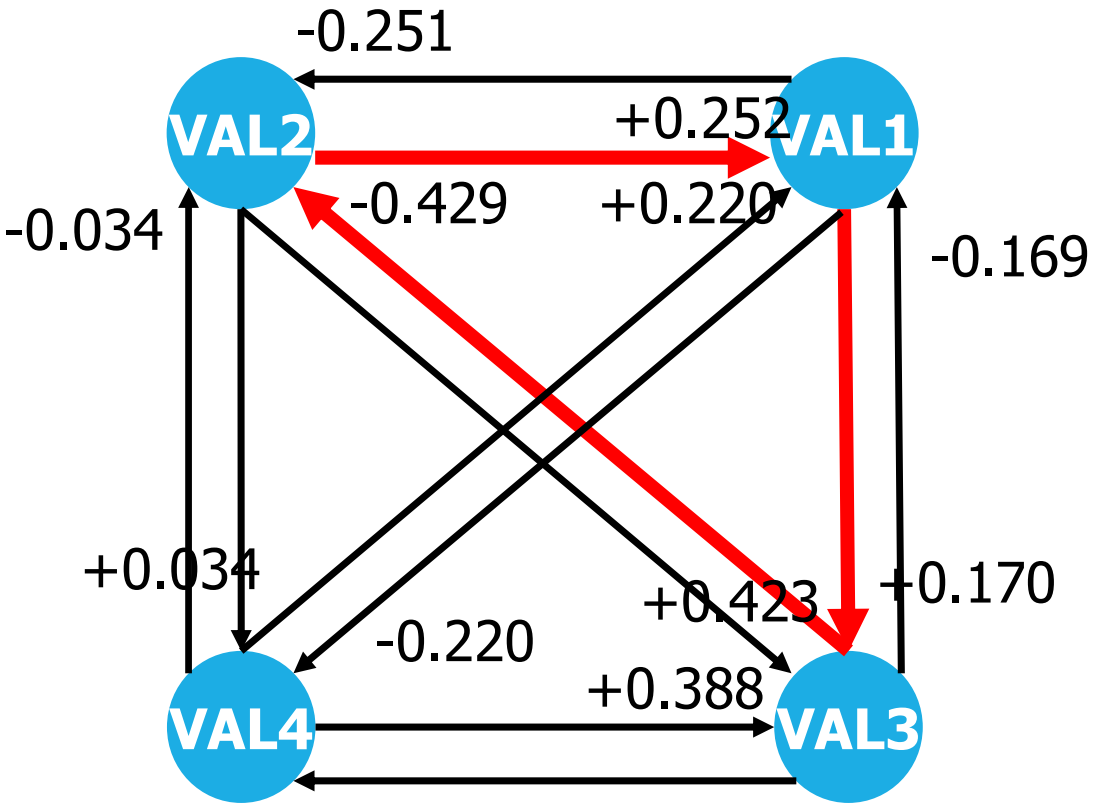
0.777 * 0.843 * 1.536 = 1,0061 > 1.0

$\Downarrow$

c'è arbitrage quando $\exists$ ciclo a peso > 1

Modello:

- grafo orientato pesato completo
- peso degli archi = -ln(tasso di cambio)
- il ciclo con prodotto dei cambi > 1 diventa un ciclo in cui la somma dei logaritmi ha peso negativo
- algoritmo di Bellman-Ford per rilevare il ciclo a peso negativo



+0.220
+0.170
-0.429
=
-0.039

Riferimenti

- Principi:
  - Sedgewick Part 5 21.1
  - Cormen 24.1
- Algoritmo di Dijkstra:
  - Sedgewick Part 5 21.2
  - Cormen 24.4
- Cammini minimi e massimi in DAG:
  - Sedgewick Part 5 21.4
  - Cormen 24.3
- Algoritmo di Bellman-Ford:
  - Sedgewick Part 5 21.7
  - Cormen 24.2

Esercizi di teoria

- 12. Visite dei grafi e applicazioni
  - 12.1 Algoritmo di Dijkstra
  - 12.2 Algoritmo per i DAG pesati
  - 12.3 Algoritmo di Bellman-Ford