I cammini minimi

Paolo Camurati, Fulvio Corno, Matteo Sonza Reorda
Dip. Automatica e Informatica
Politecnico di Torino

Cammini minimi

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

Definizioni:

peso w(p) di un cammino p:

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

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

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

Cammino minimo da u a v:

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

Problemi classici

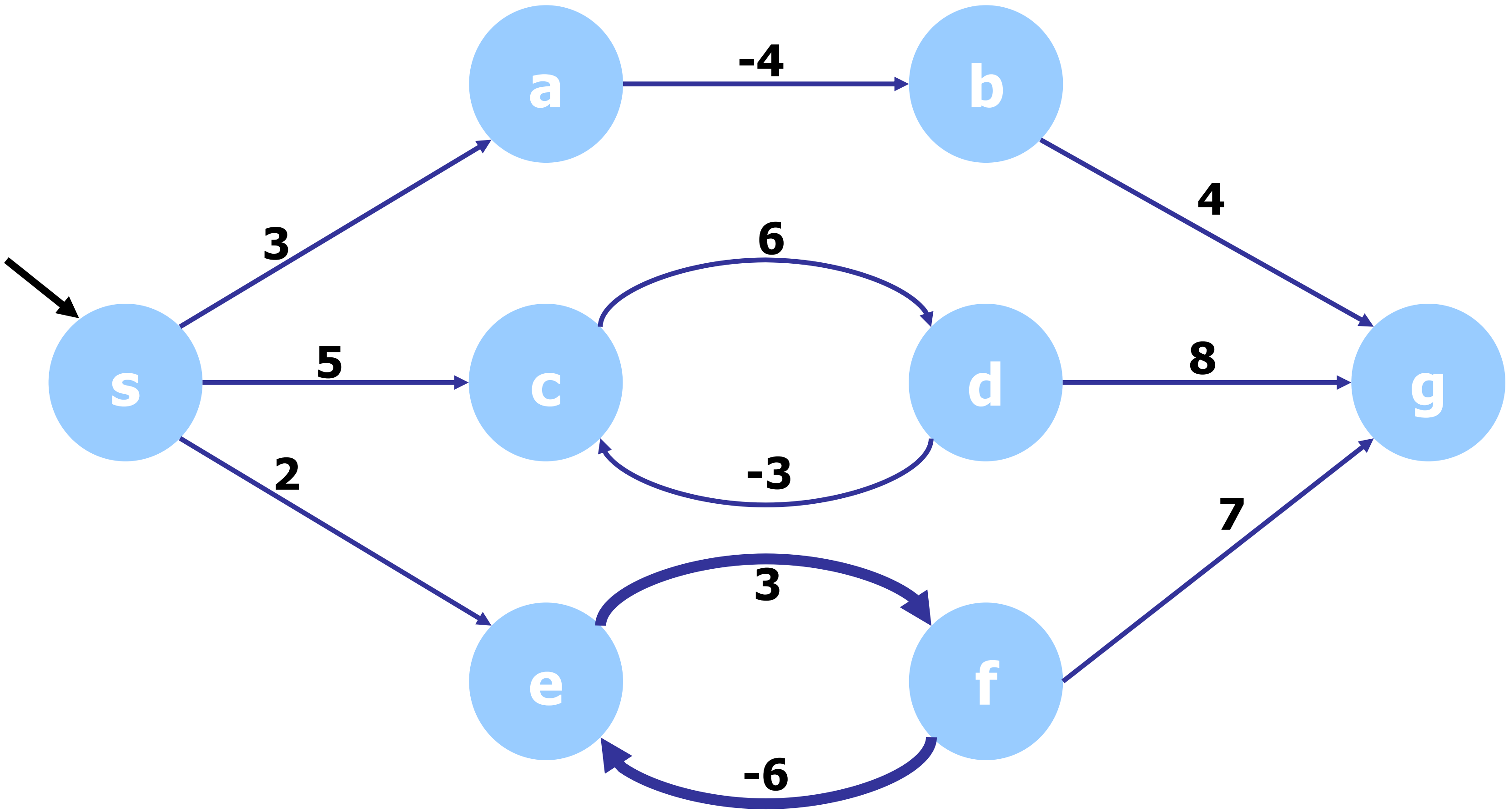
Cammini minimi:

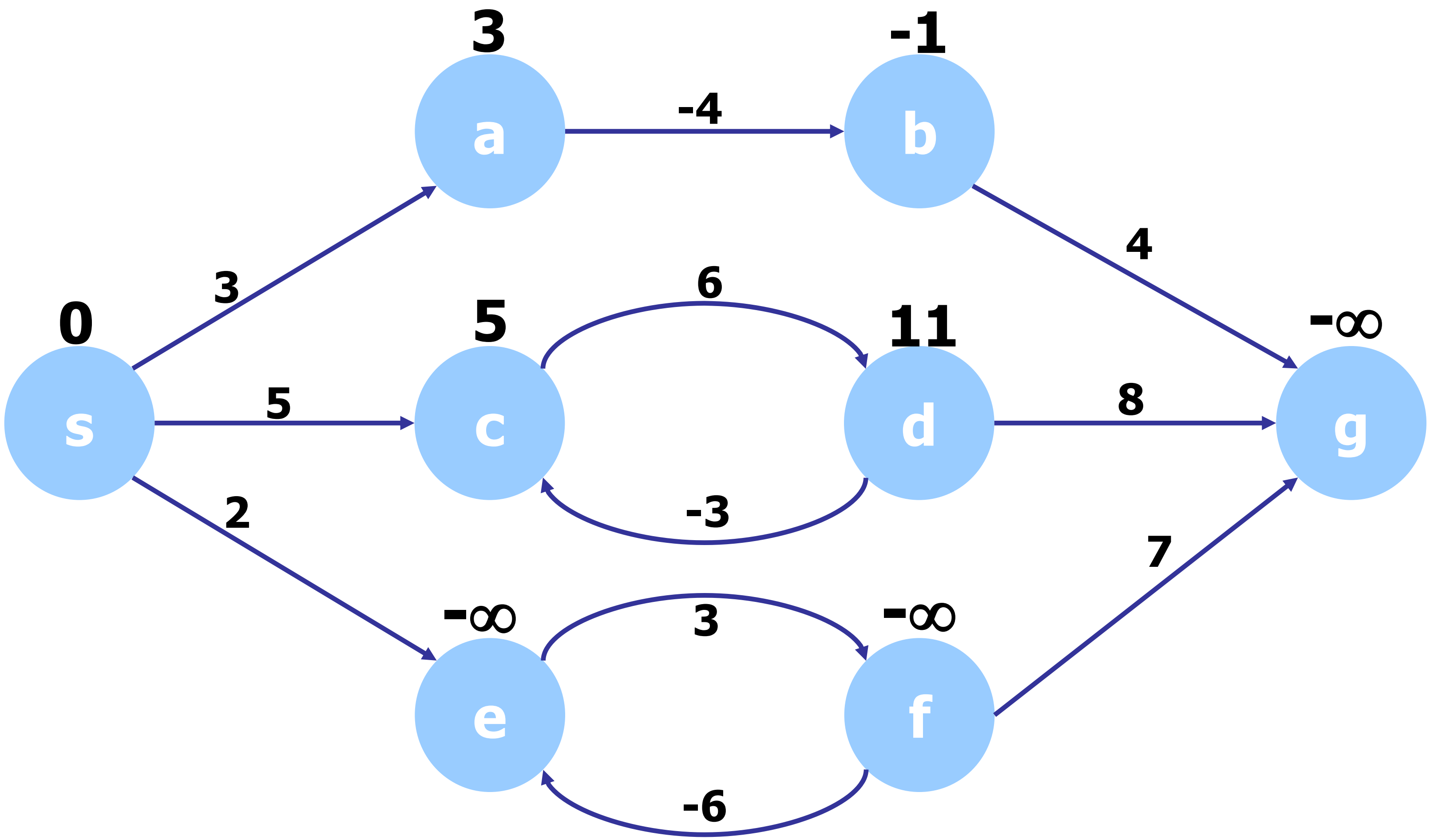
- da sorgente singola: cammino minimo e suo peso da s a ogni altro vertice v
  - algoritmo di Dijkstra
  - algoritmo di Bellman-Ford
- con destinazione singola
- tra una coppia di vertici
- tra tutte le coppie di vertici.

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 non definito, $\nexists$ 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
2
-3
7
-∞
3
-∞
e
f
-6

Rappresentazione dei cammini minimi

Vettore dei predecessori `st[v]`:

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

Sottografo dei predecessori:

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

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

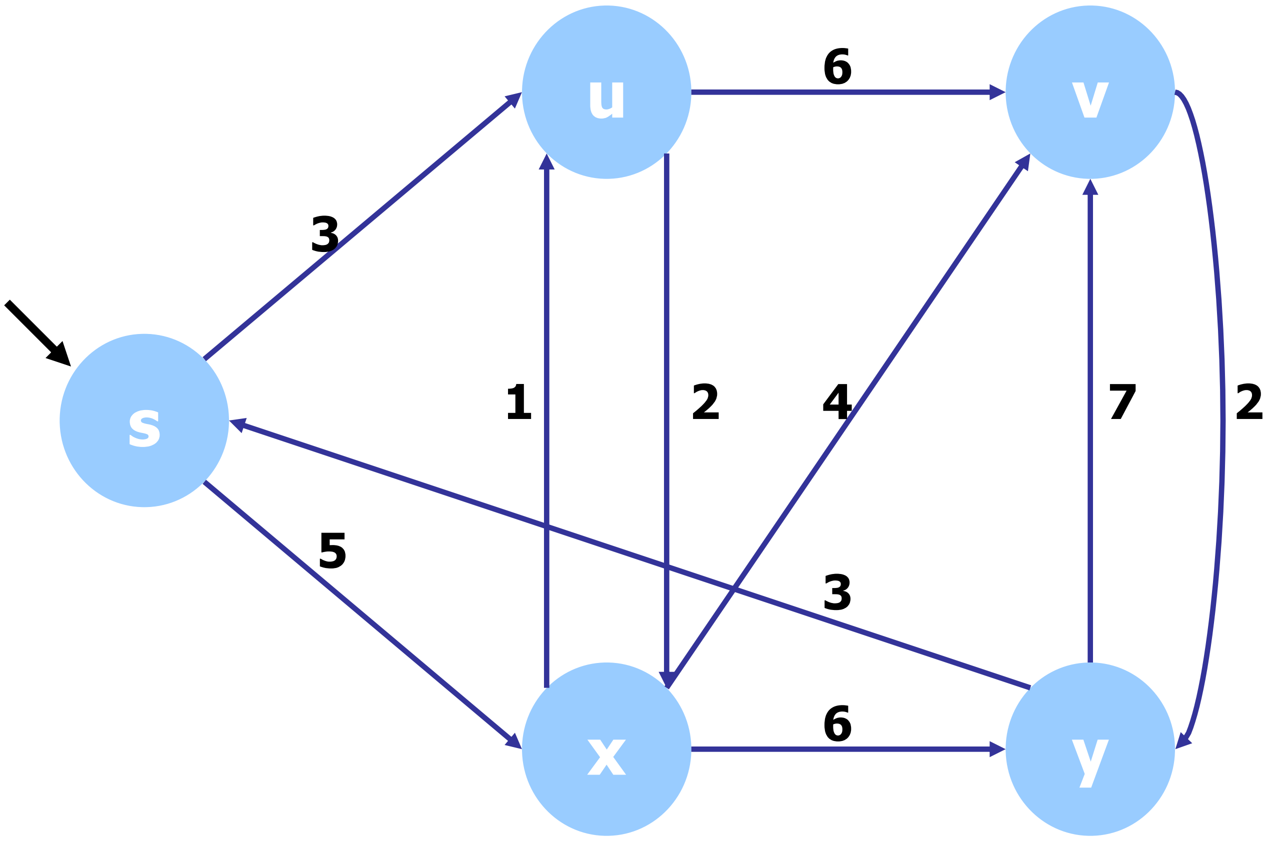
Albero dei cammini minimi:

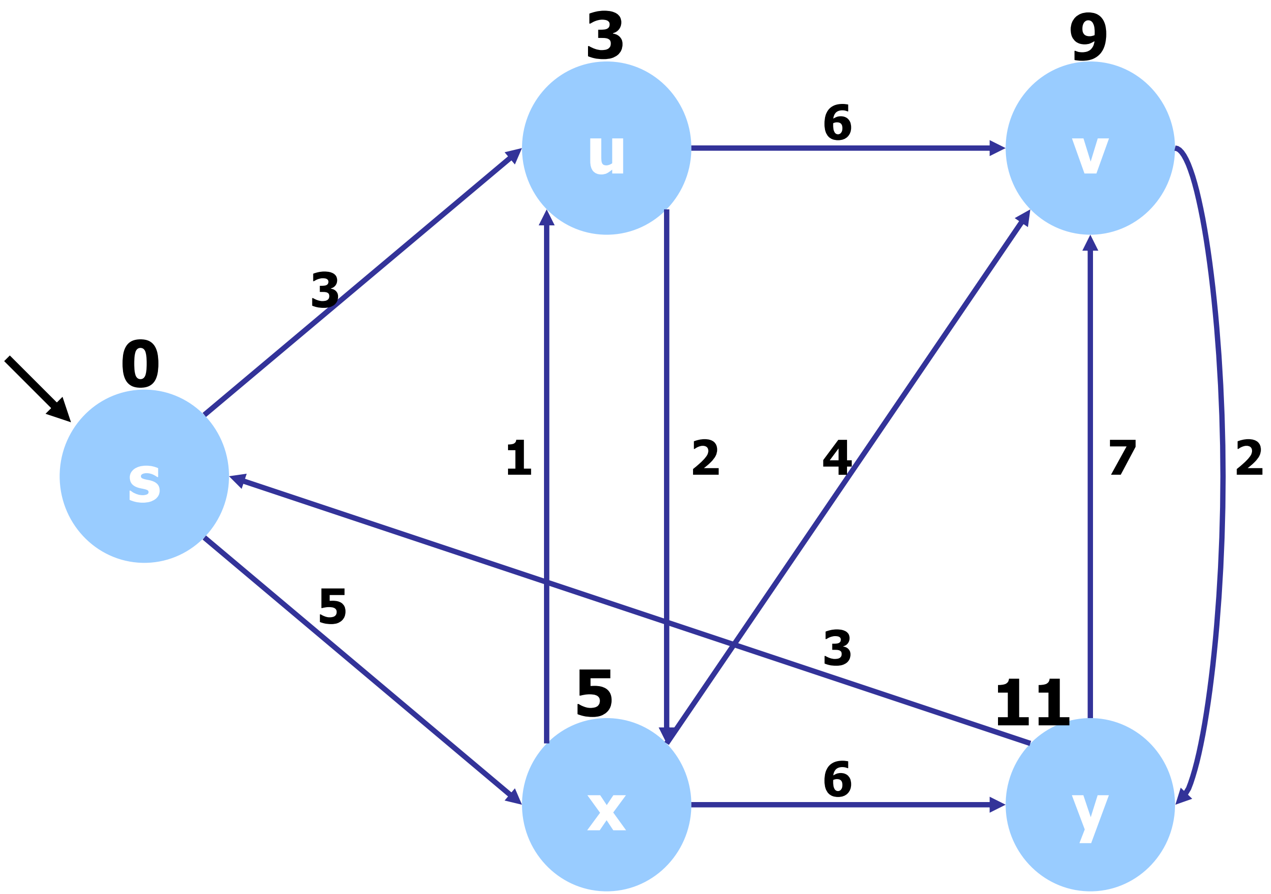
$G' = (V', E')$ dove $V' \subseteq V$ && $E' \subseteq E$

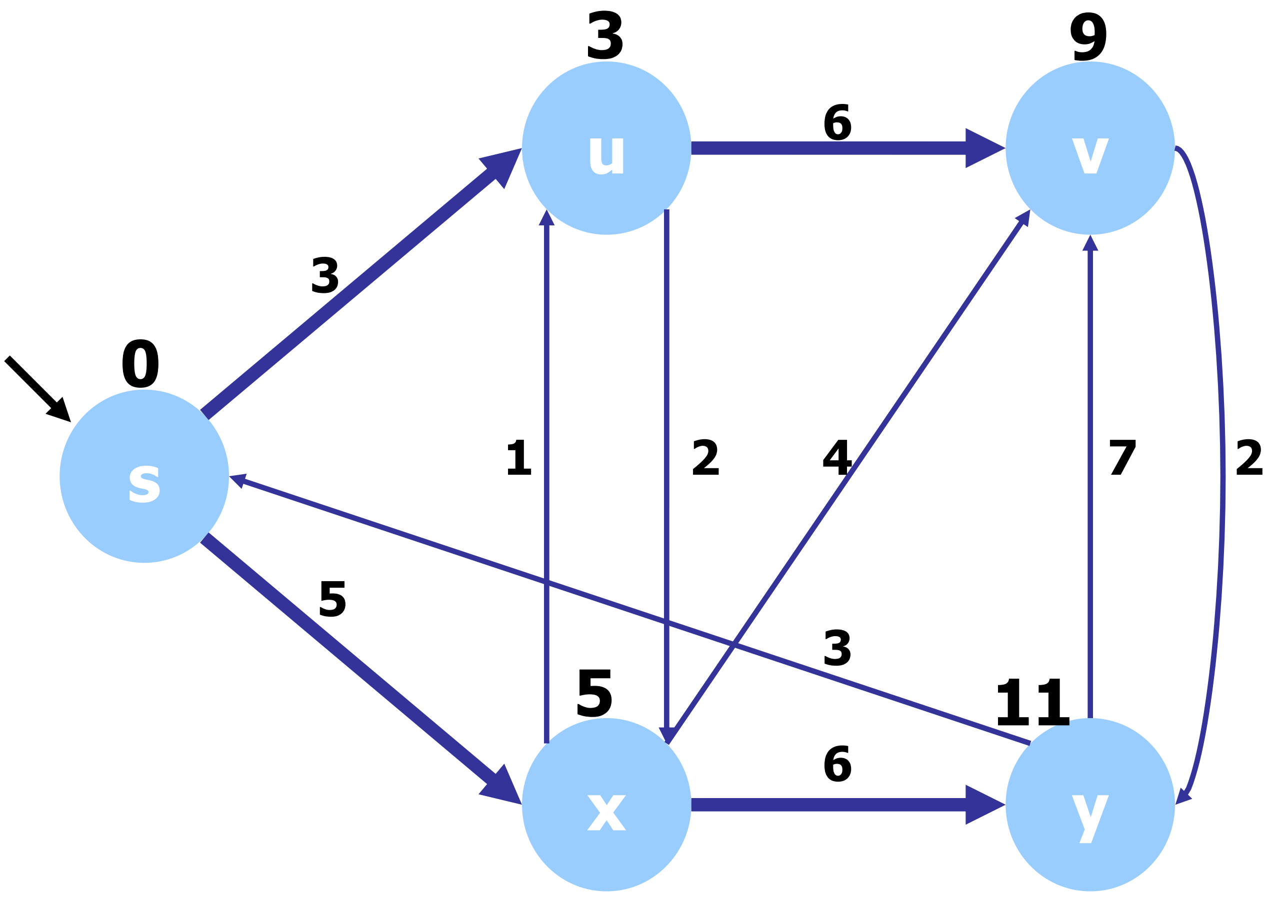
- V': insieme dei vertici raggiungibili da s
- s radice dell'albero
- $\forall v \in V'$ l'unico cammino semplice da s a v in G' è un cammino minimo da s a v in G.

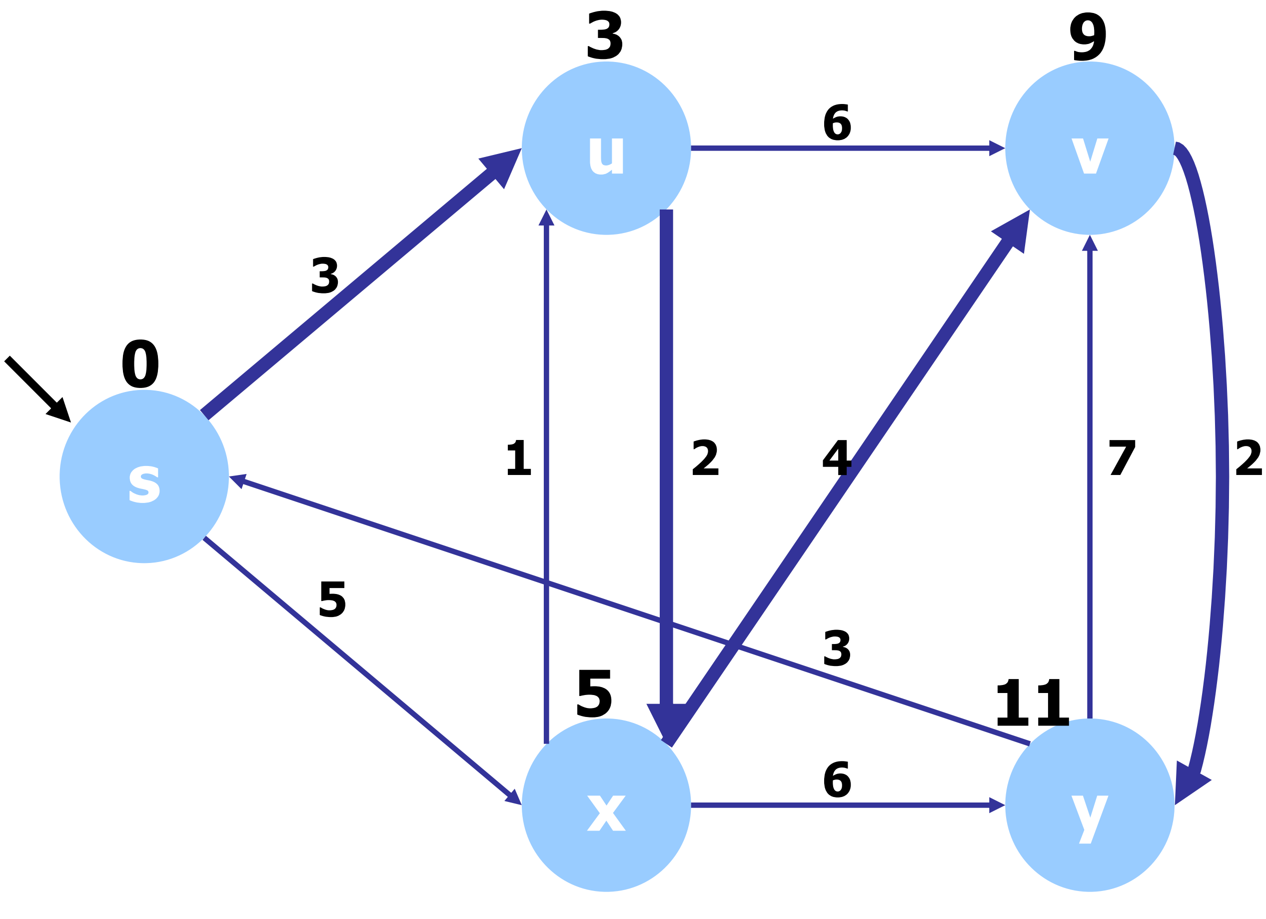
Nei grafi non pesati: algoritmo di visita in ampiezza.

Esempio




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


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


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

Fondamenti teorici

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.

Corollario:

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

Cammino minimo p da s a v decomposto in

- un sottocammino da s a u
- un arco (u,v).

Allora

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

Lemma:

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

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

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

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).

Rilassamento

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

  inizialmente:

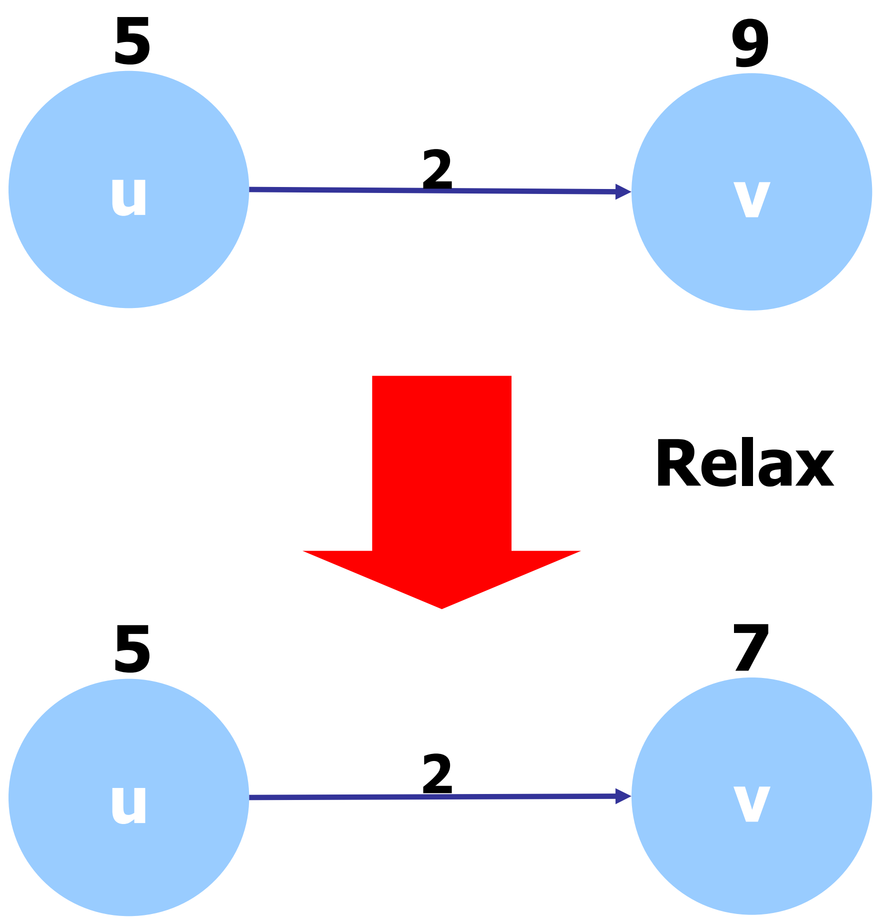
  $\forall$ v $\in$ V `wt[v]= maxWT, st[v] = -1;`

  `wt[s] = 0;`

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

```
if (wt[v]>wt[u]+e.wt) {
    wt[v] = wt[u]+e.wt;
    st[v] = u;
}
```

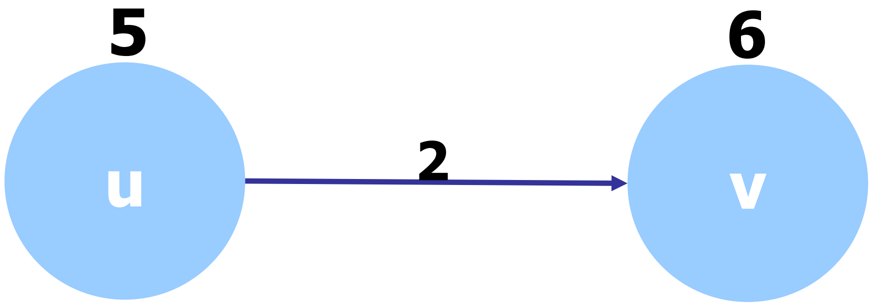
Esempio




```
wt[v] =9
wt[u] = 5
e.wt = 2
wt[v] > wt[u] + e.wt
```

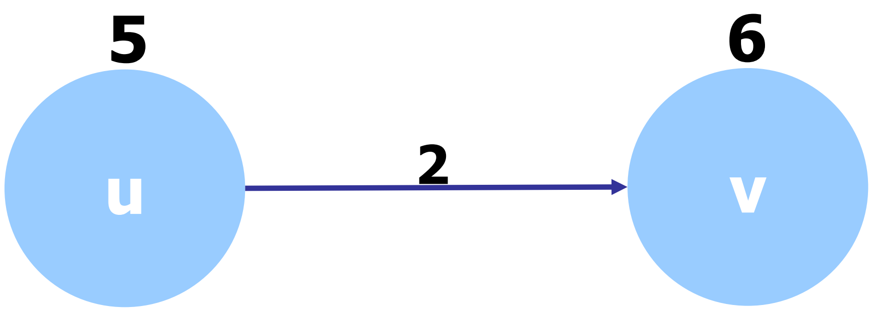
```
wt[v] = 7
st[v] = u
```

cammino minimo da s a v = cammino minimo da s a u + arco (u,v)




```
wt[v] =6
wt[u] = 5
e.wt = 2
wt[v] < wt[u] + e.wt
```

Relax



Il rilassamento non ha avuto effetto.

Proprietà

Lemma:

G=(V,E): grafo orientato, pesato w: E→***R***.

e = (u,v) $\in$ E

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

`wt[v] ≤ wt[u] + e.wt`

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

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

Lemma:

G=(V,E): grafo orientato, pesato w: E→***R***.

sorgente s ∈ V

inizializzazione di `wt` e `st`

∀ v ∈ V `wt[v]` ≥ δ(s,v)

- per tutti i passi di rilassamento sugli archi
- quando `wt[v]` = δ(s,v), allora `wt[v]` non cambia più

Lemma:

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

sorgente $s \in V$

cammino minimo da s a v composto da

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

inizializzazione di `wt` e `st`

applicazione del rilassamento su e= (u,v)

se prima del rilassamento `wt[u]` $= \delta(s,u)$

dopo il rilassamento `wt[v]` $= \delta(s,v)$.

Applicazione

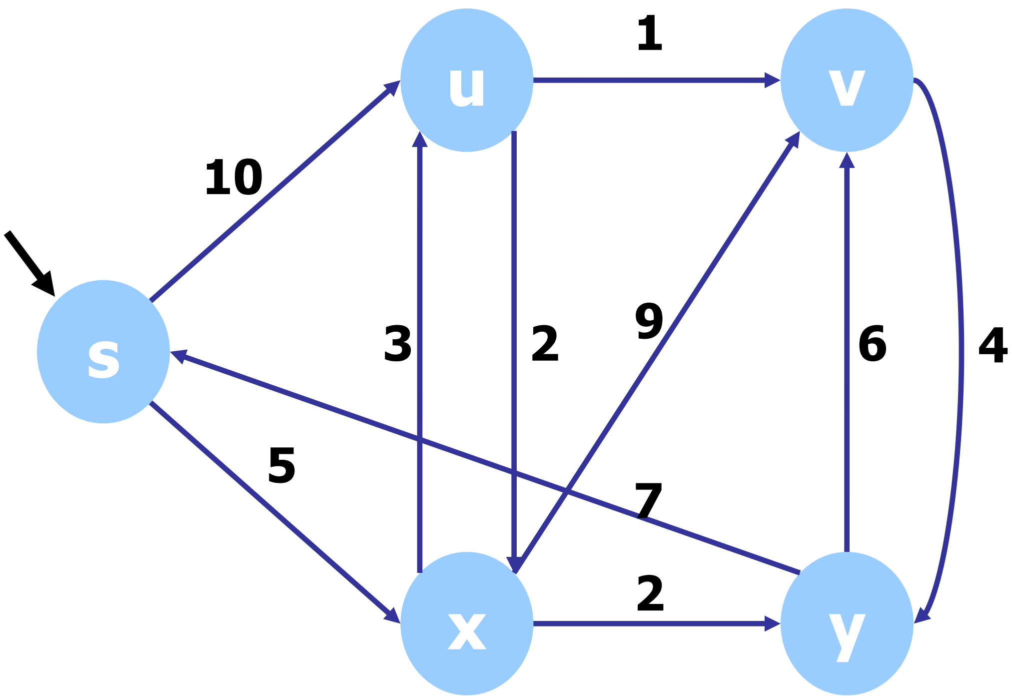
Rilassamento:

- 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 di 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 (`wt[u]` minimo)
  - inserisce u in S
  - rilassa tutti gli archi uscenti da u.

Esempio



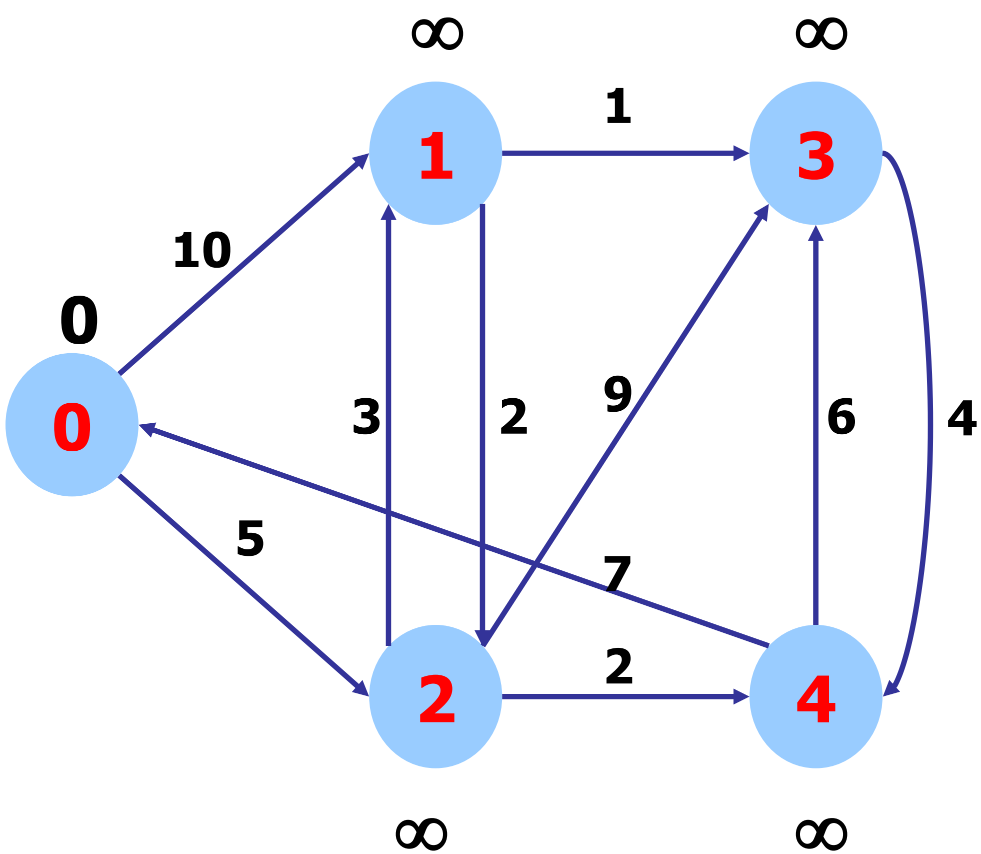
ST

0	s
1	u
2	x
3	v
4	y

S={}

PQ={s/0, u/∞,v/∞, x/∞, y/∞}

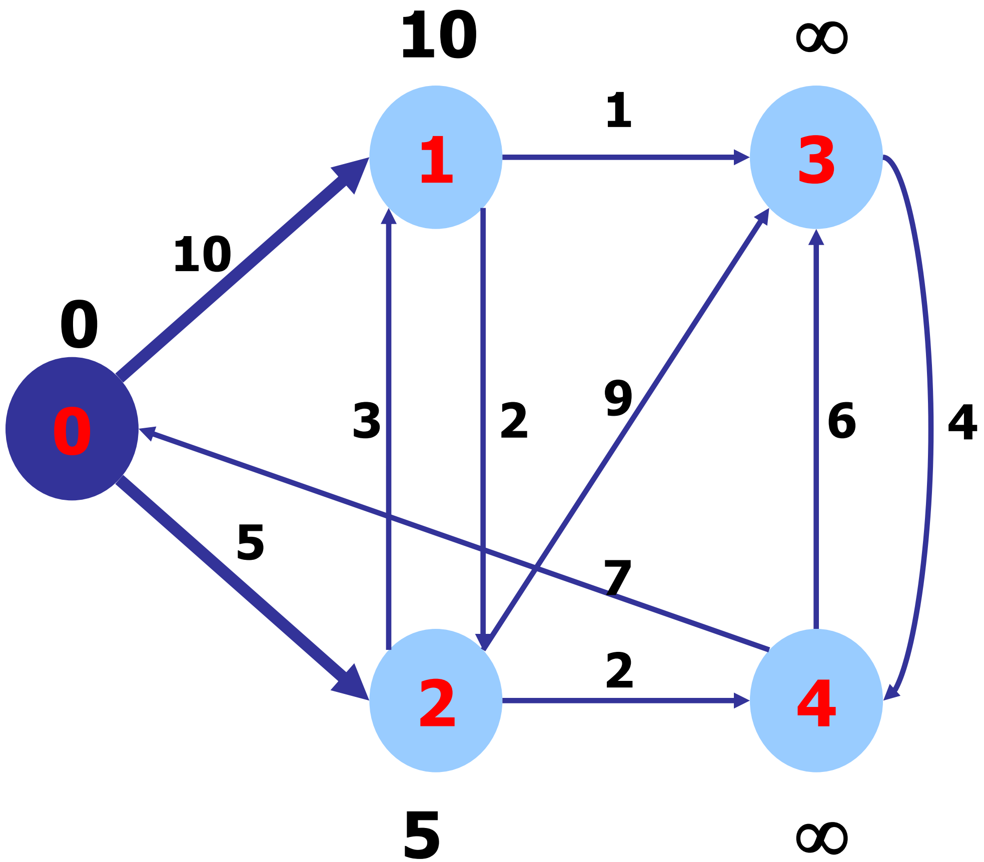
Coda a priorità visualizzata per semplicità come vettore.
I nodi compaiono con il loro nome originale per leggibilità



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

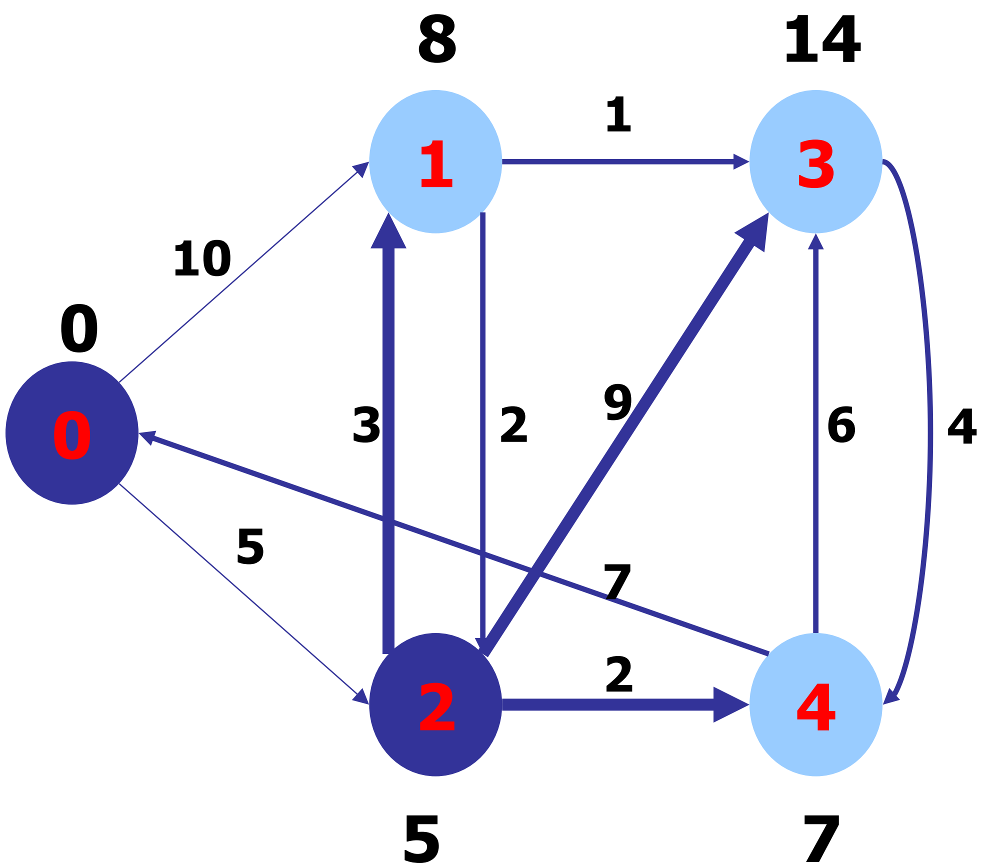
mindist	0	∞	∞	∞	∞
	0	1	2	3	4

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



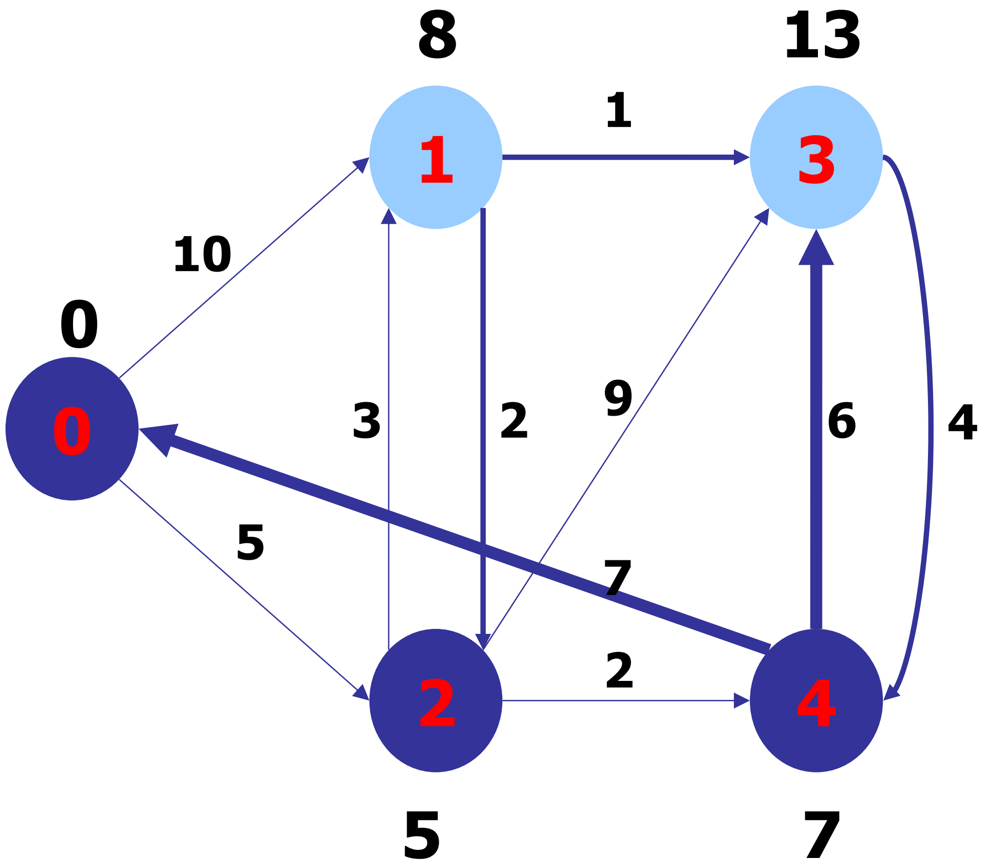
st	0	0	0	-1	-1
	0	1	2	3	4
mindist	0	10	5	∞	∞
	0	1	2	3	4

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
mindist	0	8	5	14	7
	0	1	2	3	4

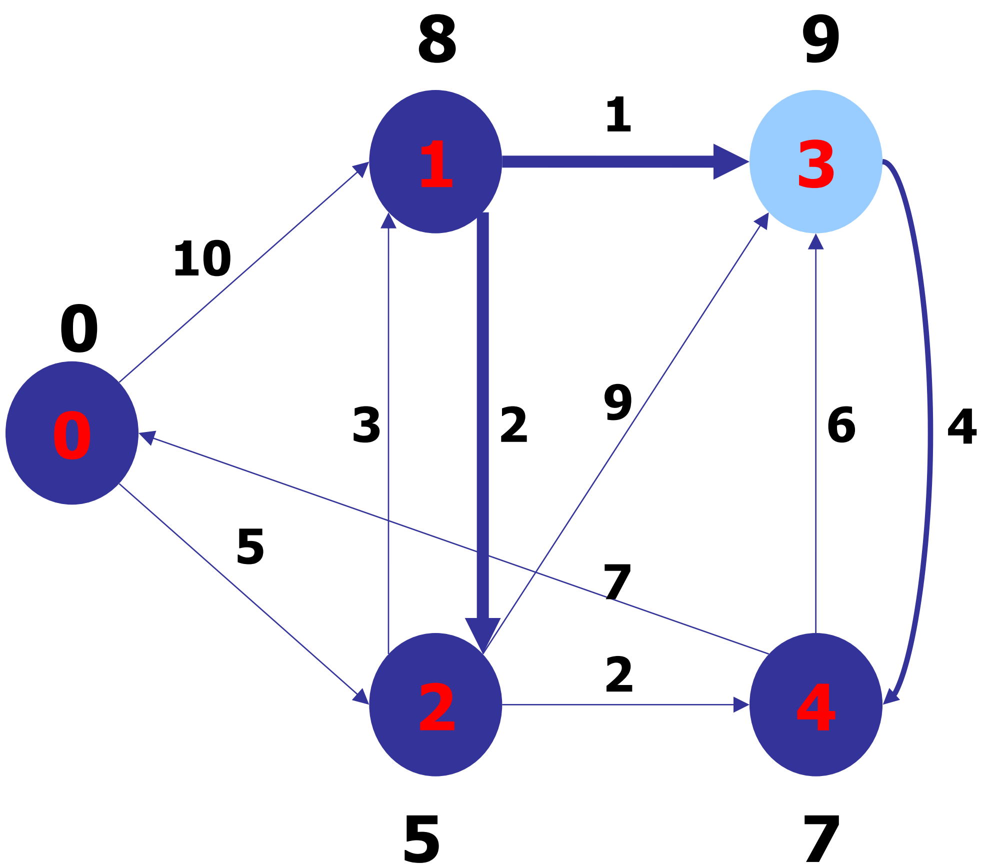
S={s, x, y}
relax (y,s), (y,v)
PQ={u/8,v/13}



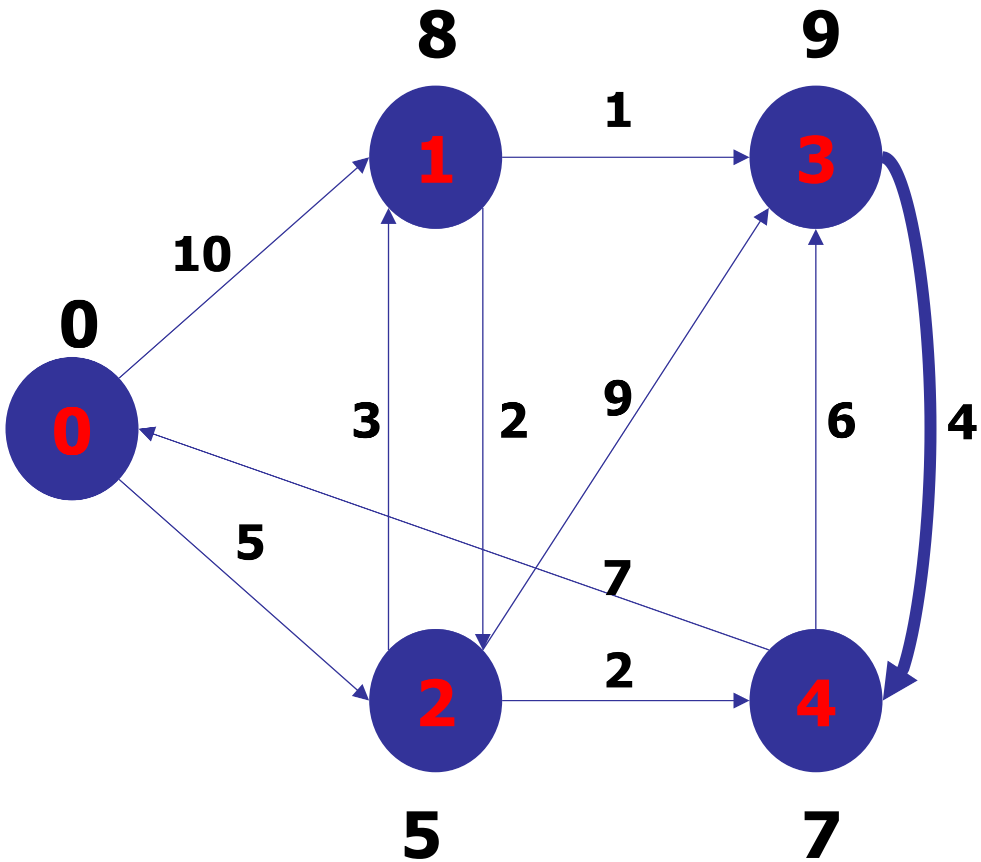
st	0	2	0	4	2
	0	1	2	3	4

mindist	0	8	5	13	7
	0	1	2	3	4

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



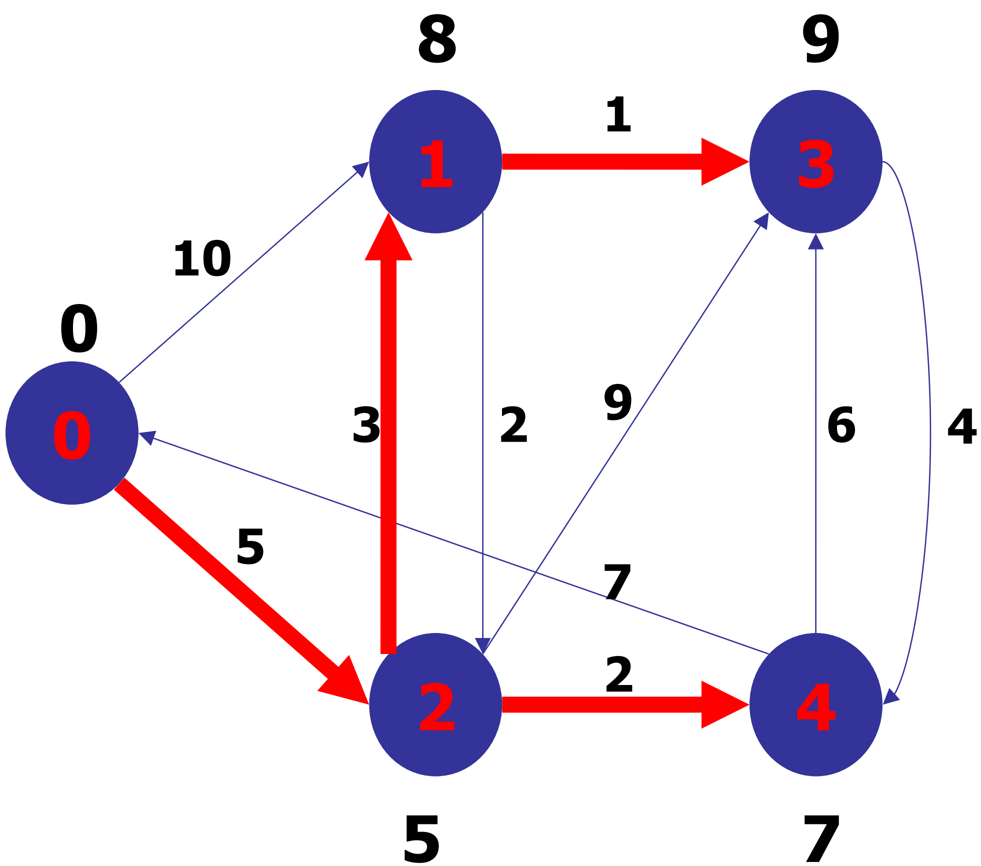
st	0	2	0	1	2
	0	1	2	3	4
mindist	0	8	5	9	7
	0	1	2	3	4



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

st	0	2	0	1	2
	0	1	2	3	4
mindist	0	8	5	9	7
	0	1	2	3	4

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



	0	1	2	3	4
st	0	2	0	1	2

	0	1	2	3	4
mindist	0	8	5	9	7

```
void GRAPHspD(Graph G) {
  int v, w; link t;
  char name[MAX];
  PQ pq = PQinit(G->V);
  int *st, *mindist;
  st = malloc(G->V*sizeof(int));
  mindist = malloc(G->V*sizeof(int));

  printf("Insert start node: "); scanf("%s", name);
  int s = STsearch(G->tab, name);
  if (s == -1) { printf("Node doesn't exist\n");  return; }

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

PQ con priorità in mindist

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

Complessità

$\Theta(|V|)$

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

  $O(\lg|V|)$

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

$O(\lg|V|)$

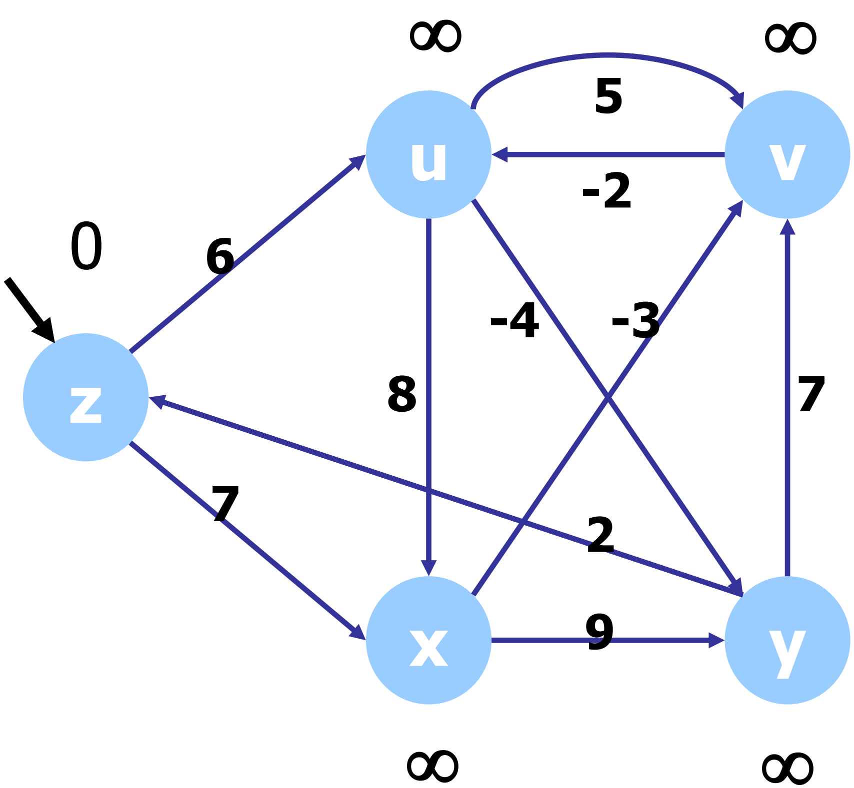
$O(|E|)$

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

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

Dijkstra e grafi con pesi negativi

- ∃ archi a peso negativo
- ∄ ciclo a peso negativo



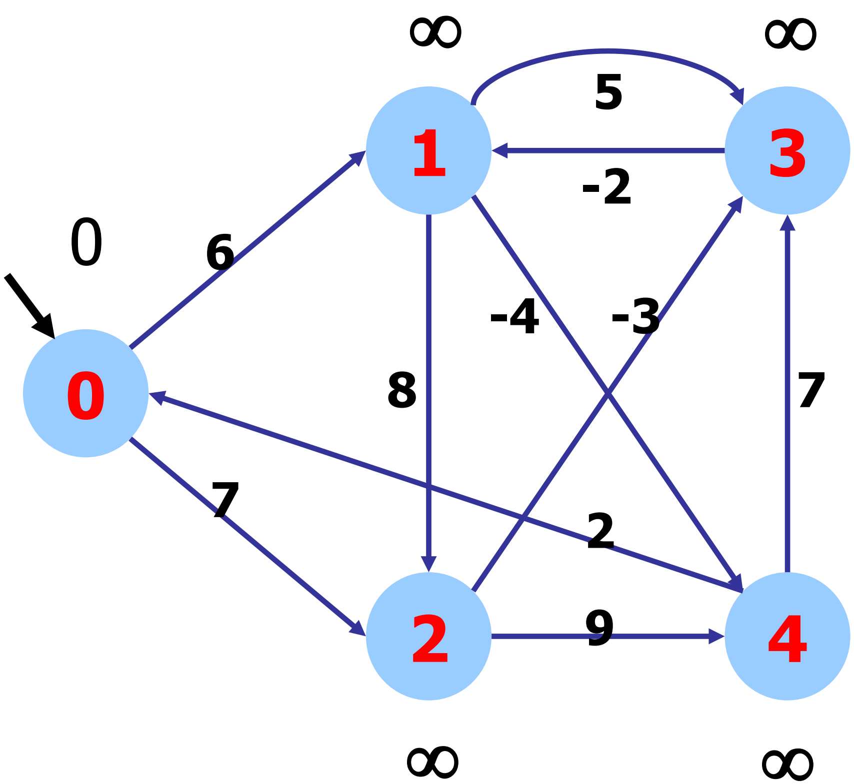
ST

0	z
1	u
2	x
3	v
4	y

Dijkstra e grafi con pesi negativi

$\exists$ archi a peso negativo

$\nexists$ ciclo a peso negativo



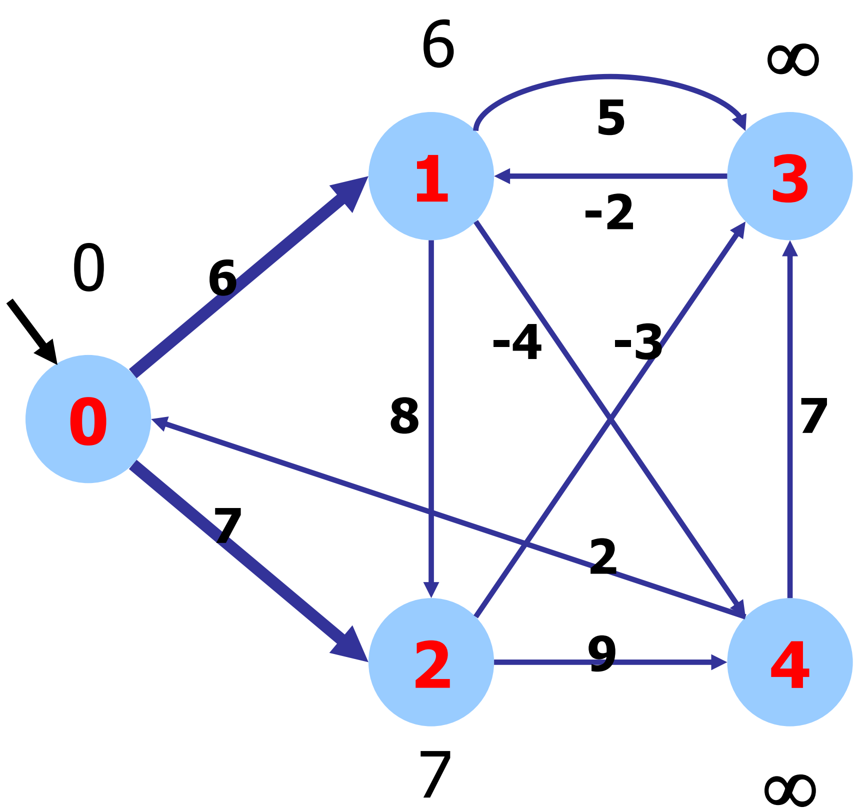
S={}

PQ={z/0, u/∞, v/∞, x/∞, y/∞}

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

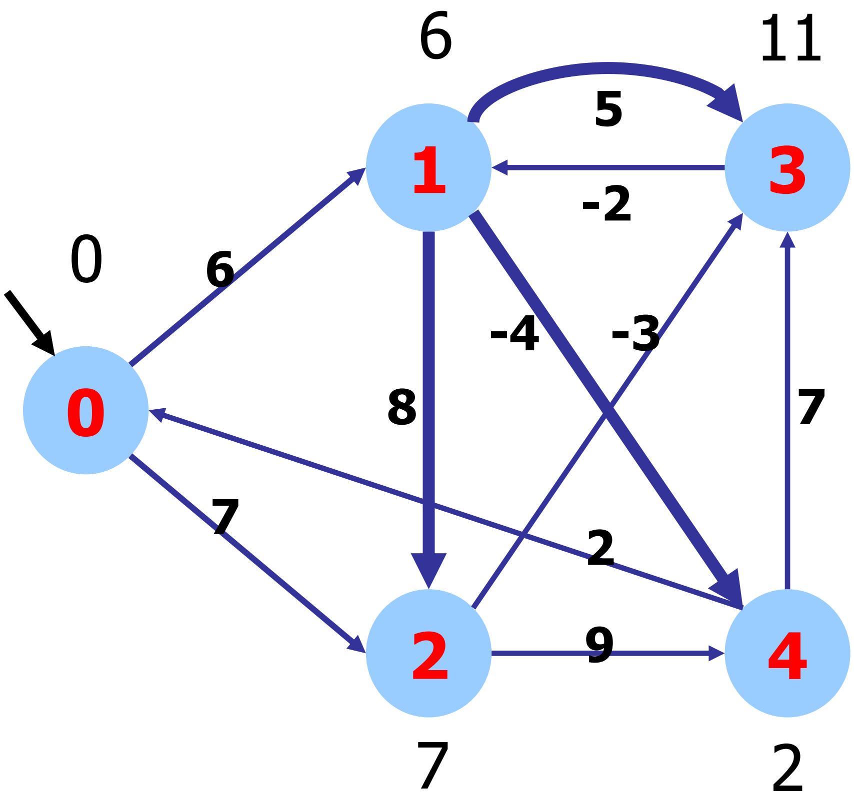
mindist	0	∞	∞	∞	∞
	0	1	2	3	4

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



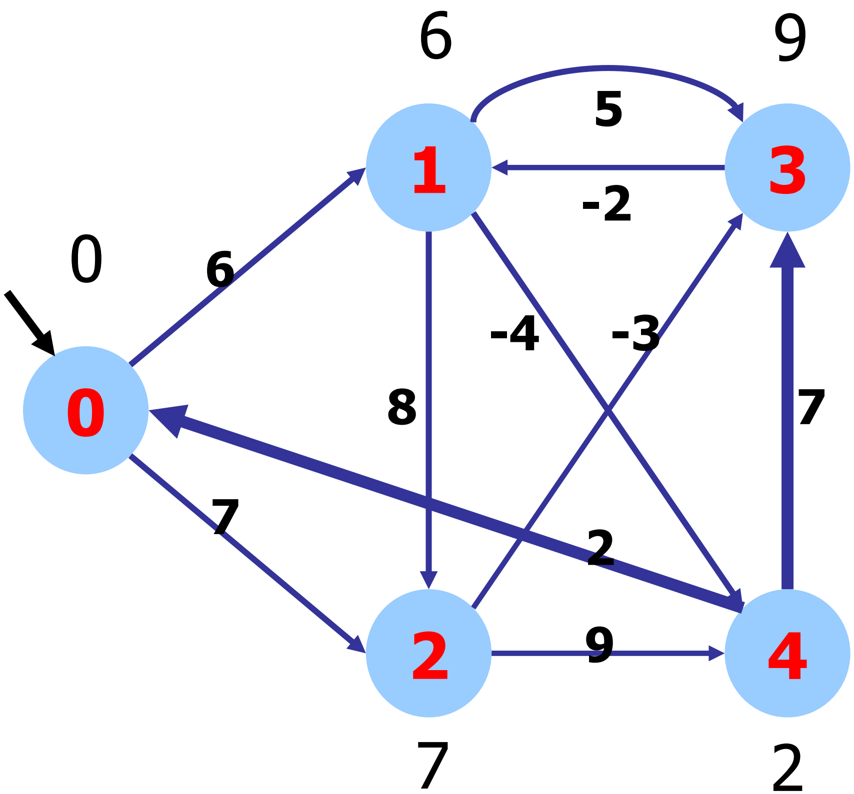
S={z}
relax (z,u), (z,x)
PQ={u/6, x/7, v/∞, y/∞}

st	0	0	0	-1	-1
	0	1	2	3	4
mindist	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}

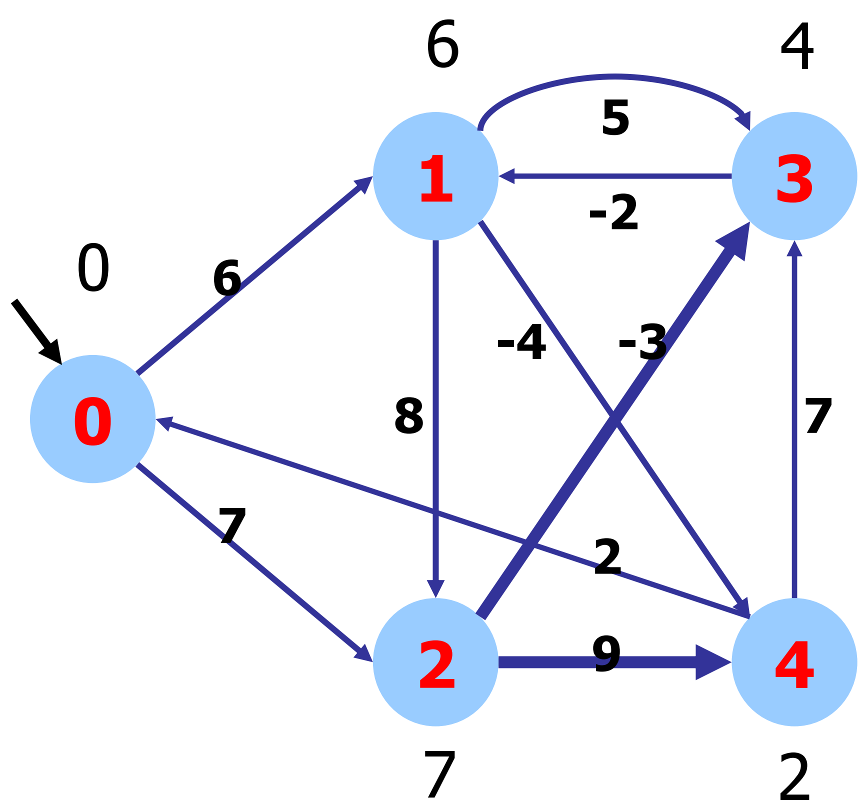
	0	1	2	3	4
st	0	0	0	1	1
mindist	0	6	7	11	2



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

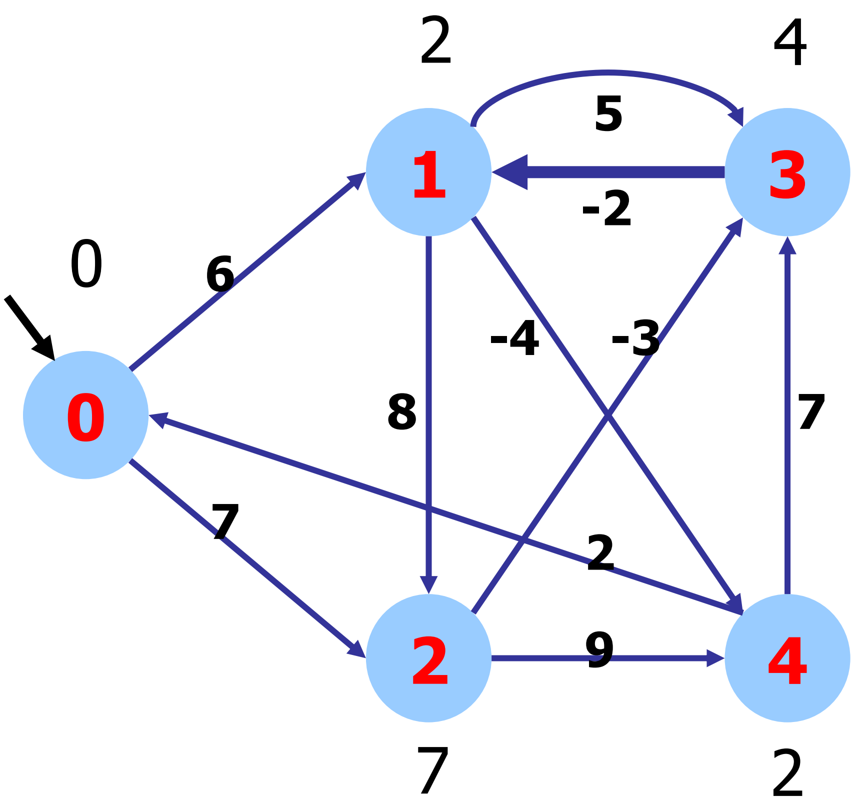
st	0	0	0	4	1
	0	1	2	3	4

mindist	0	6	7	9	2
	0	1	2	3	4



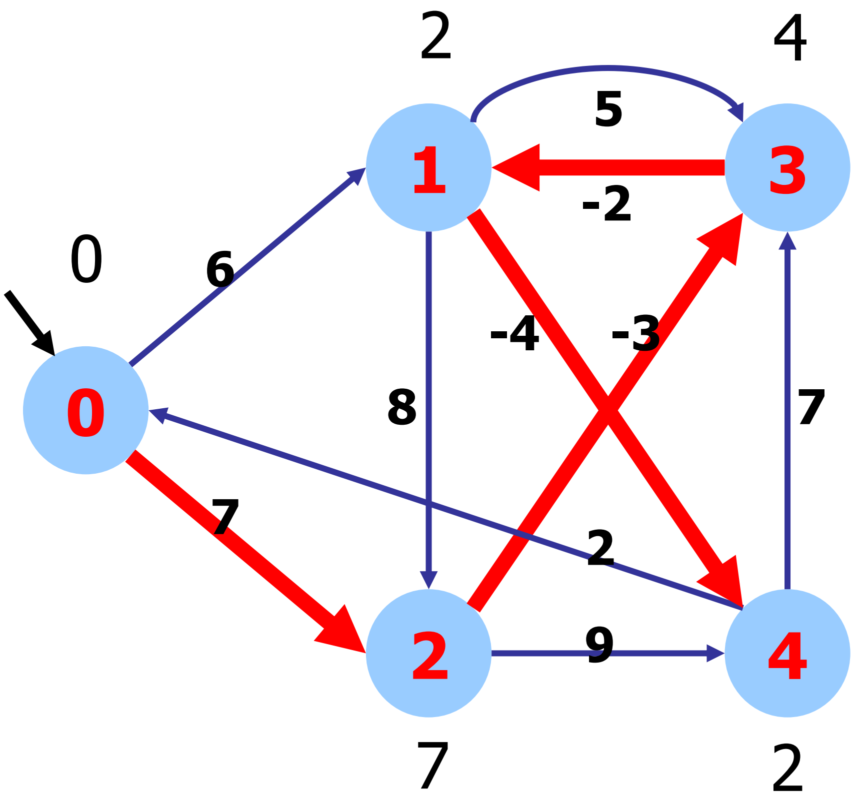
S={z, u, y, x}
relax (x,v), (x,y)
PQ={v/4}

	0	1	2	3	4
st	0	0	0	2	1
mindist	0	6	7	4	2



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

	0	1	2	3	4
st	0	3	0	2	1
mindist	0	2	7	4	2



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

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

mindist					
mindist	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).

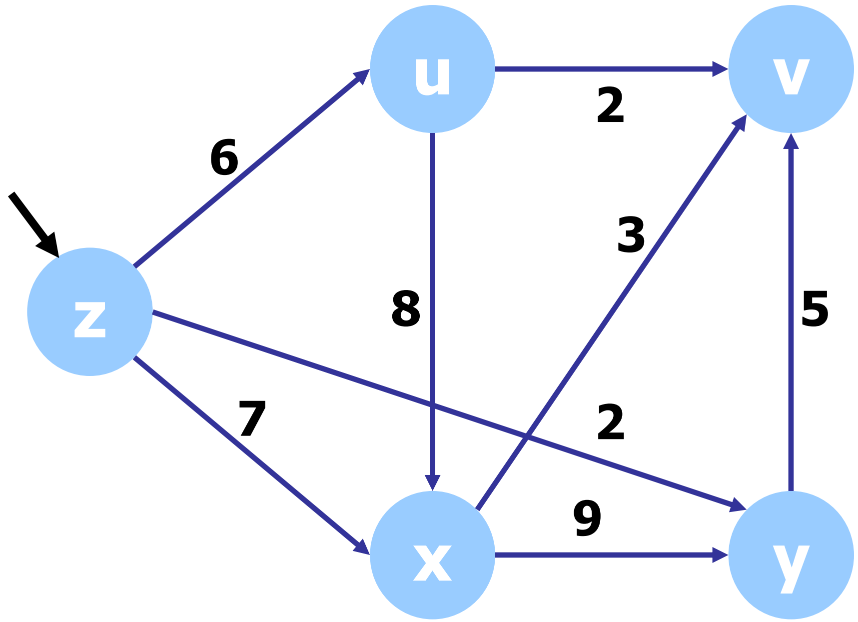
Cammini minimi su DAG pesati

L’assenza di cicli semplifica l’algoritmo:

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

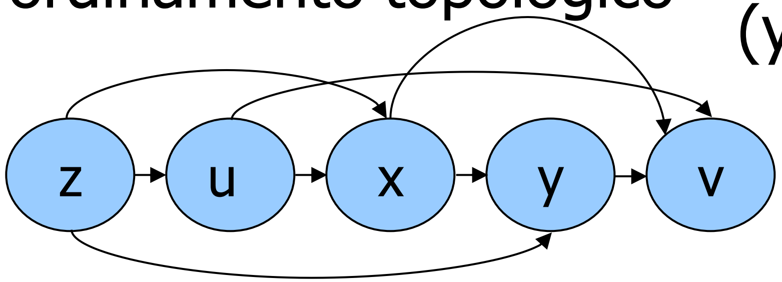
Esempio

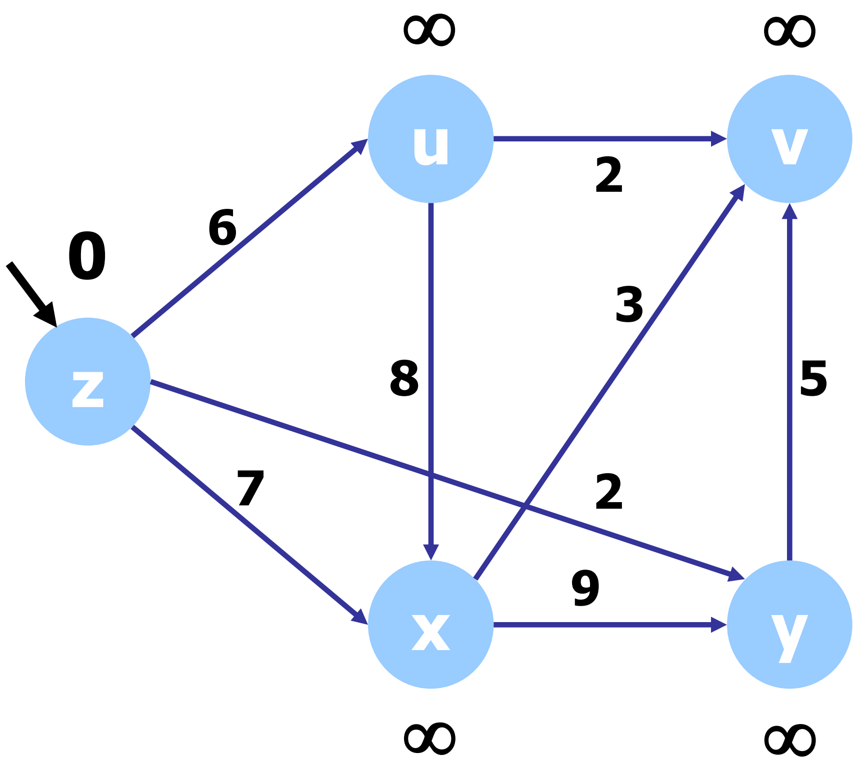
I nodi compaiono con il loro nome originale per leggibilità



Ordine di applicazione della relaxation
(z, u)
(z, x)
(z, y)
(u, v)
(u, x)
(x, v)
(x, y)
(y, v)

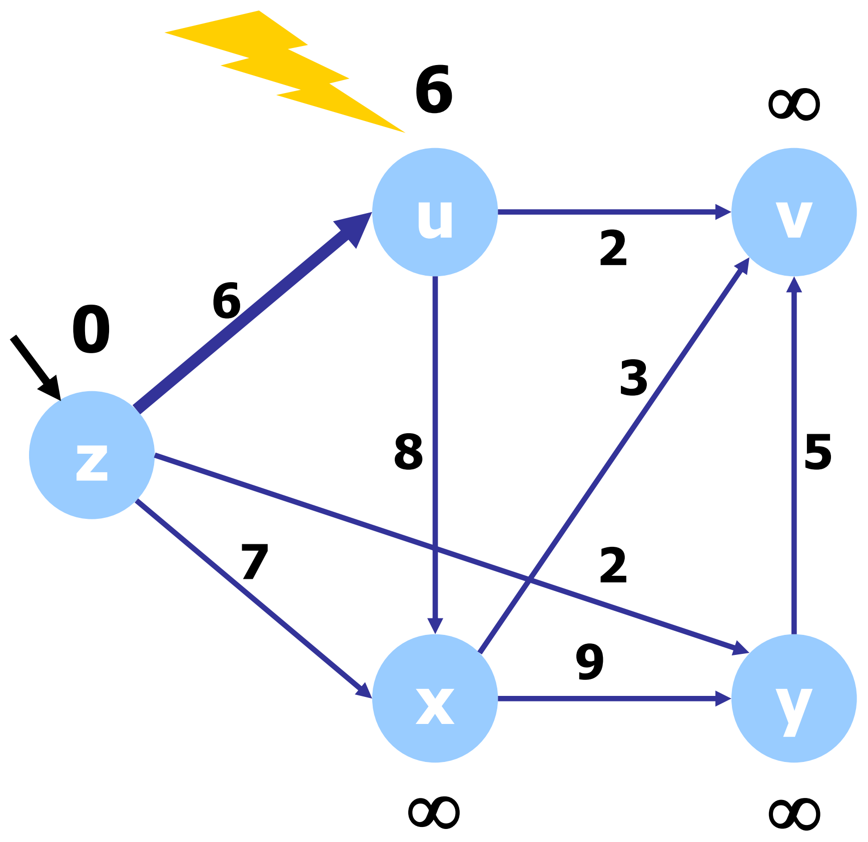
ordinamento topologico





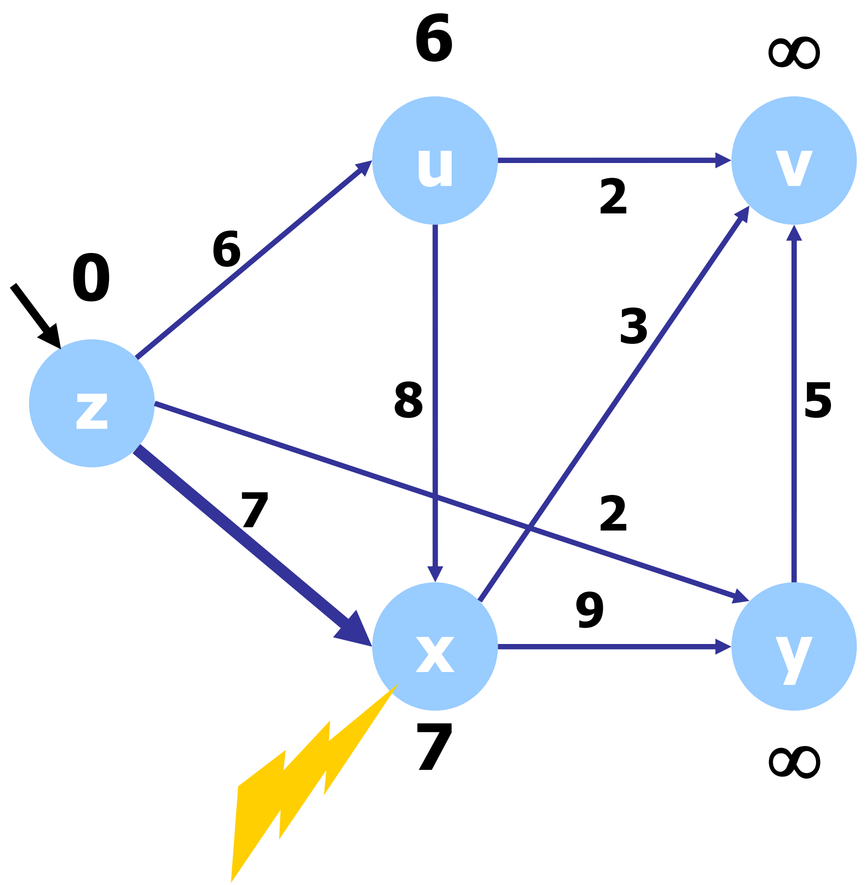
Ordine di applicazione della relaxation

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



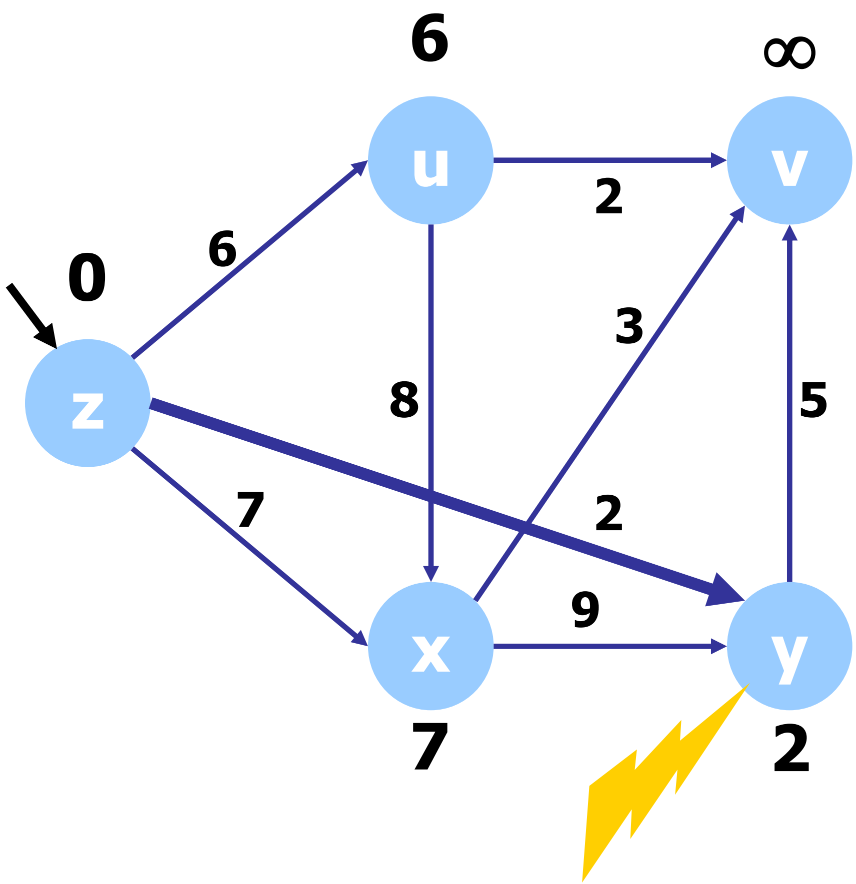
Ordine di applicazione della relaxation

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



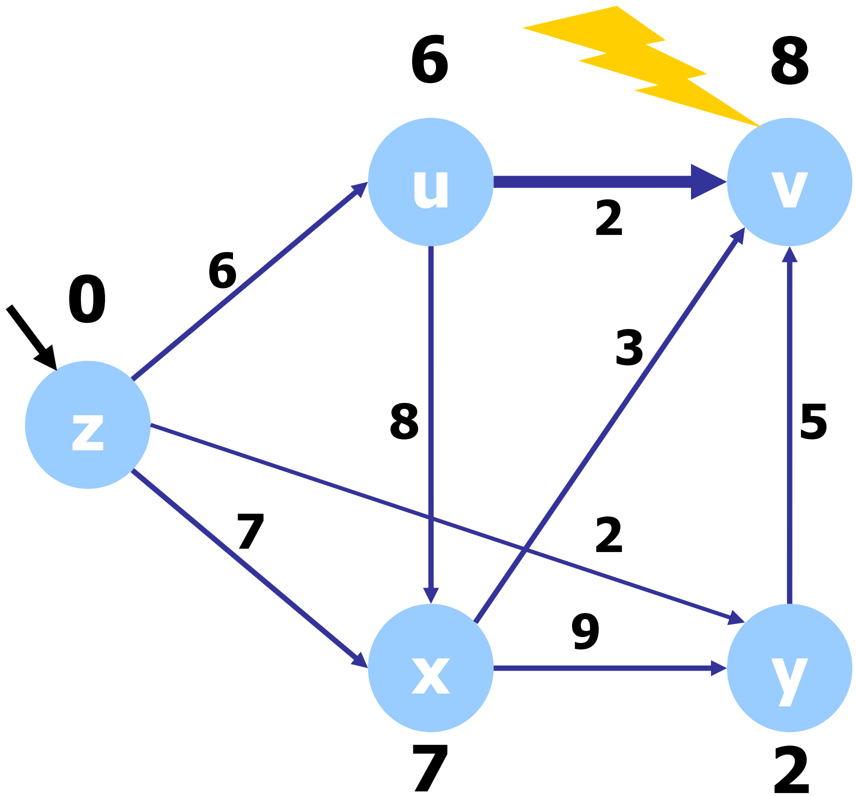
Ordine di applicazione della relaxation

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



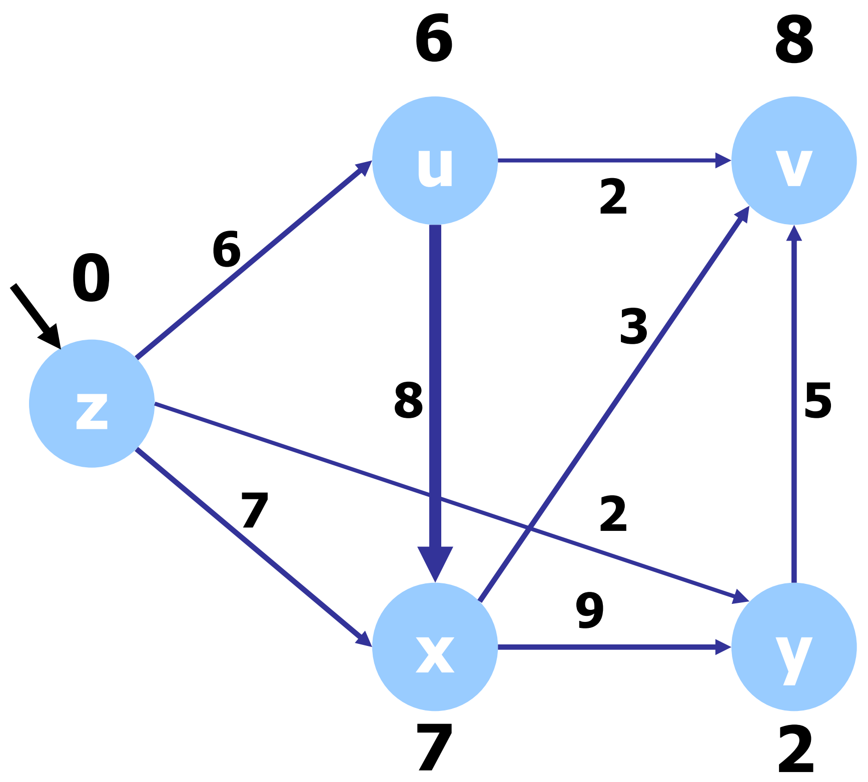
Ordine di applicazione della relaxation

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



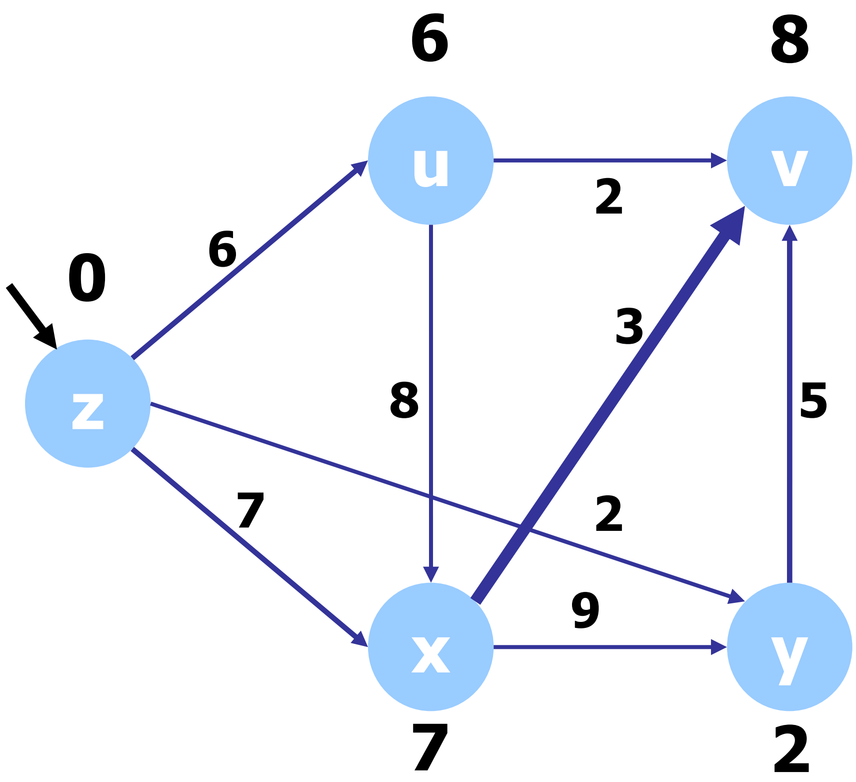
Ordine di applicazione della relaxation

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



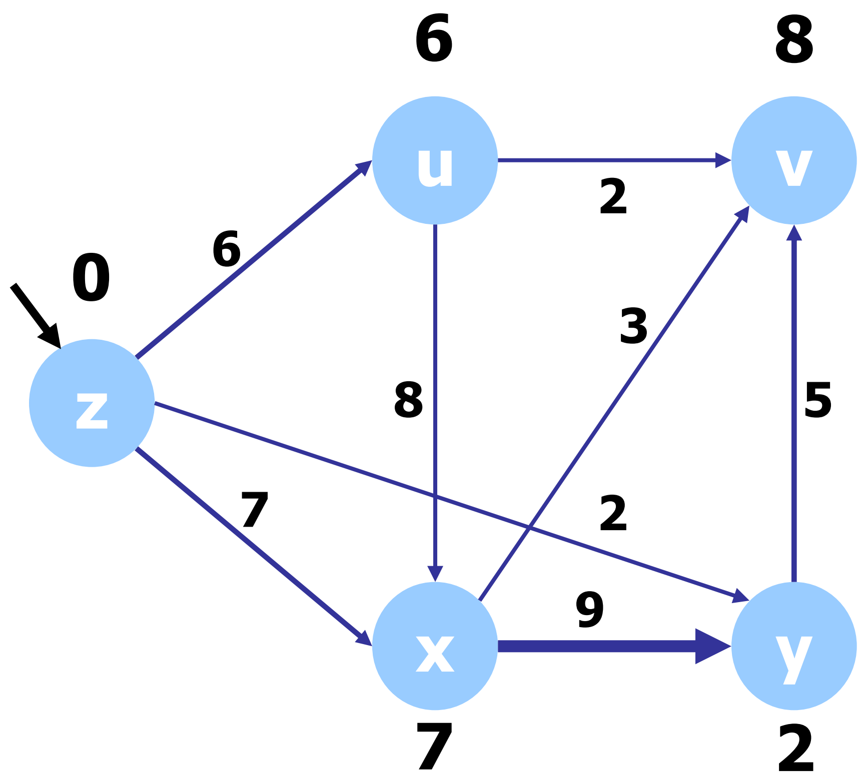
Ordine di applicazione della relaxation

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



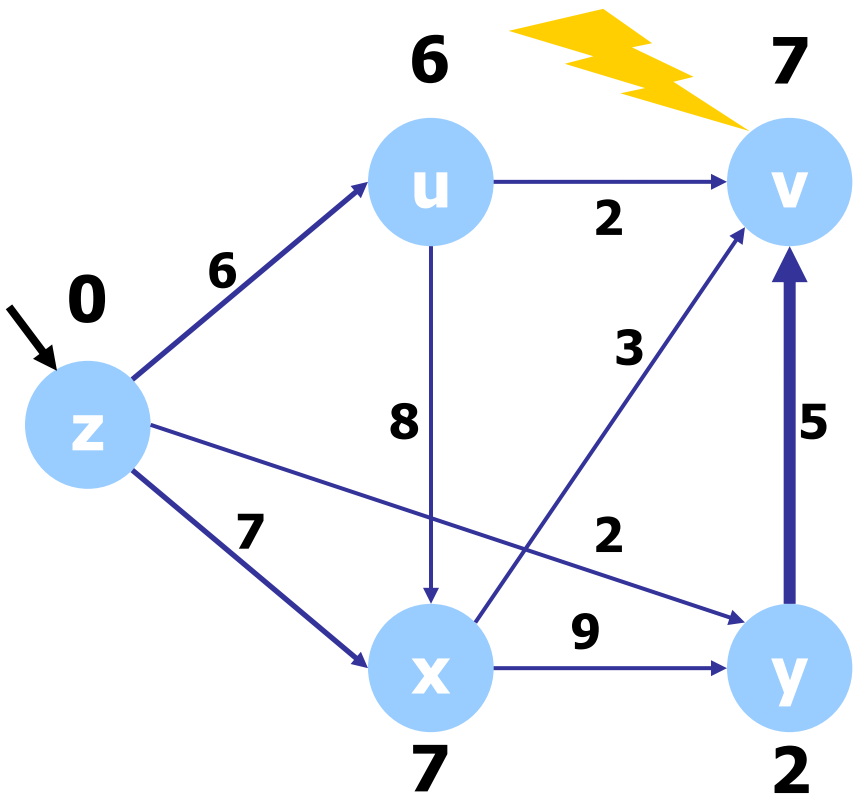
Ordine di applicazione della relaxation

(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).

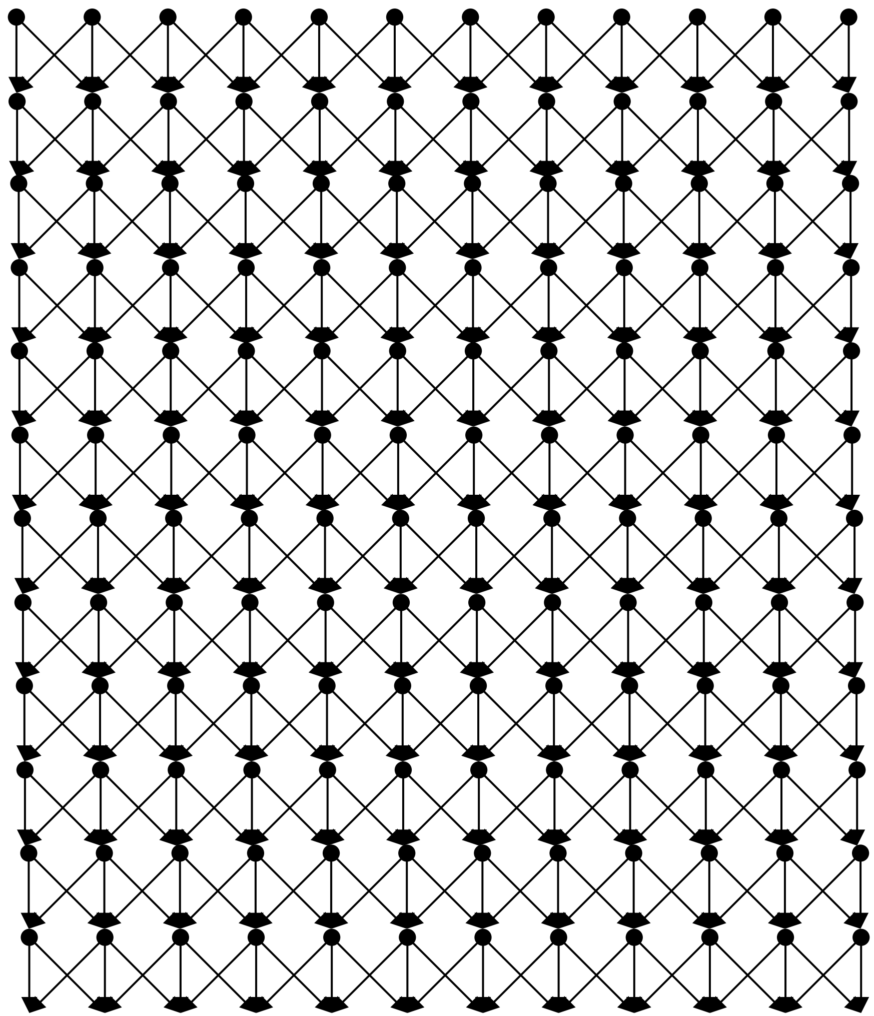
Modello: immagine come DAG pesato di pixel.

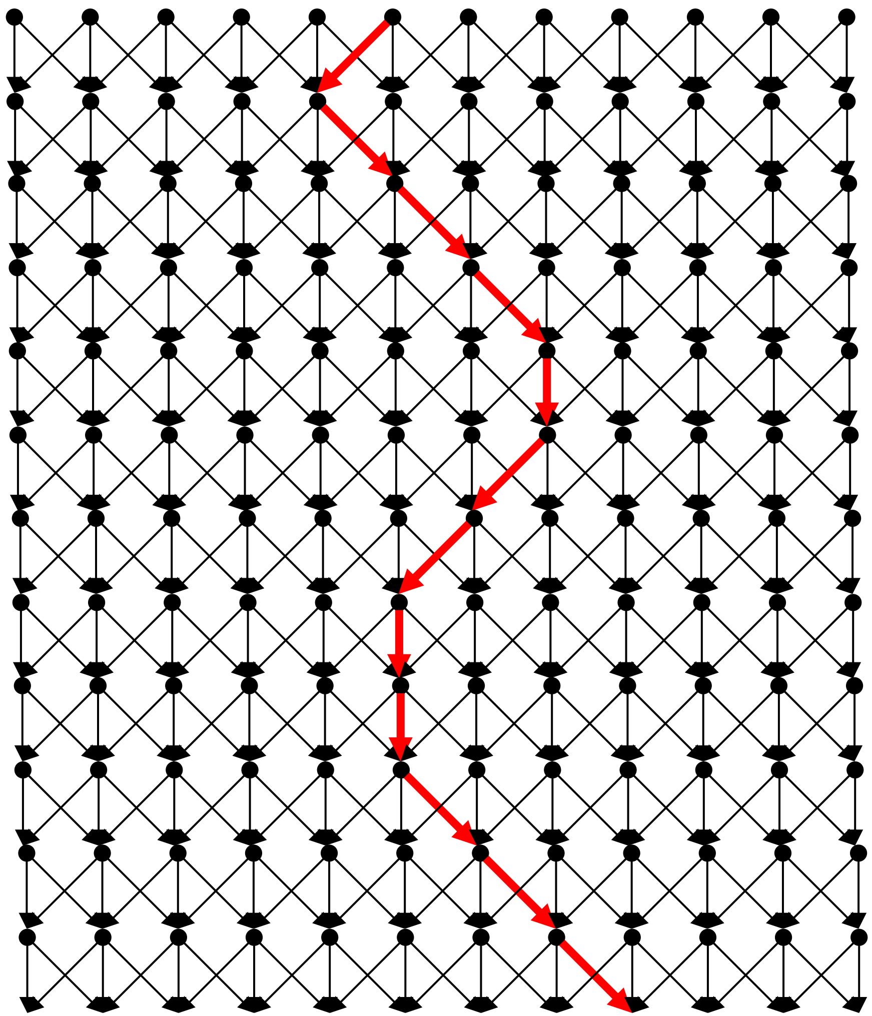
Peso dell'arco: misura del contrasto tra 2 pixel.

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

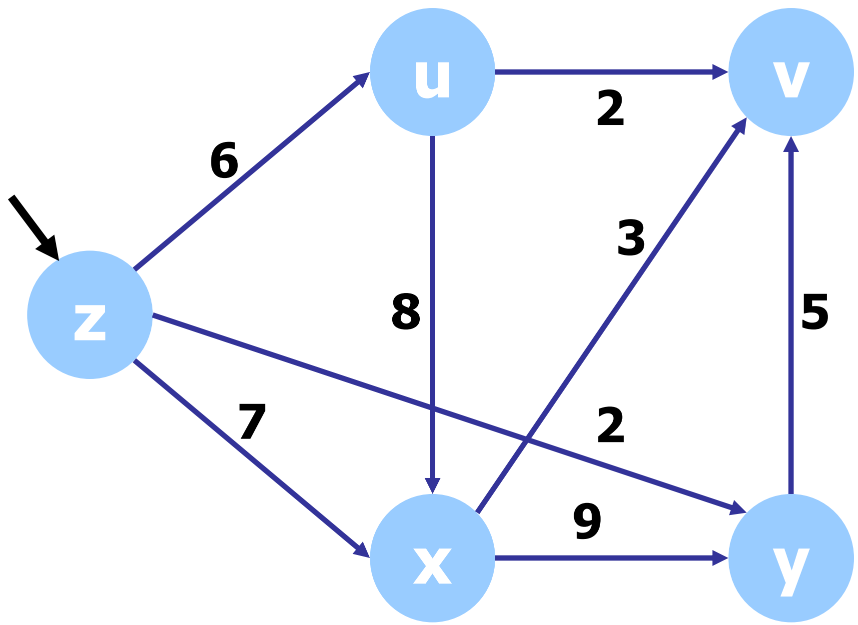
Problema non trattabile su grafi pesati qualsiasi.

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

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

```
if (wt[v]< wt[u] + e.wt) {
   wt[v] = wt[u] + e.wt;
   st[w] = v;
}
```

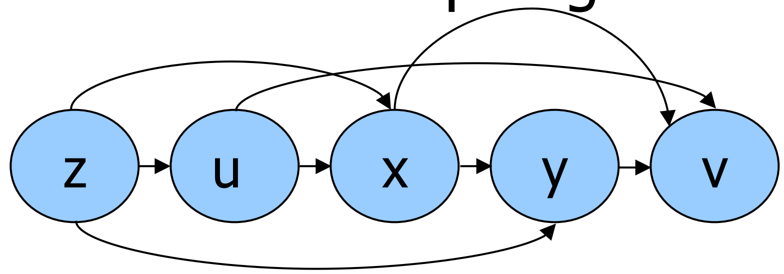
Esempio

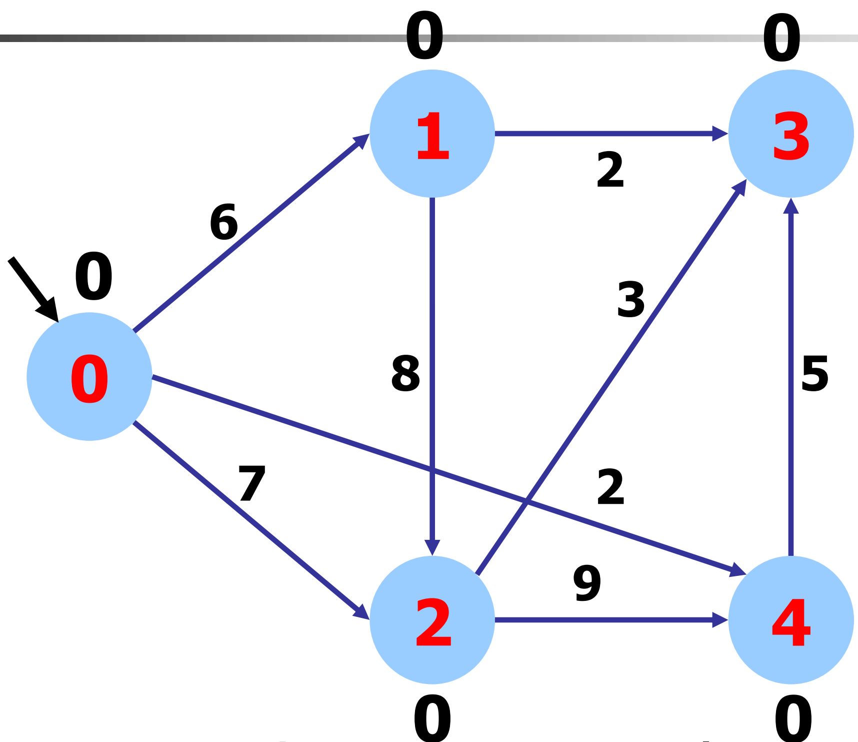


ST

0	z
1	u
2	x
3	v
4	y

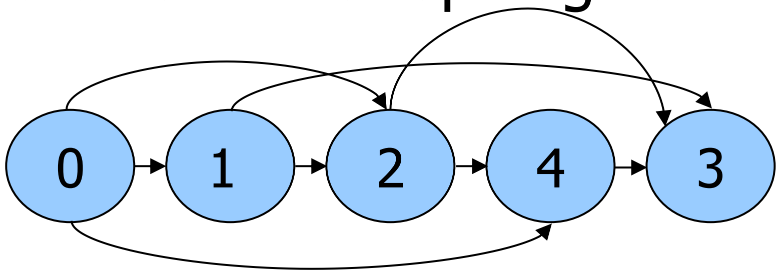
ordinamento topologico

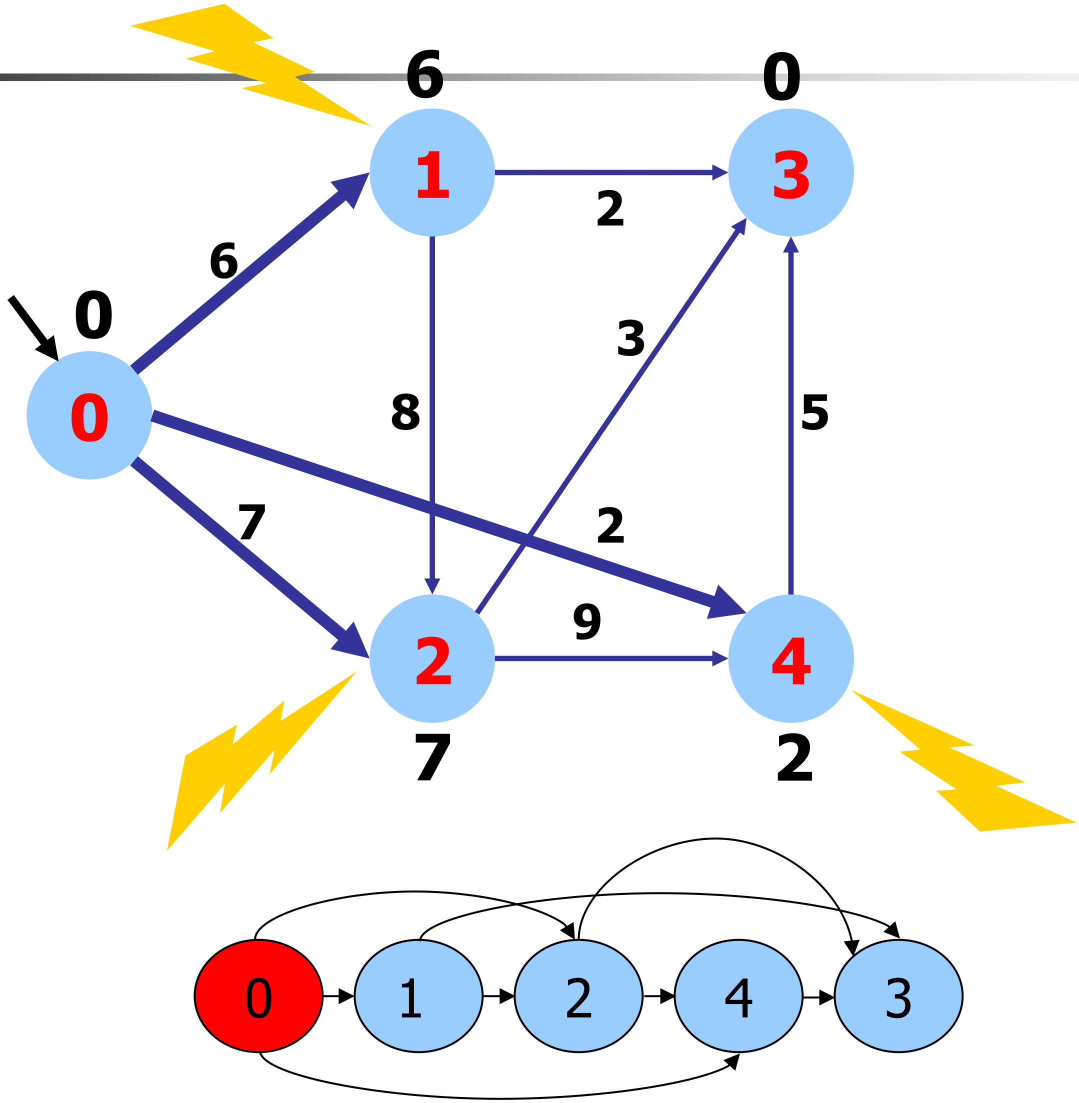


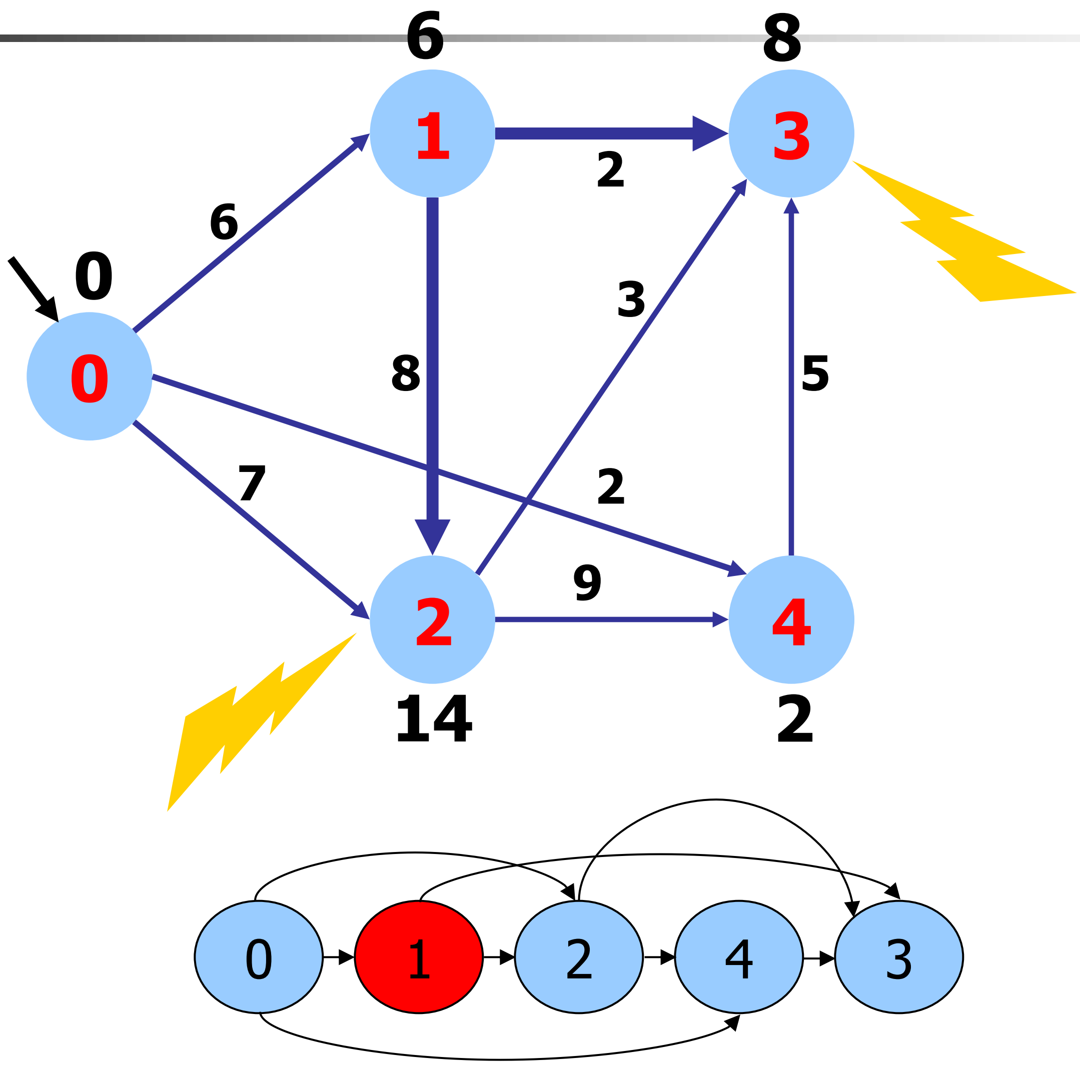


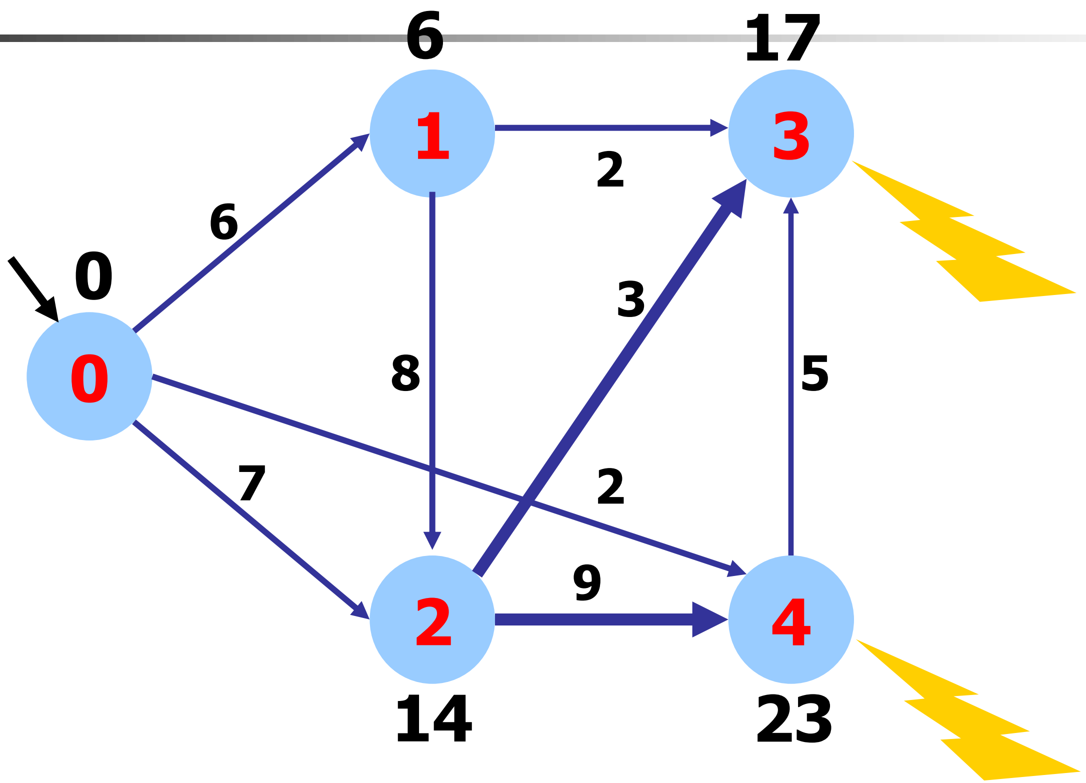
Notare le stime iniziali a 0!

ordinamento topologico

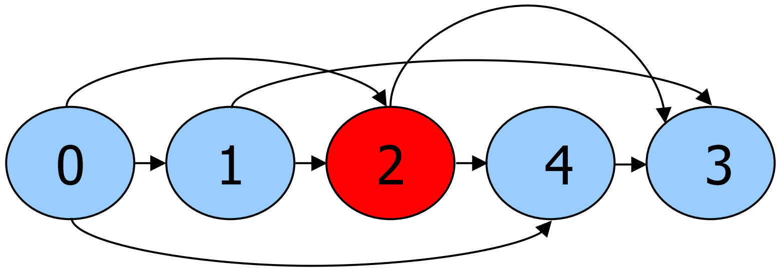



6
0
1
3
2
6
0
3
8
5
0
7
2
9
2
4
7
2
0
1
2
4
3

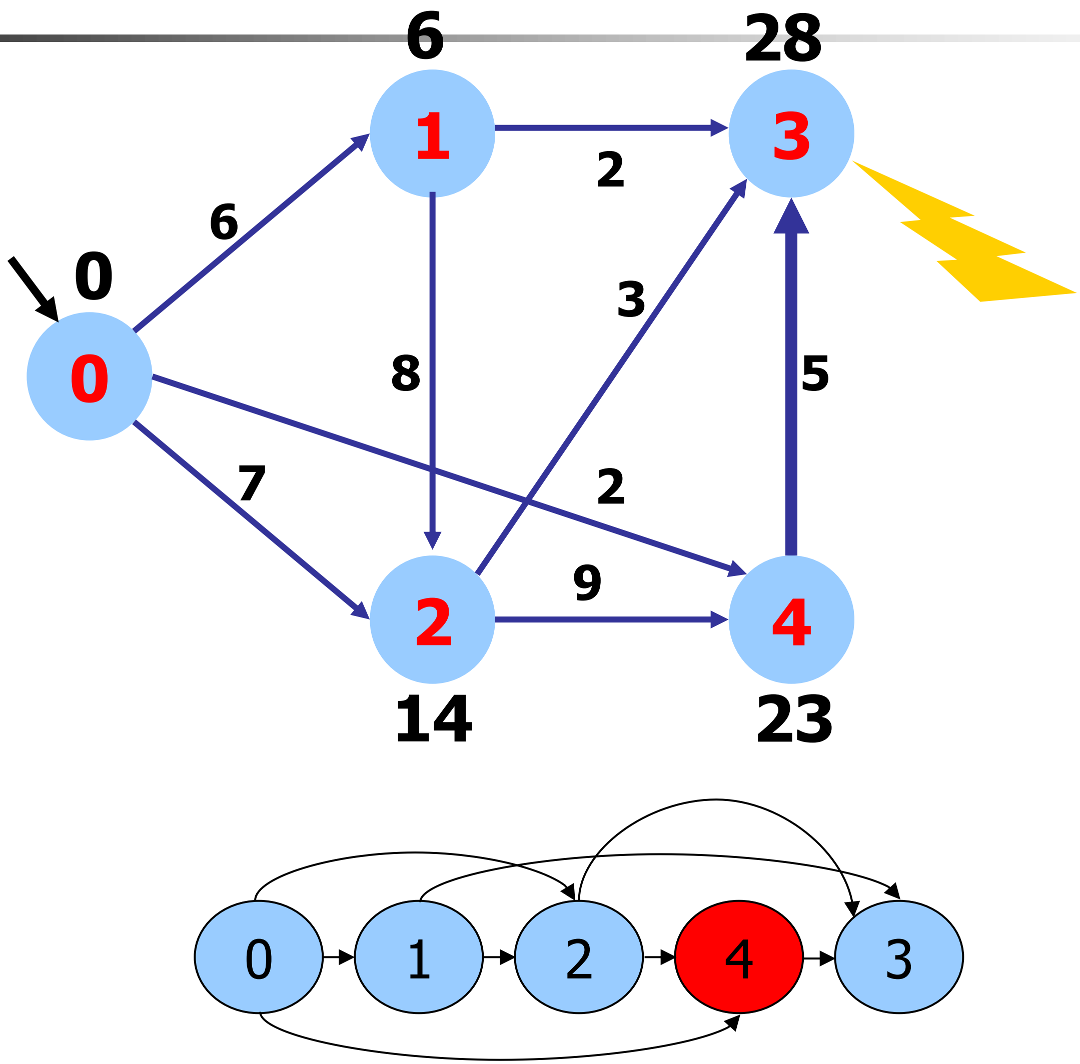

6
8
1
3
2
6
0
3
8
5
0
7
2
9
2
4
14
2
0
1
2
4
3

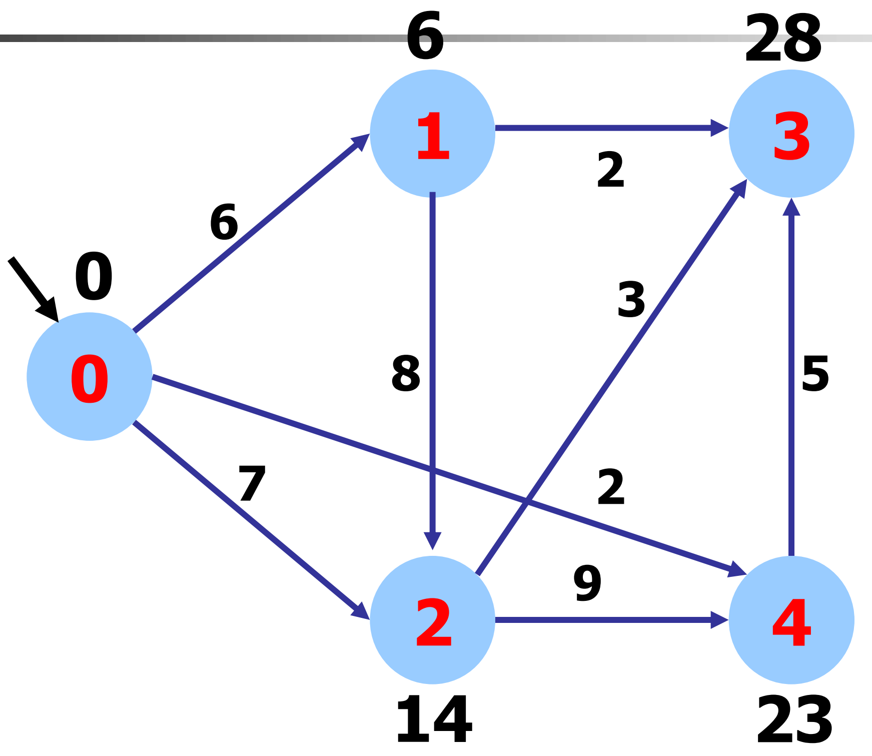


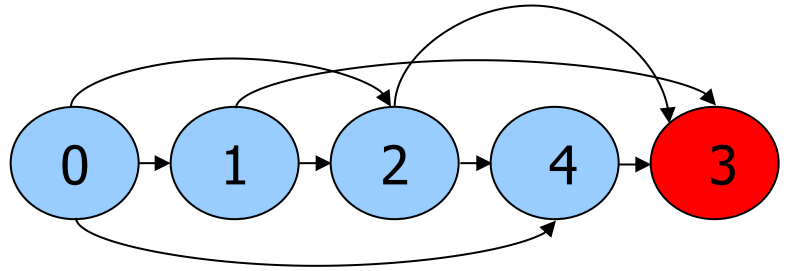
6
17
1
3
2
6
0
3
8
5
0
7
2
9
2
4
14
23



0
1
2
4
3


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

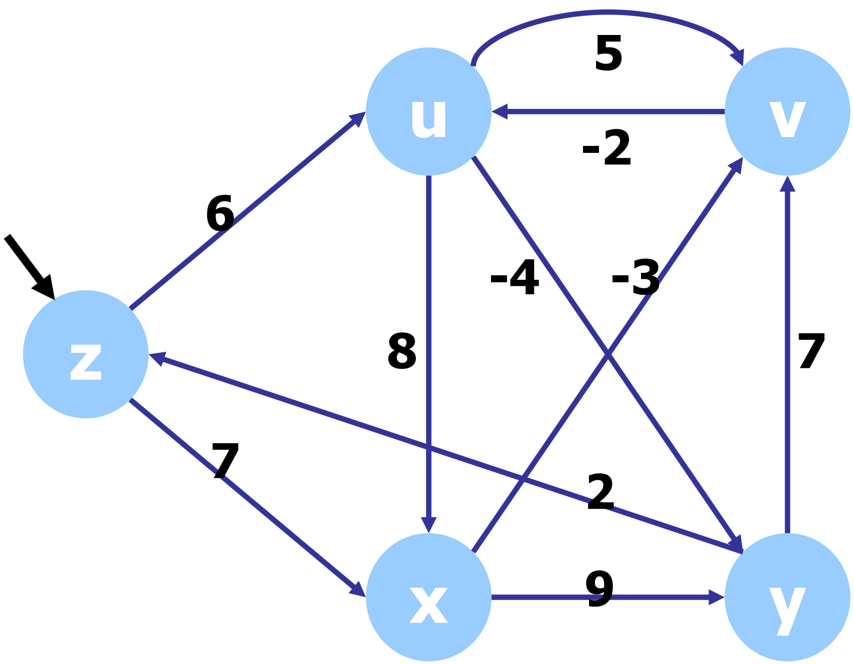

6
28
1
3
2
6
0
3
0
8
5
7
2
9
2
4
14
23


0
1
2
4
3

Algoritmo di Bellman-Ford

- Ipotesi: possono ∃ archi a peso < 0
- Rileva cicli < 0
- Strategia: greedy
- Inizializzazione di `st`
- |V|-1 passi di rilassamento sugli archi
- |V|esimo rilassamento:
  - diminuisce almeno una stima: ∃ ciclo <0
  - altrimenti soluzione ottima.

Esempio



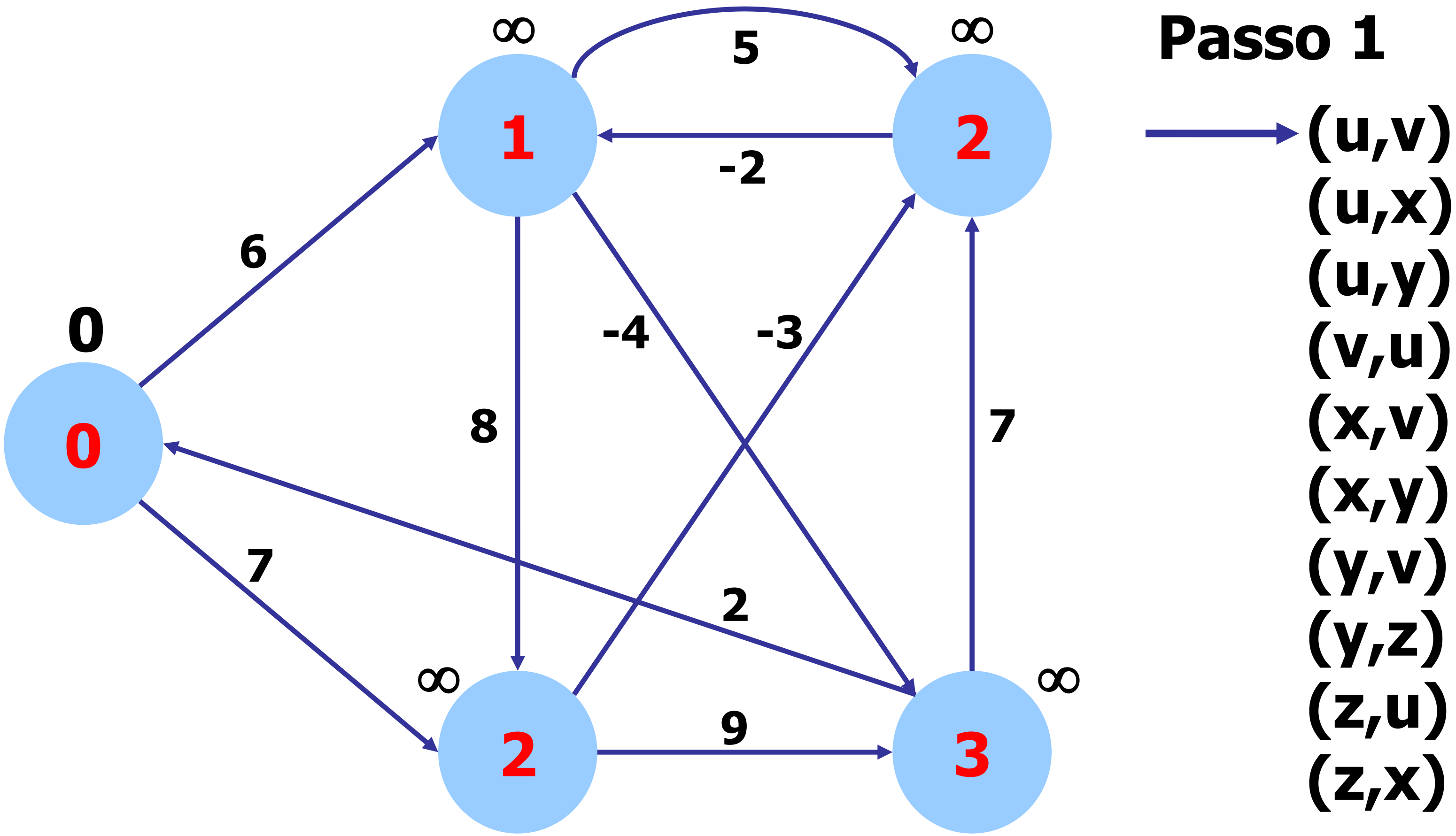
ST

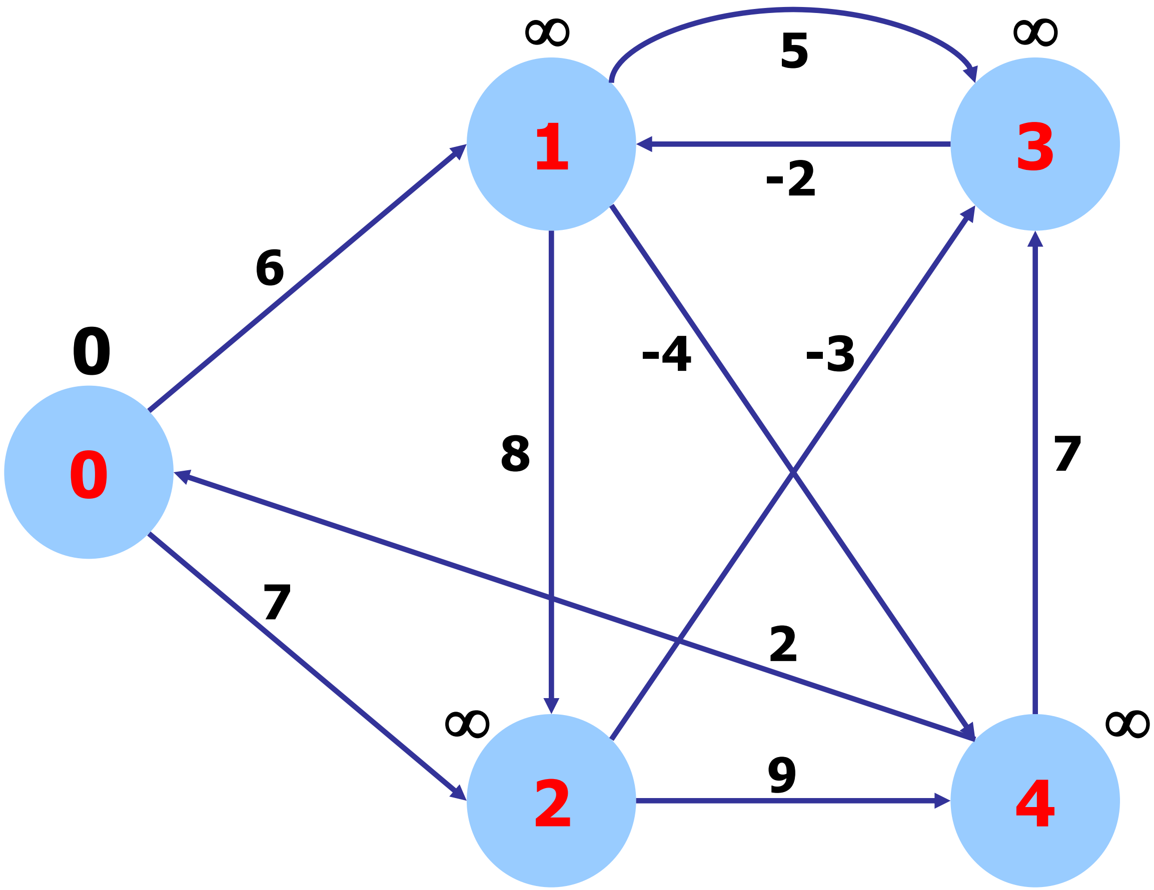
0	z
1	u
2	x
3	v
4	y

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

Negli archi i nodi compaiono con il loro nome originale per leggibilità

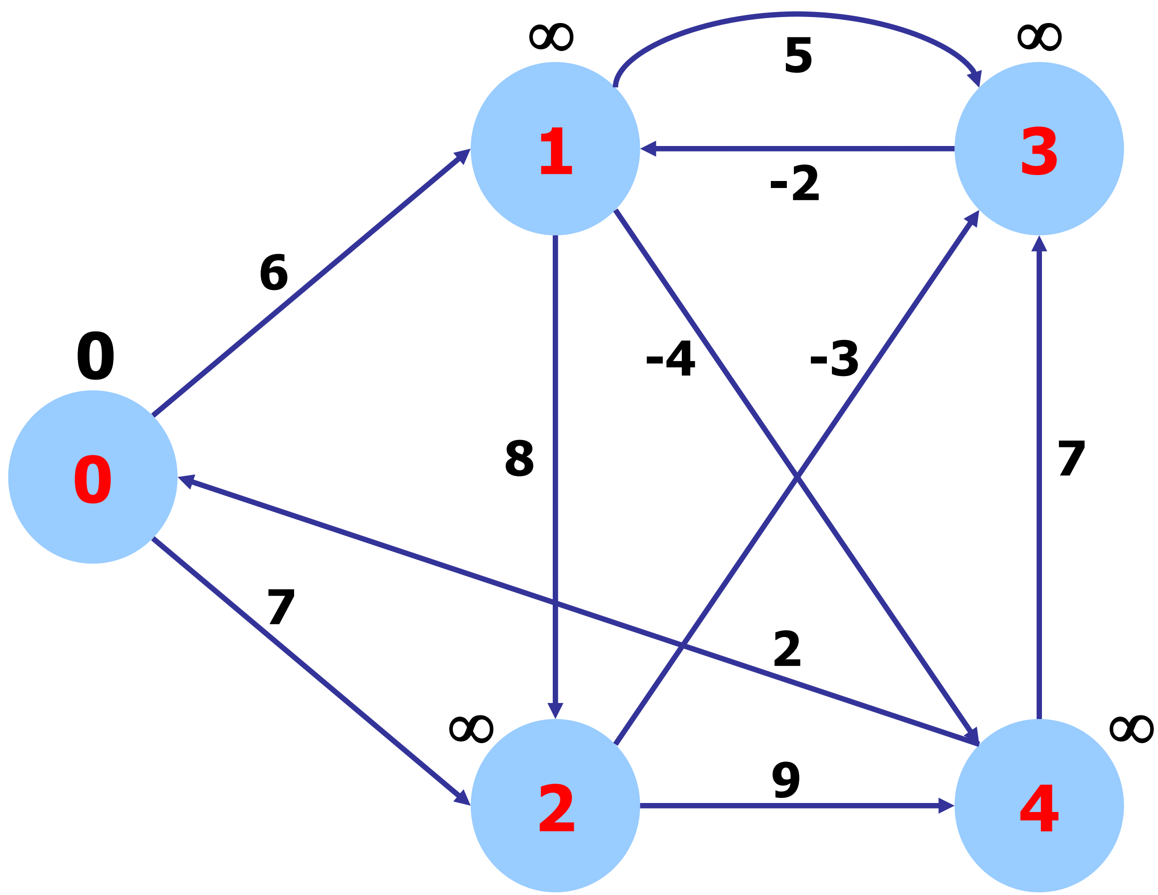
Passo 1





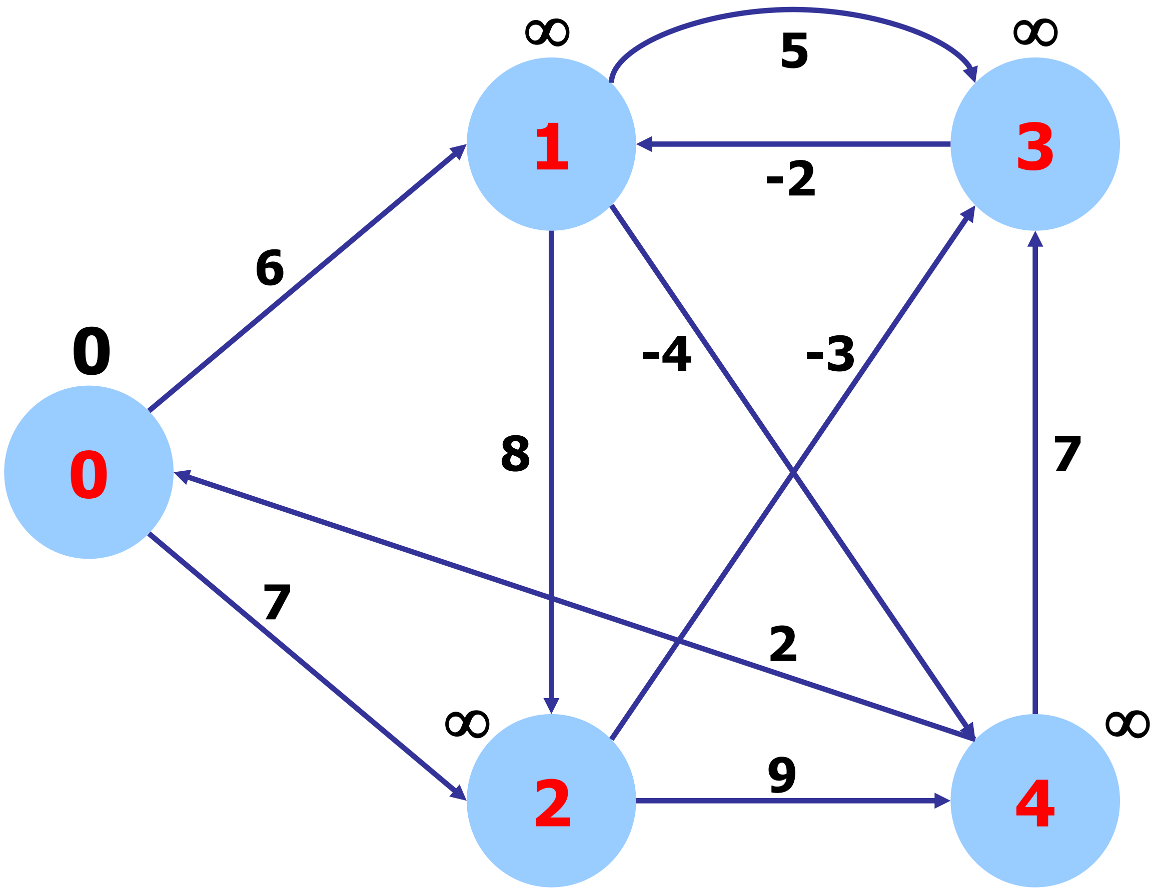
Passo 1

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



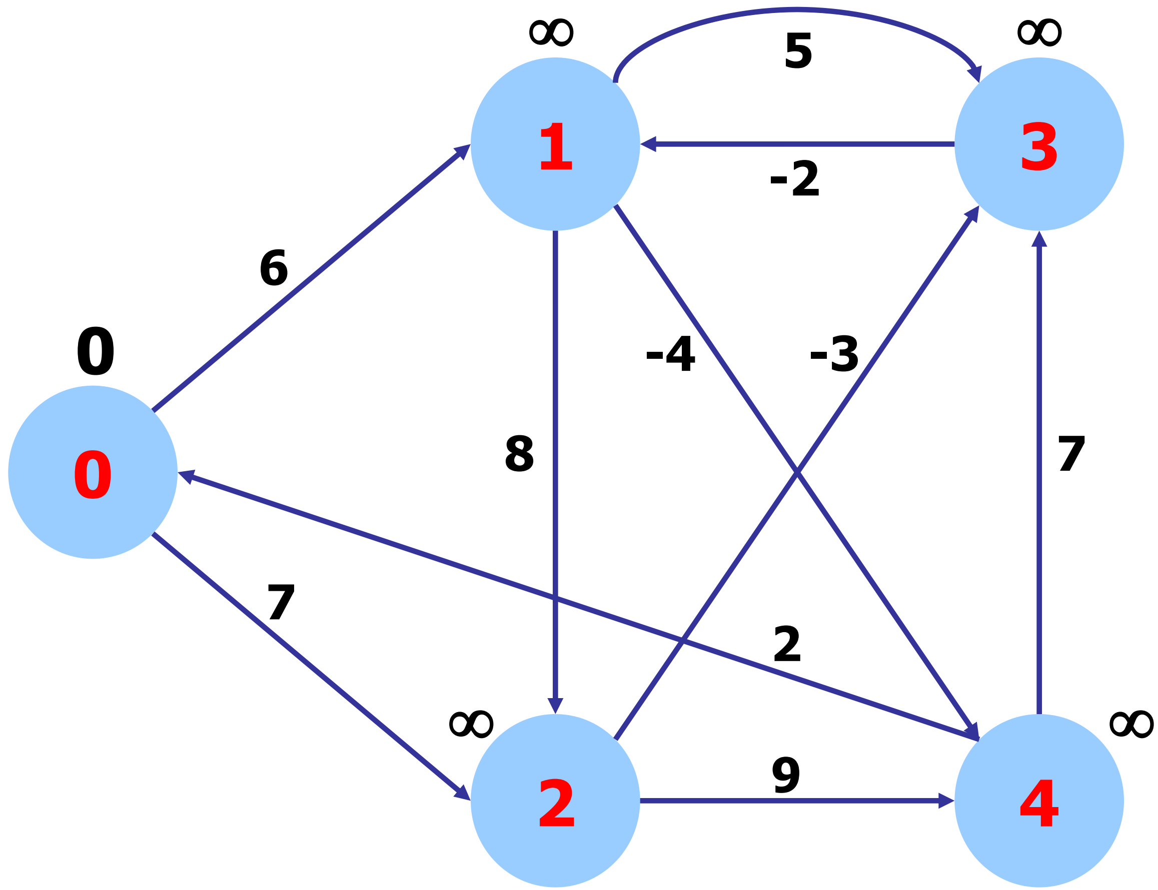
Passo 1

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



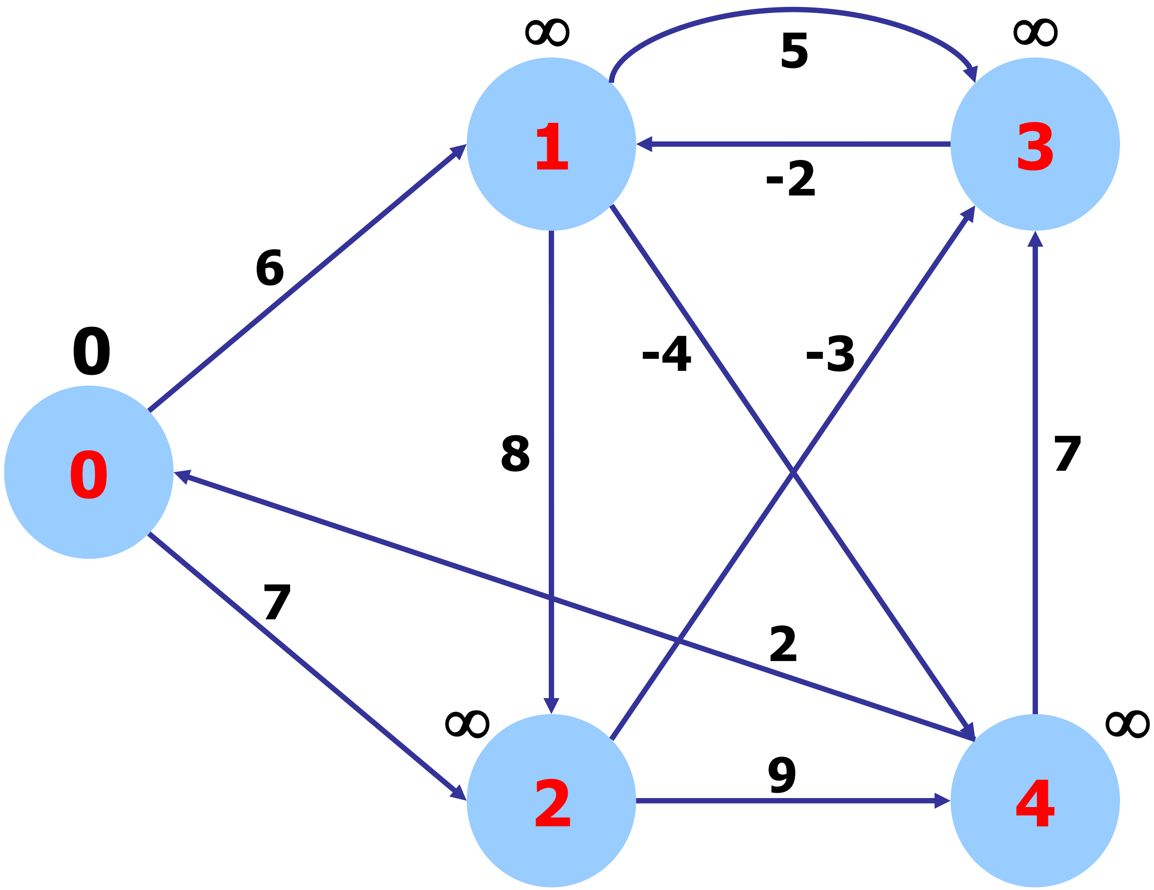
Passo 1

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



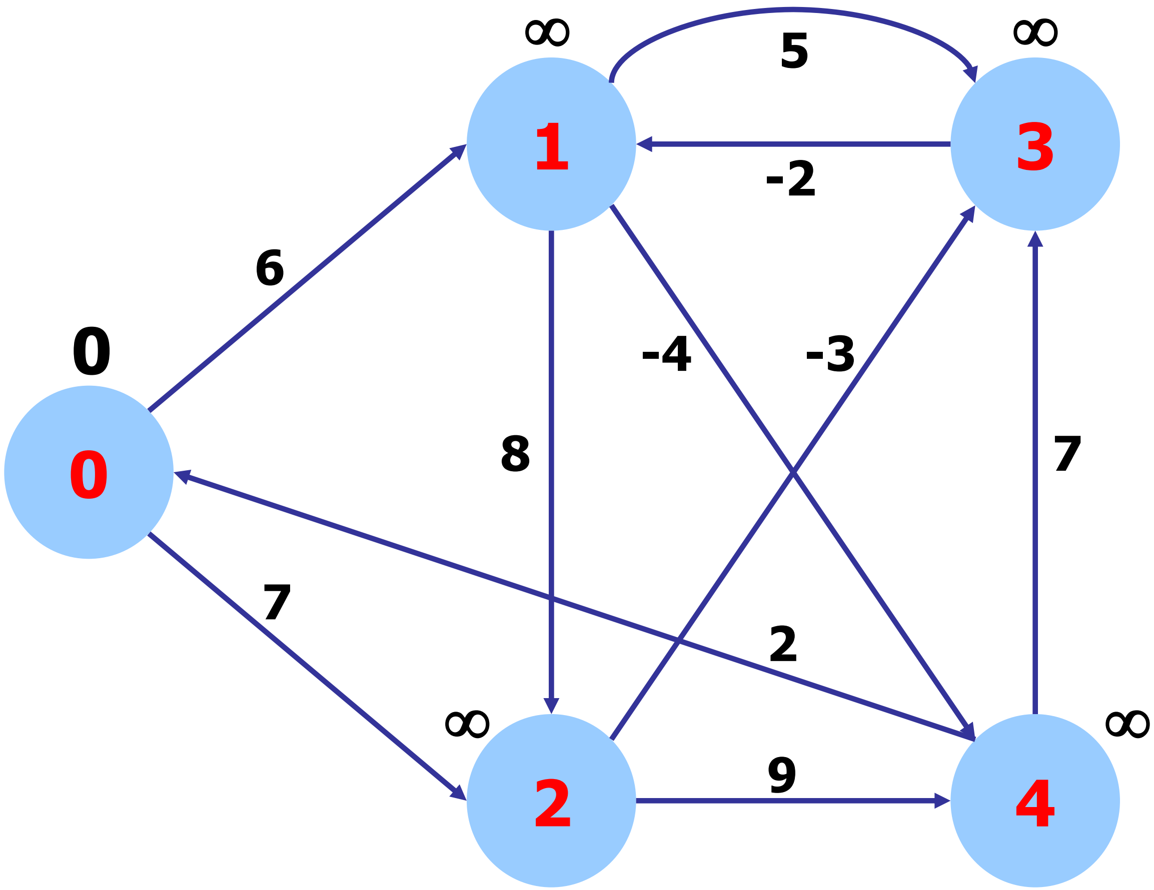
Passo 1

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



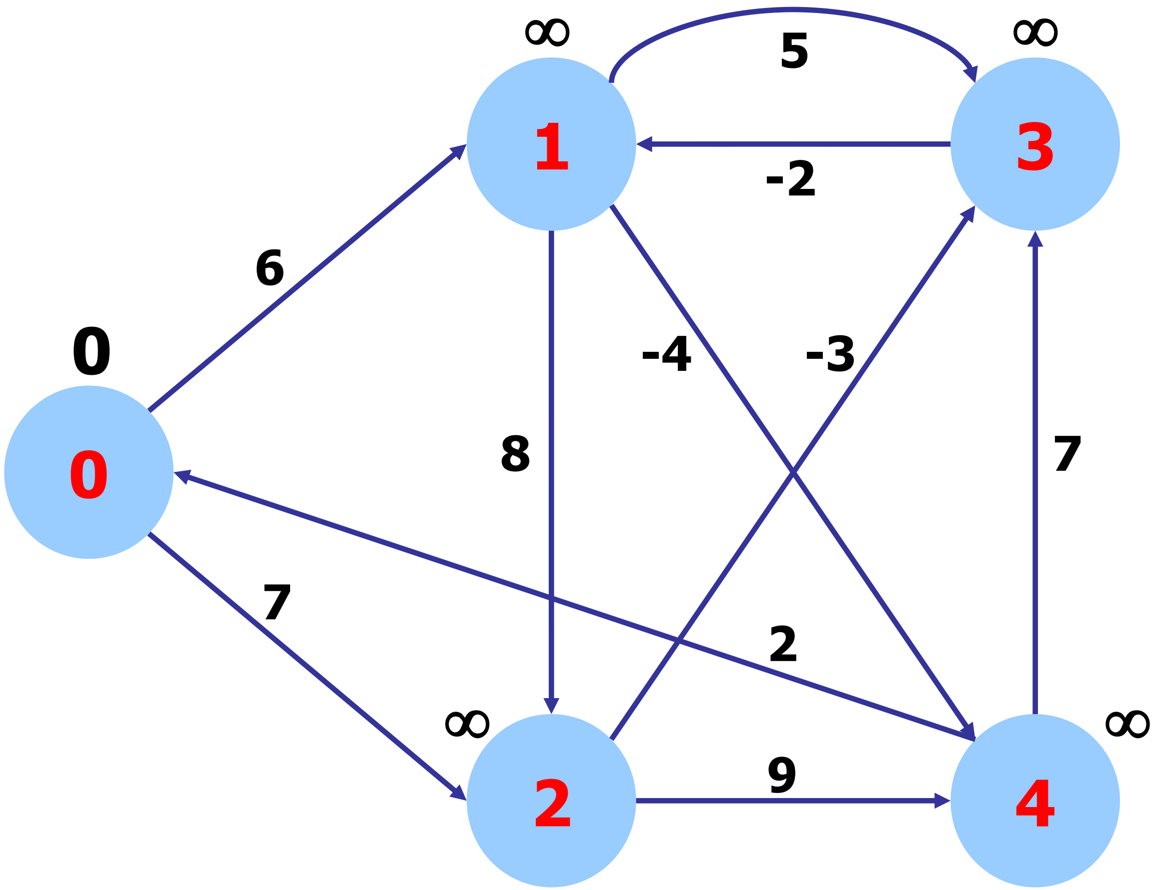
Passo 1

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



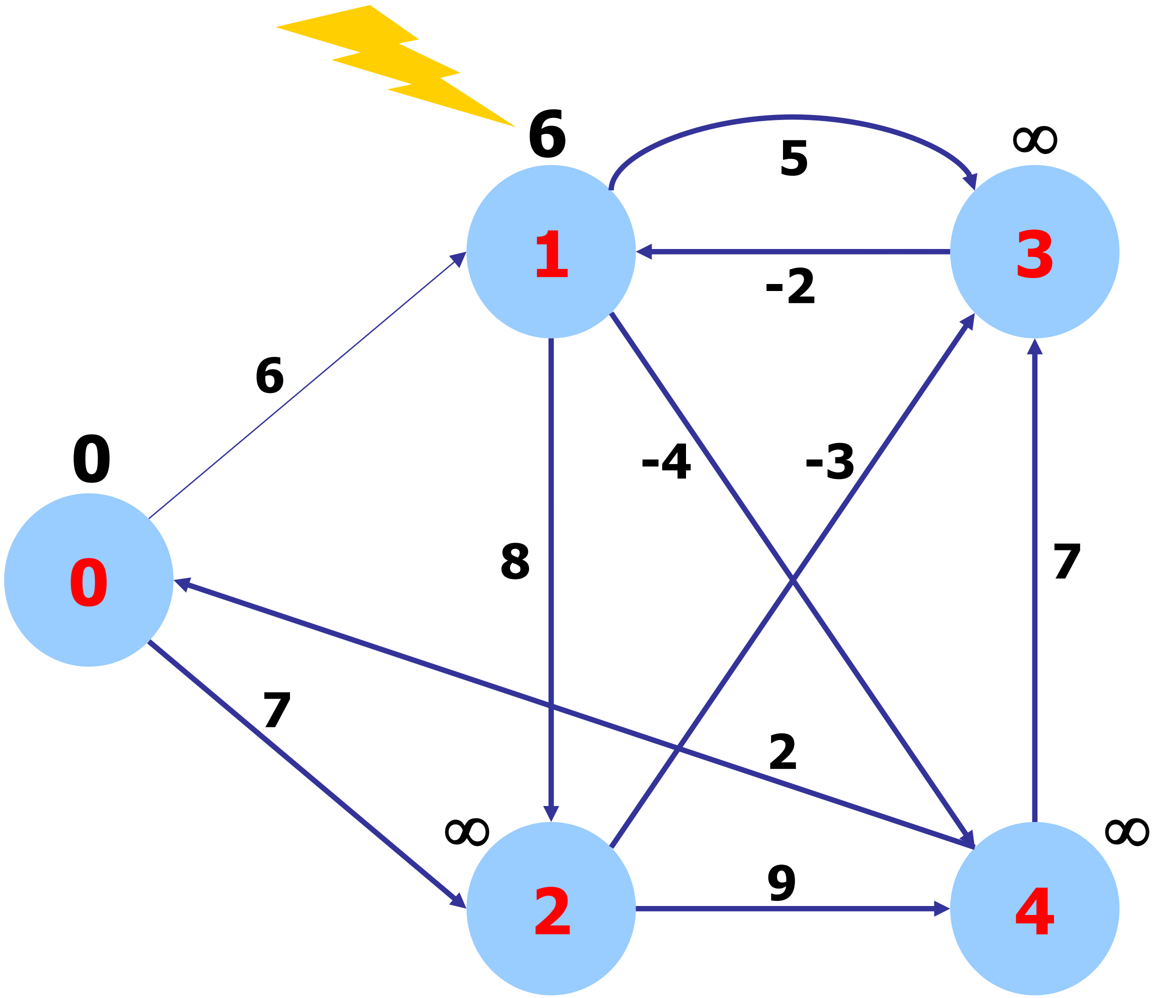
Passo 1

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



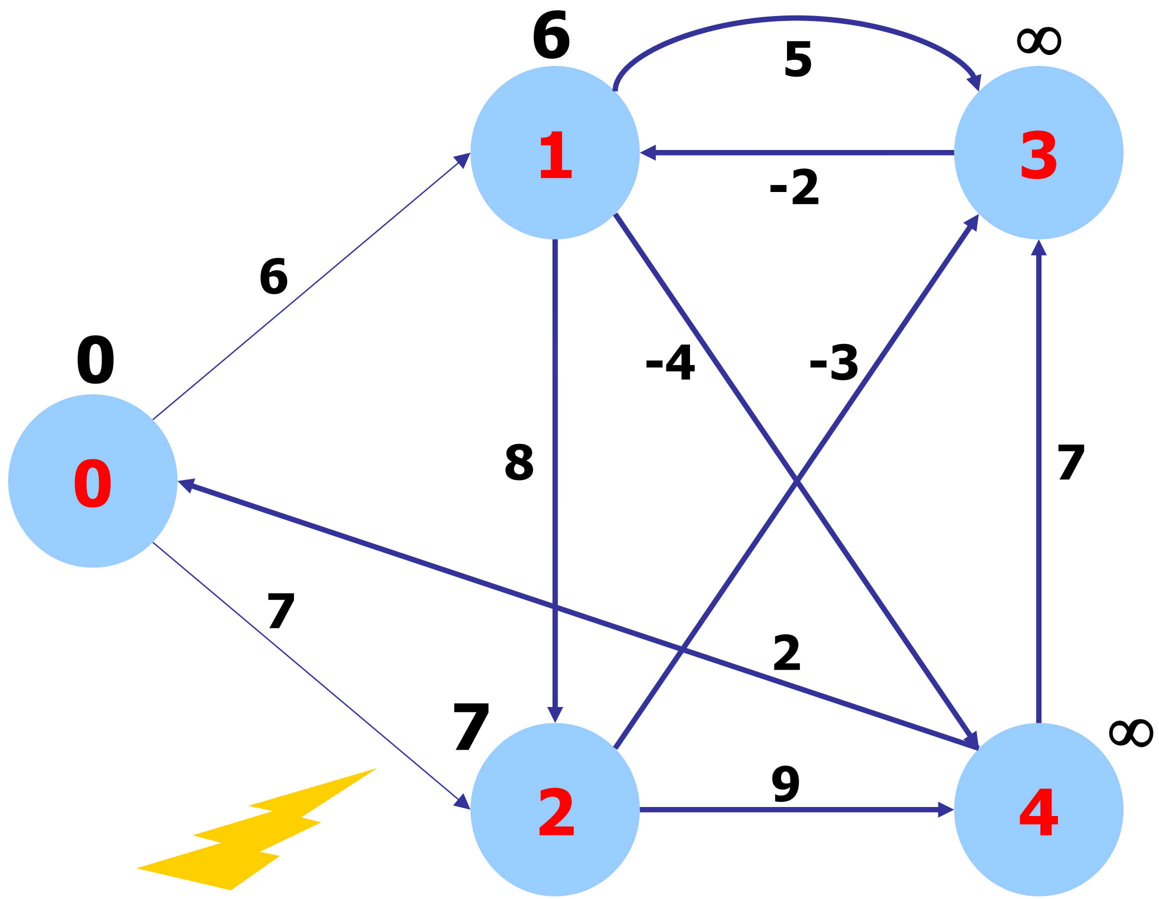
Passo 1

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



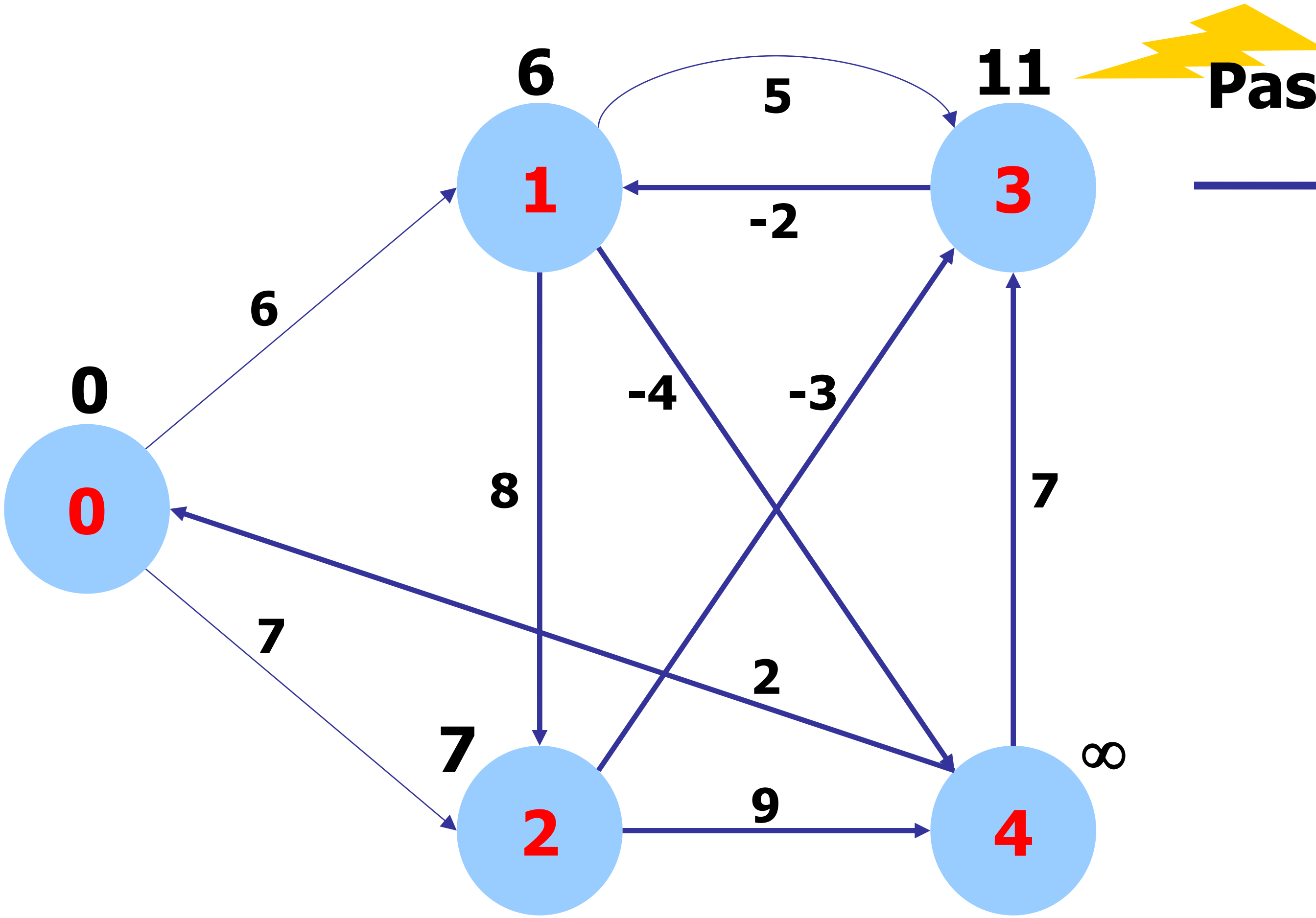
Passo 1

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

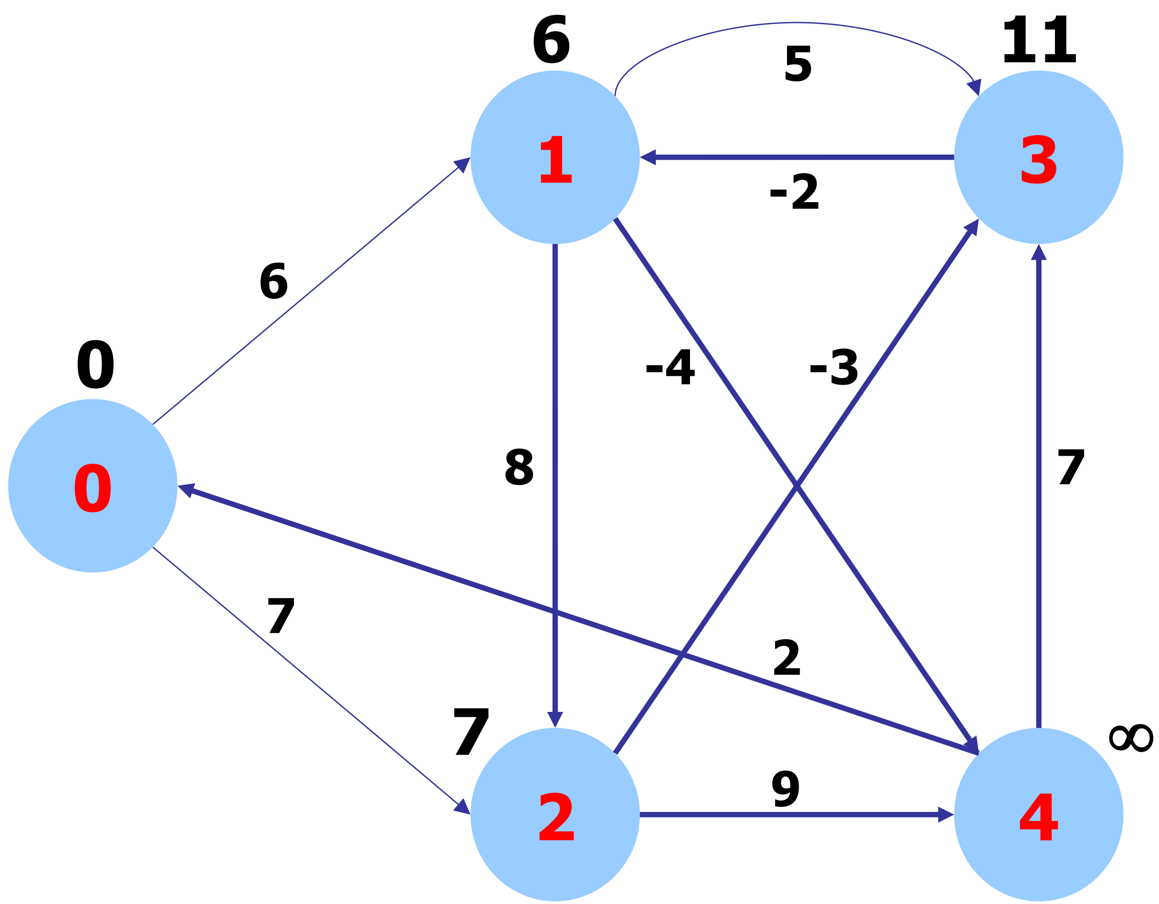


Passo 1

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

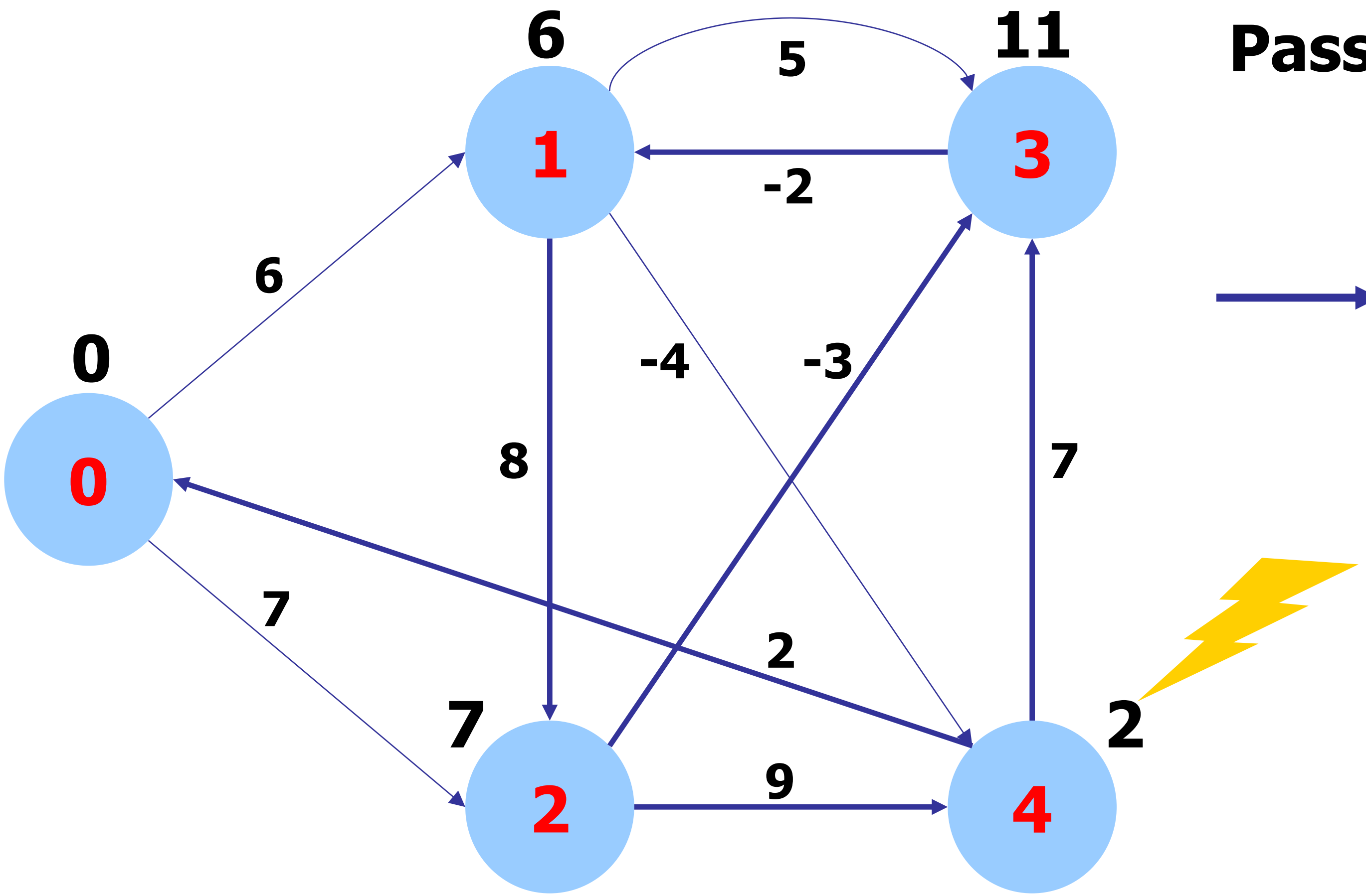


(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



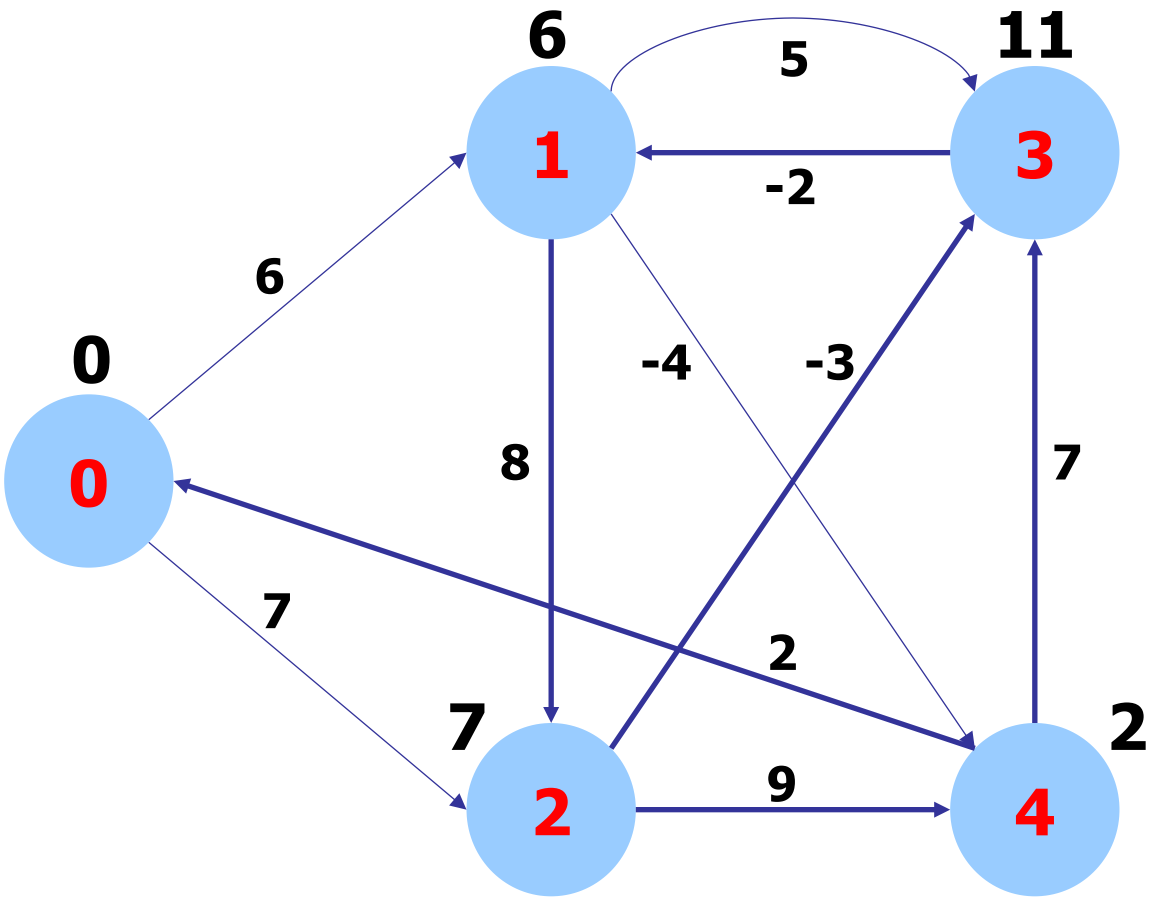
Passo 2

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



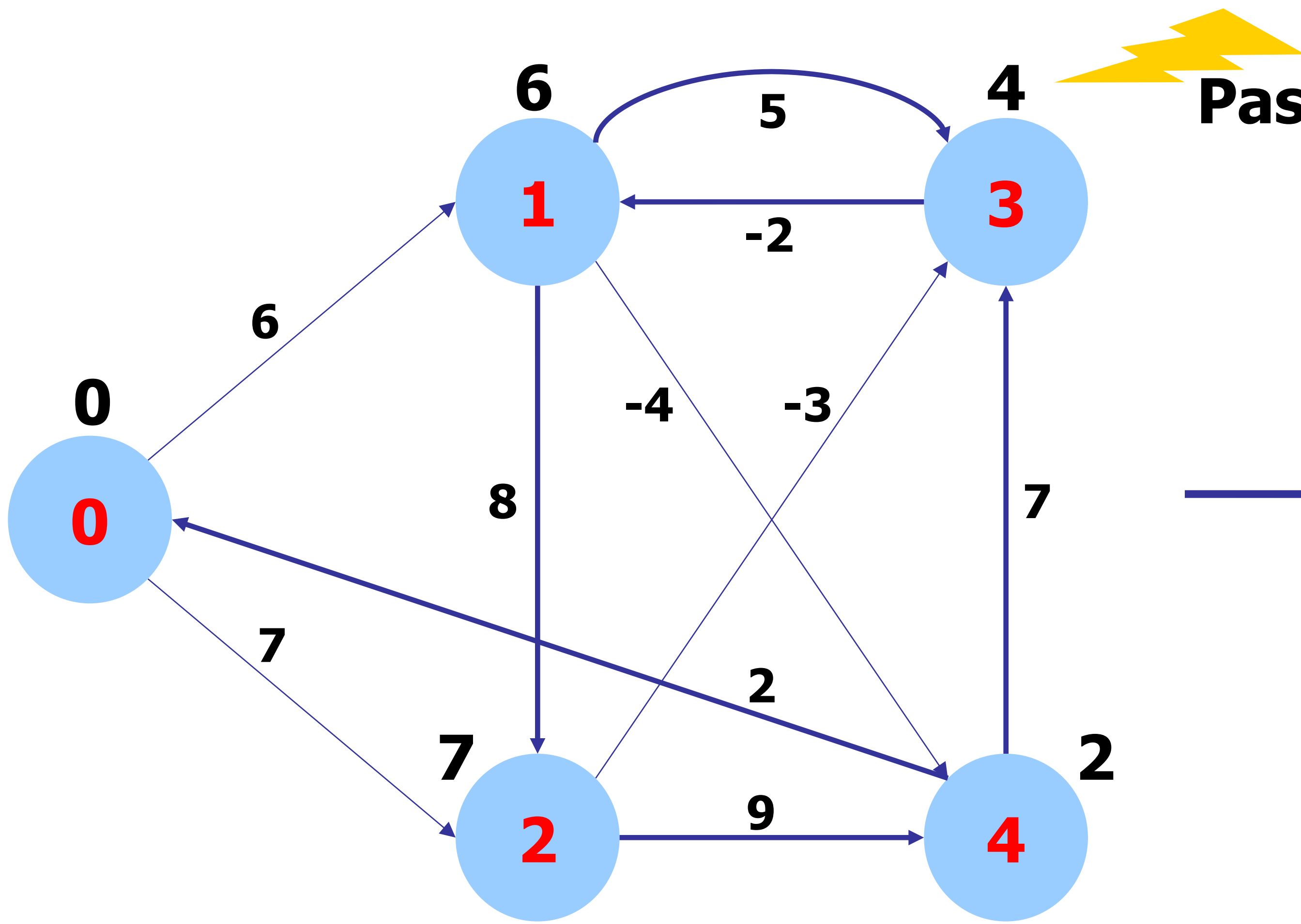
Passo 2

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

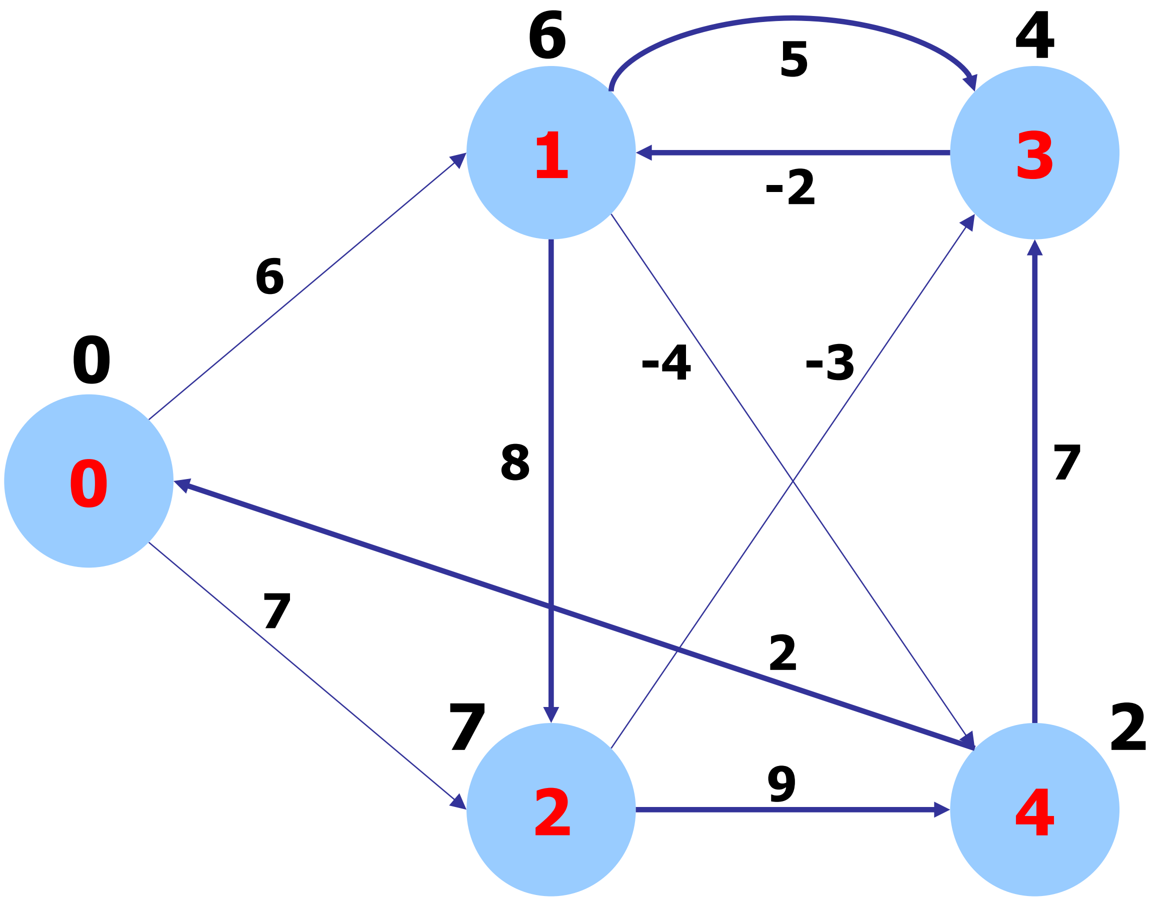


Passo 2

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

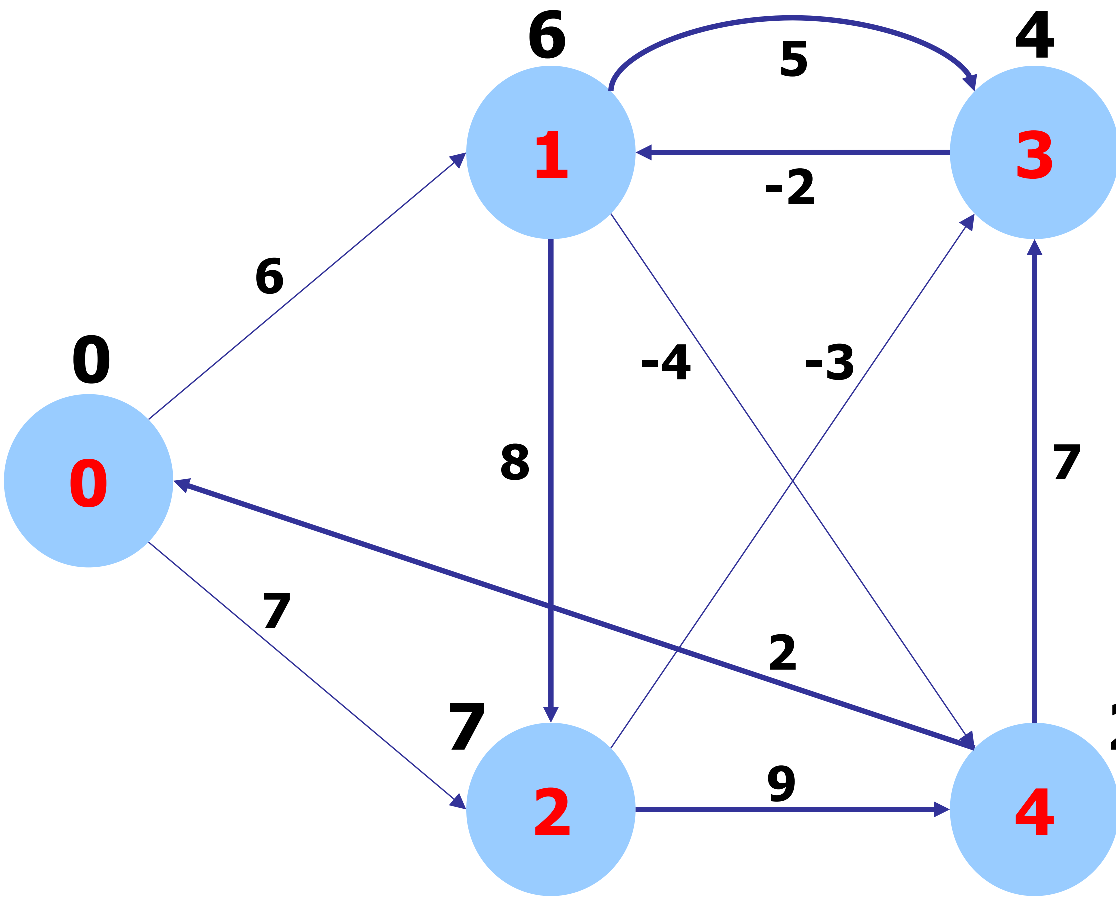


(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



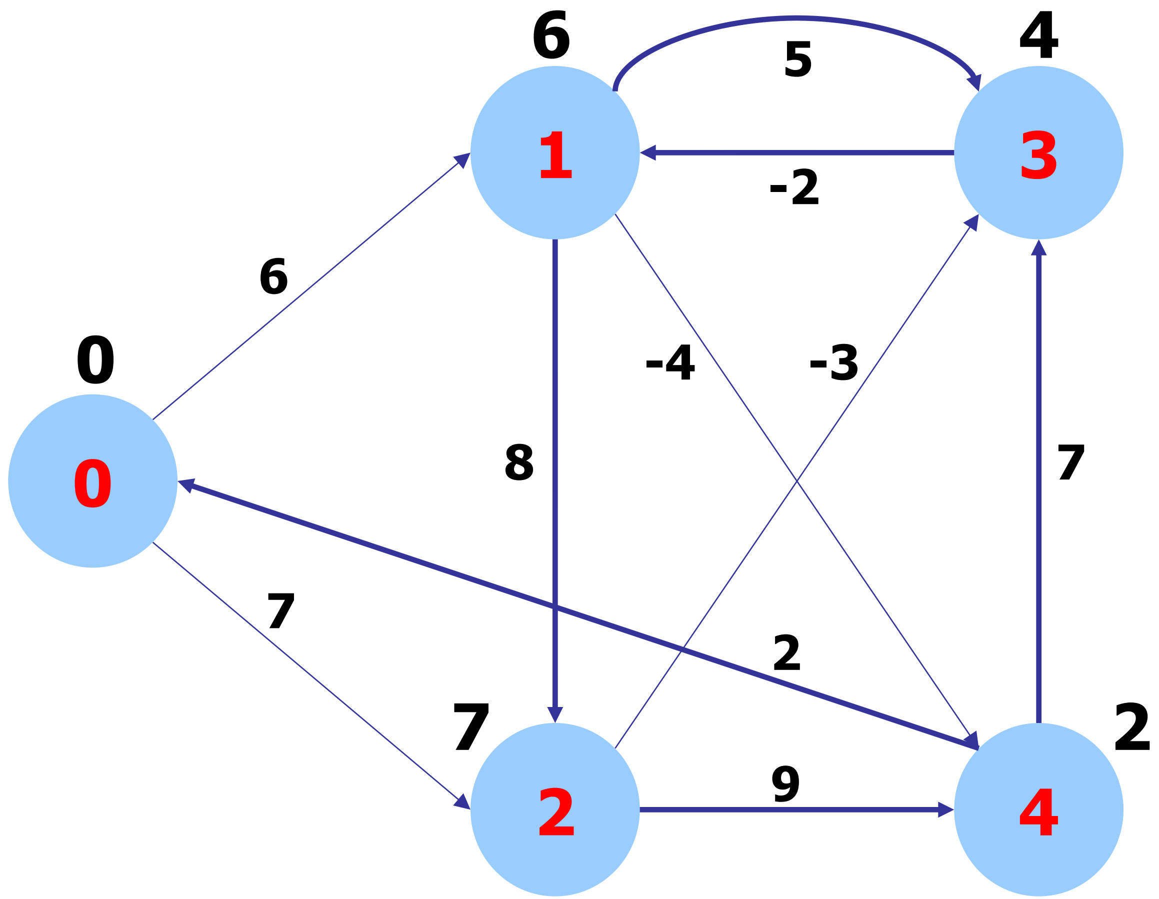
Passo 2

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



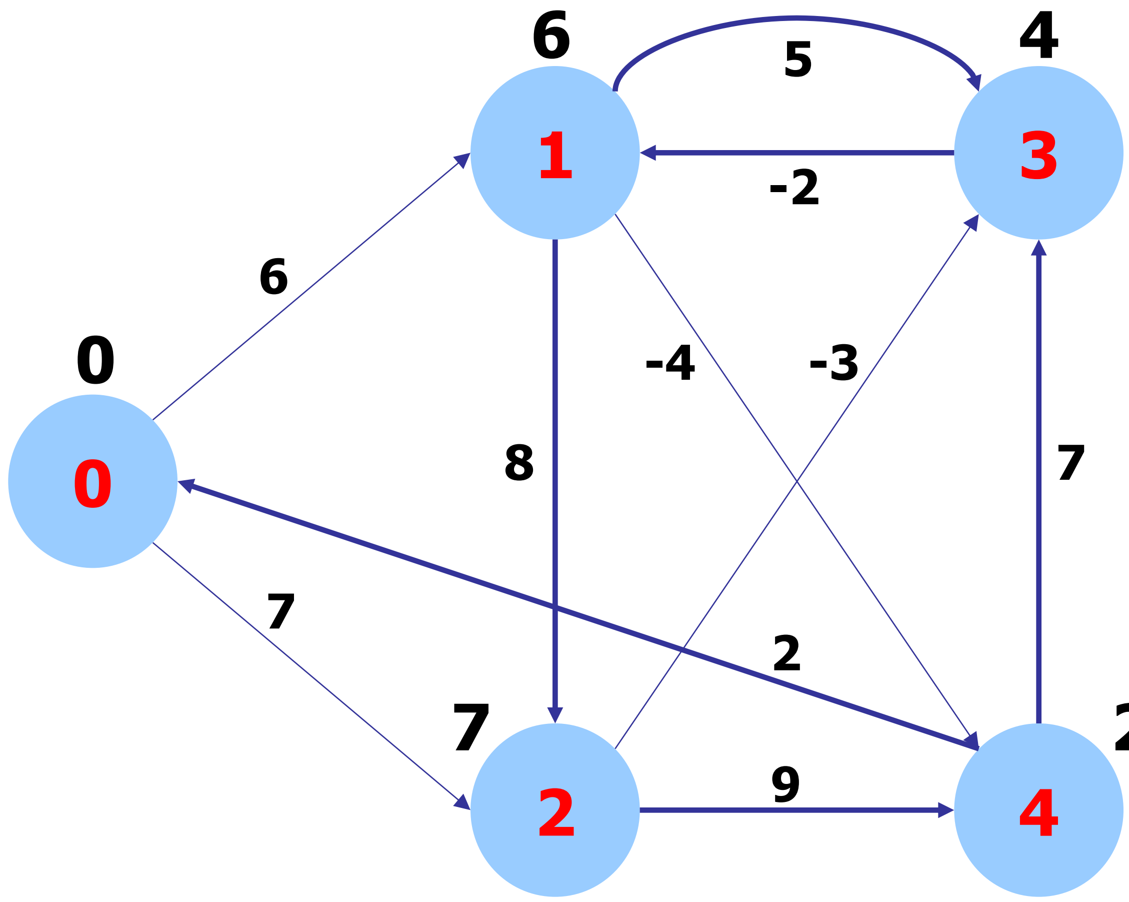
Passo 2

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



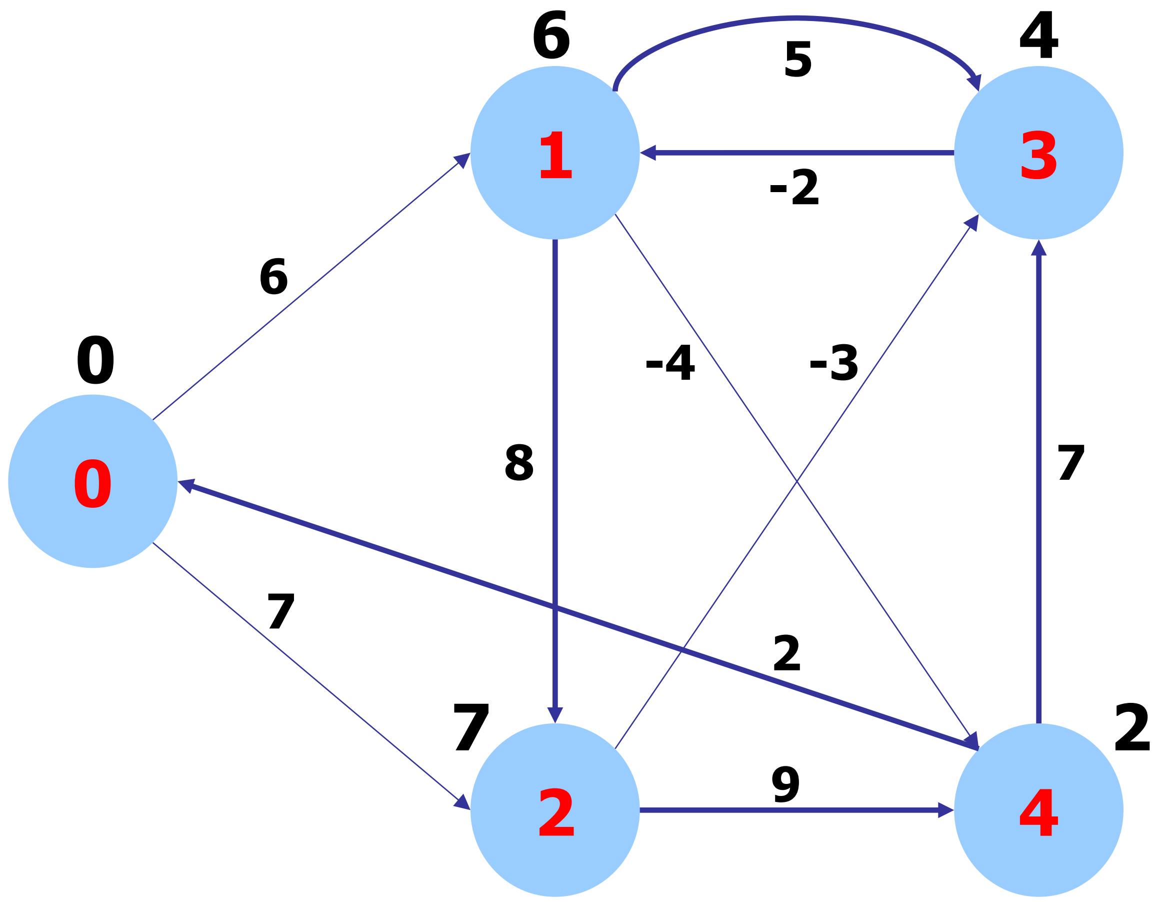
Passo 2

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



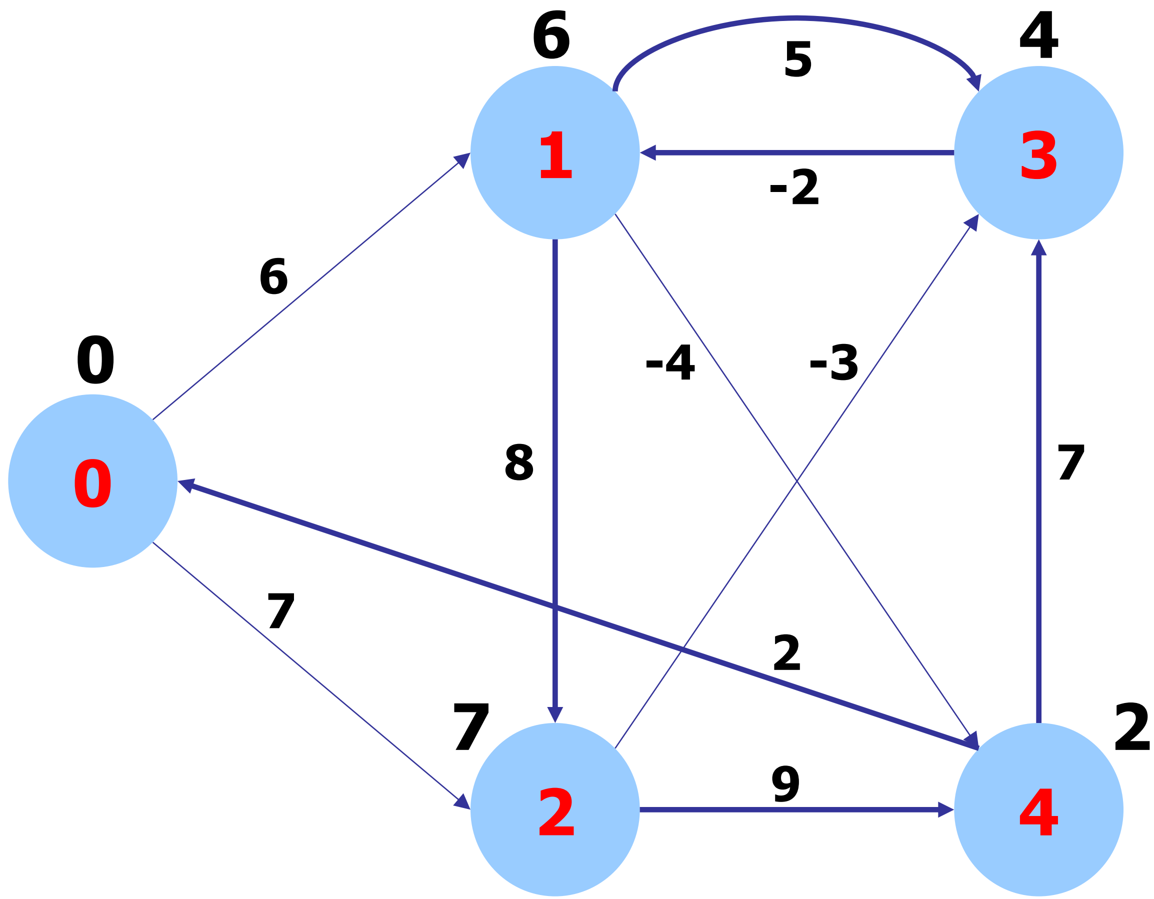
Passo 2

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



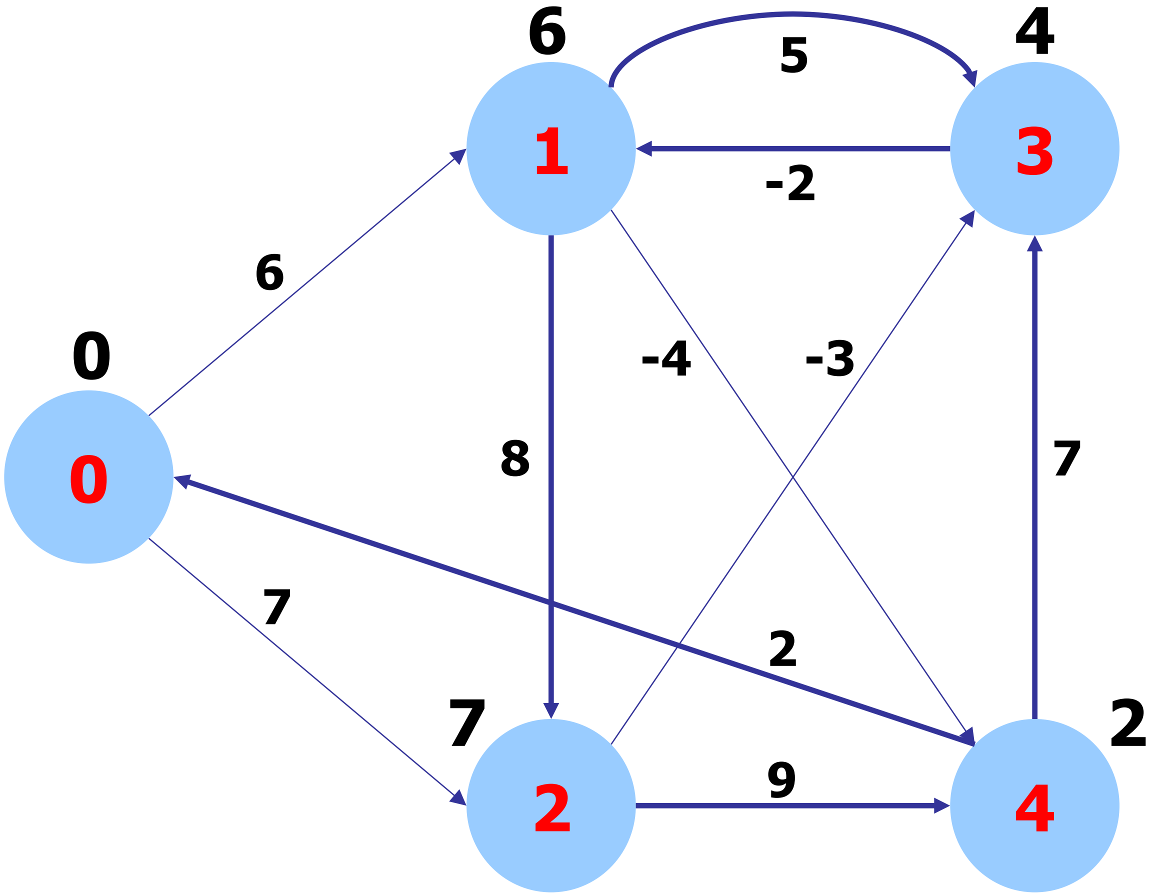
Passo 2

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



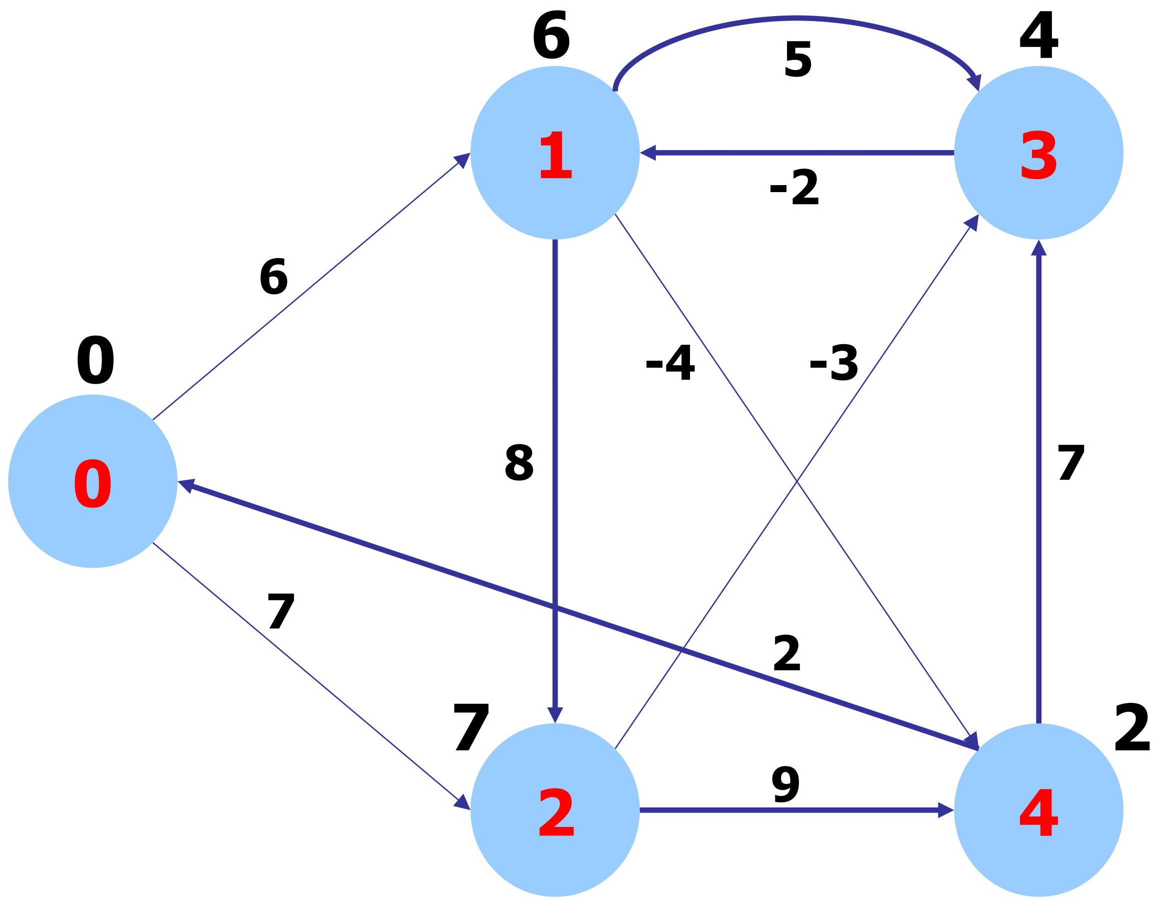
Passo 3

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



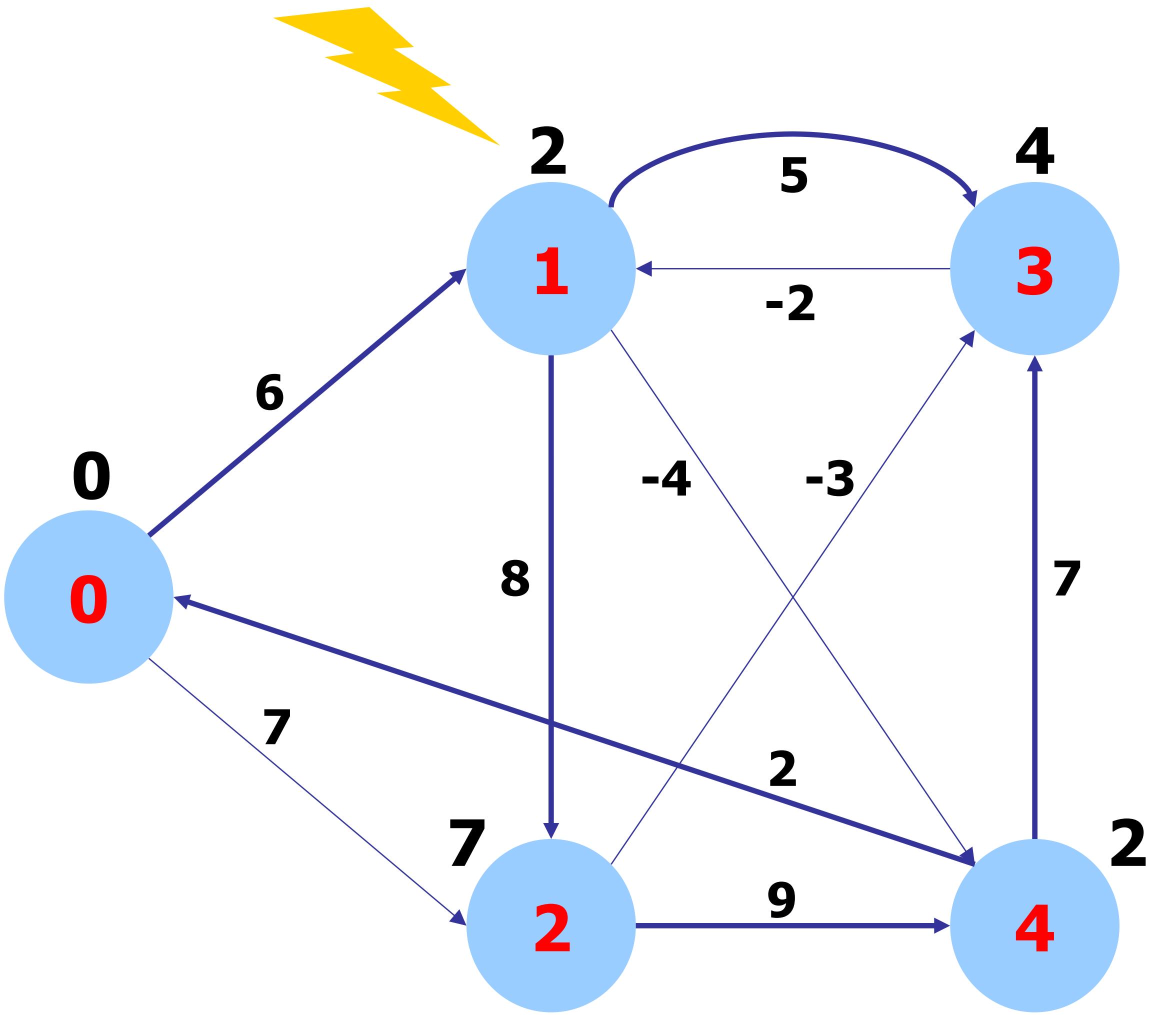
Passo 3

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



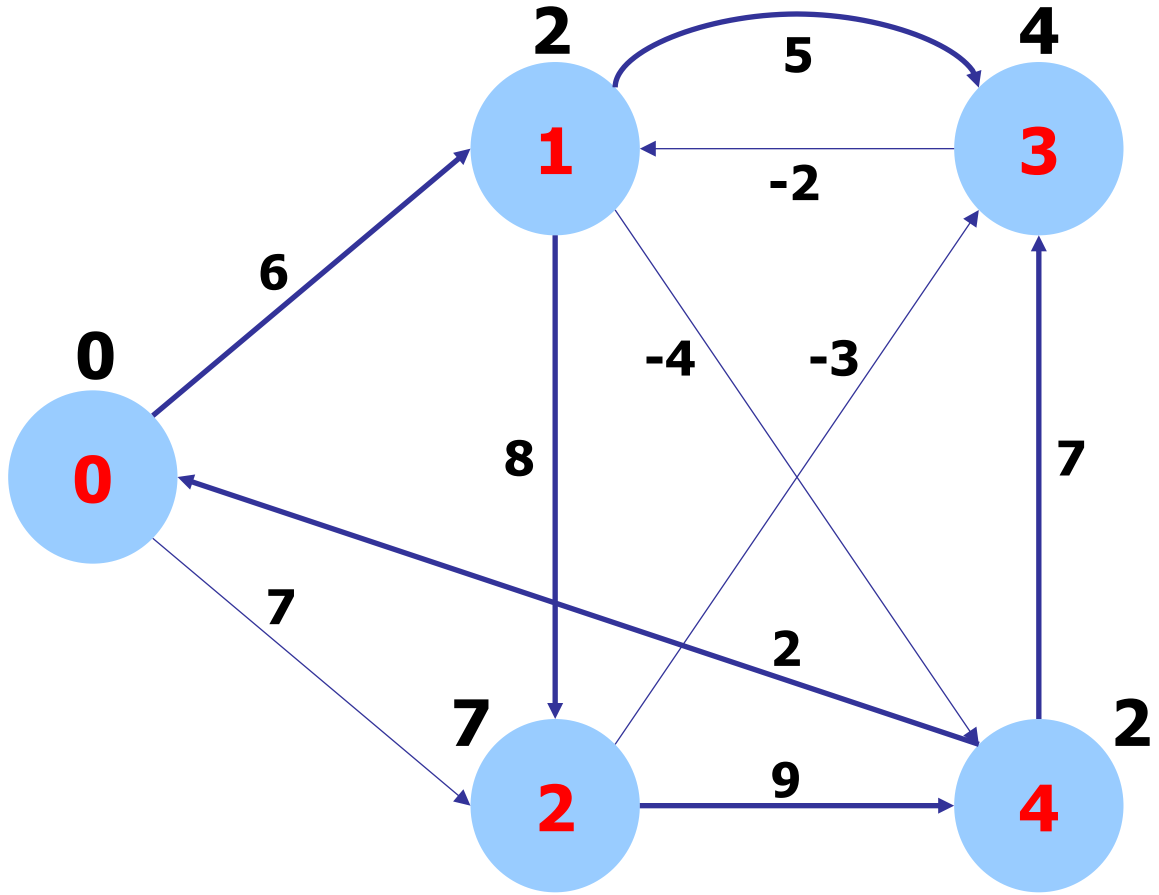
Passo 3

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



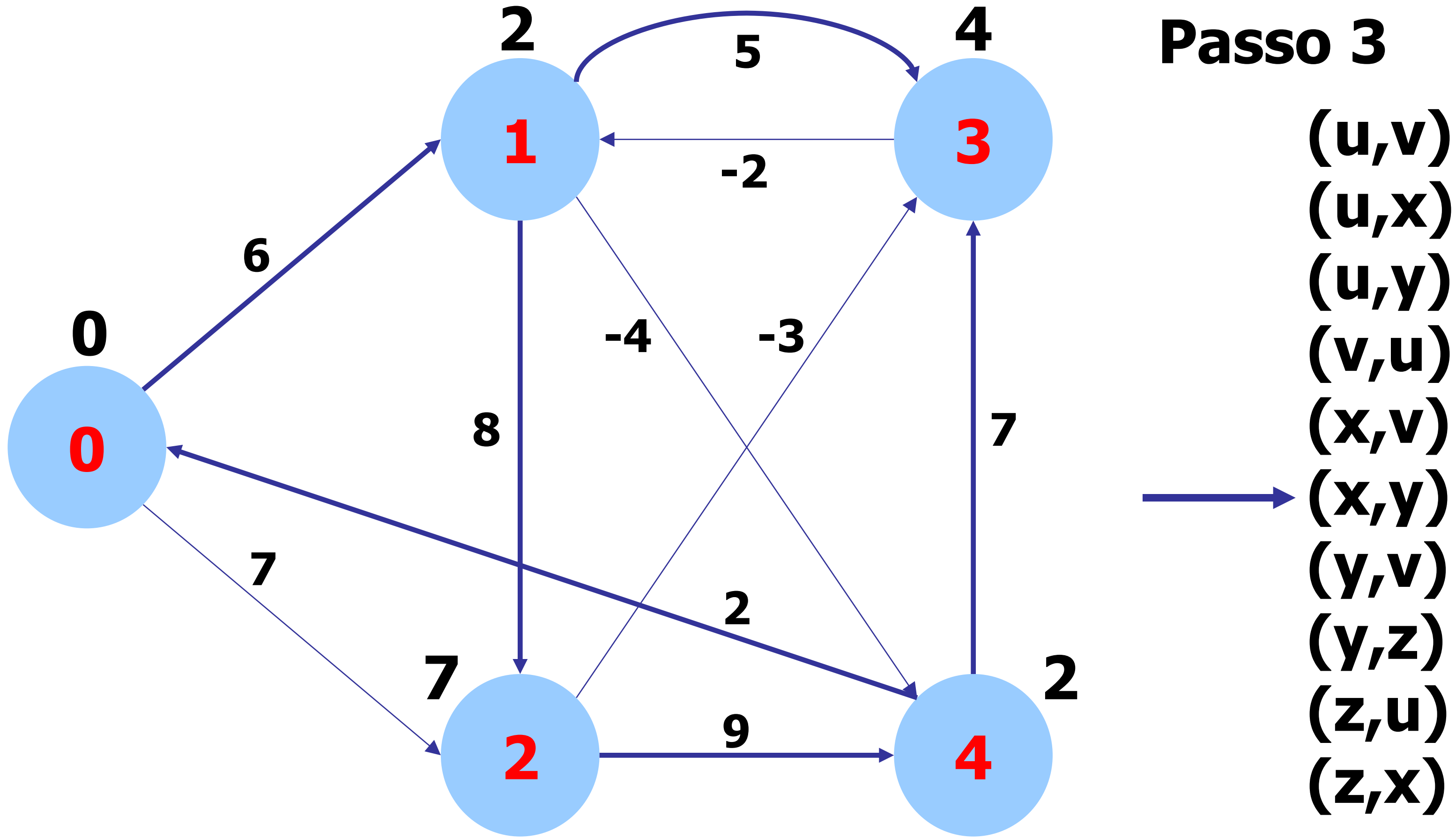
Passo 3

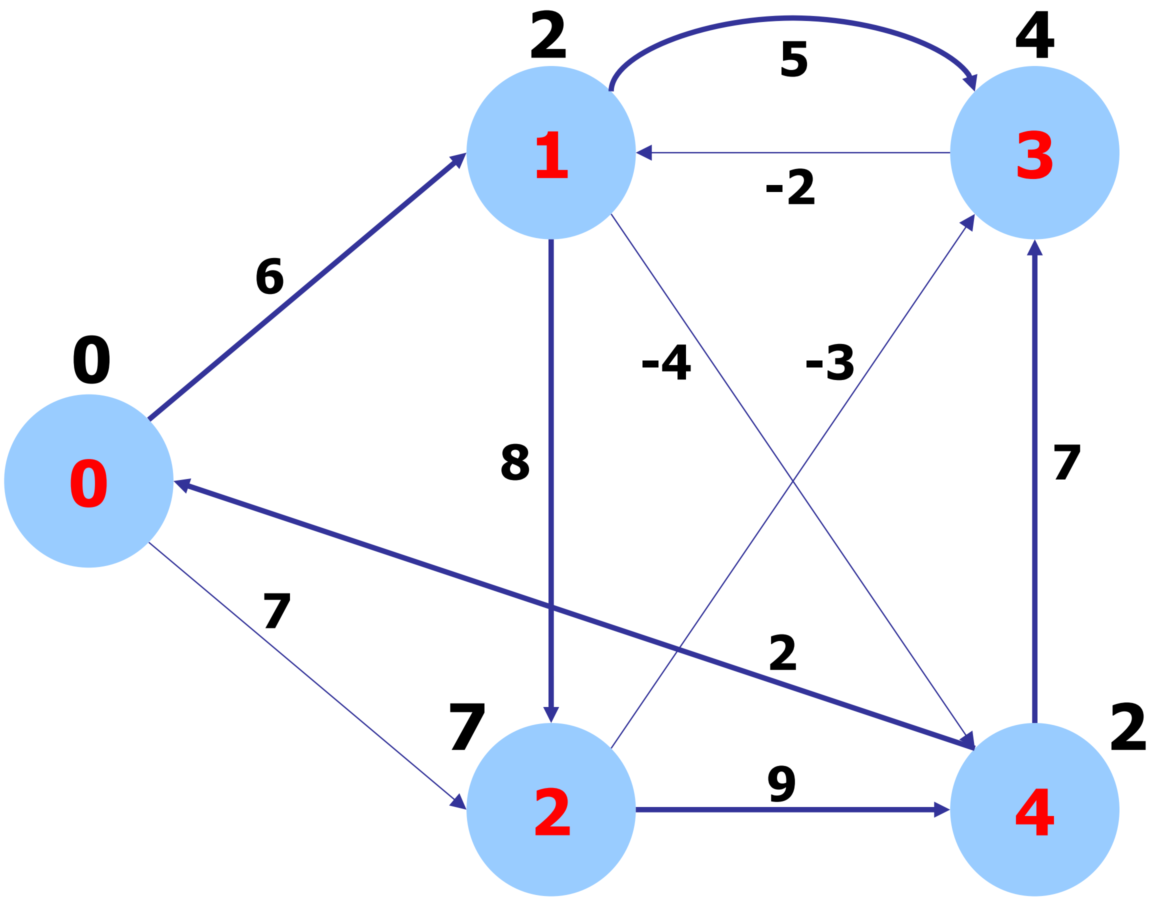
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 3

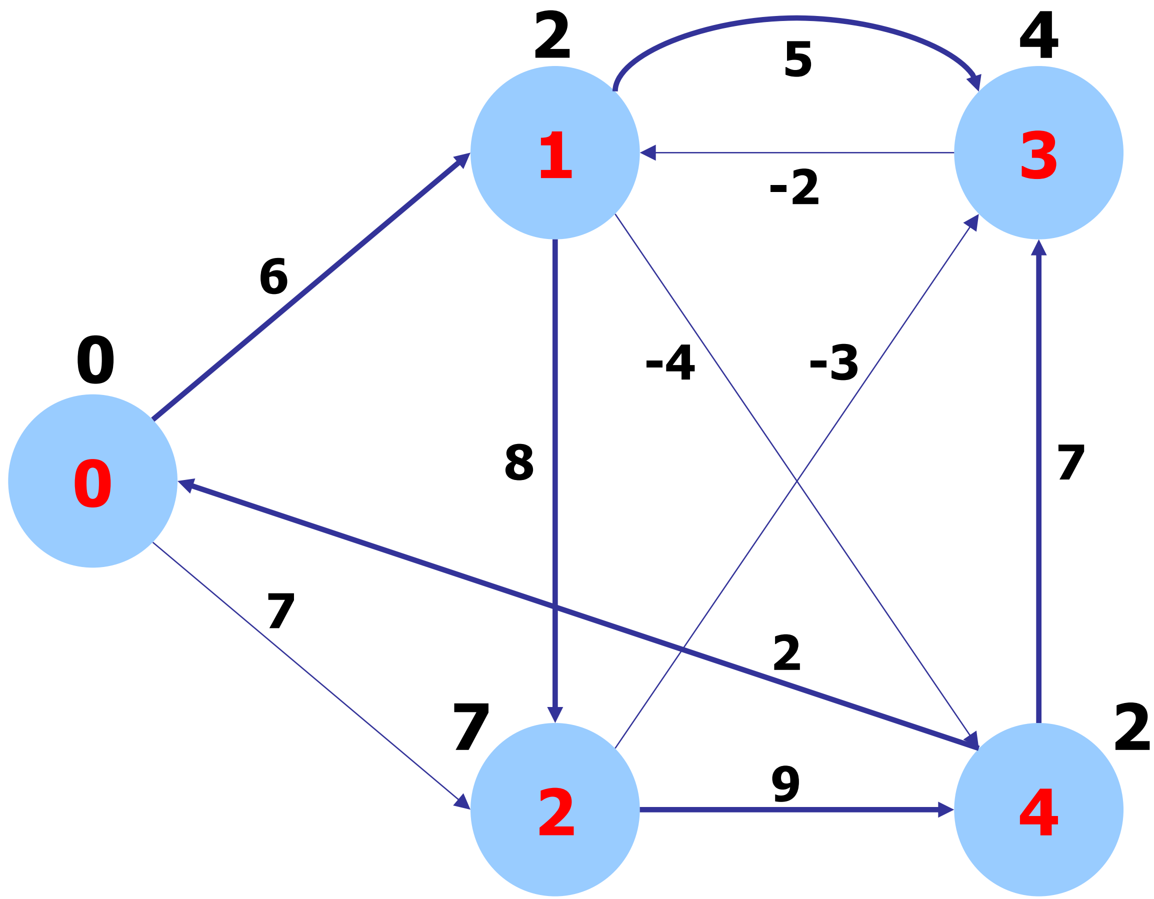
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)


2
4
5
1
3
-2
6
-4
-3
0
0
8
7
7
2
7
2
2
4
9
Passo 3
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



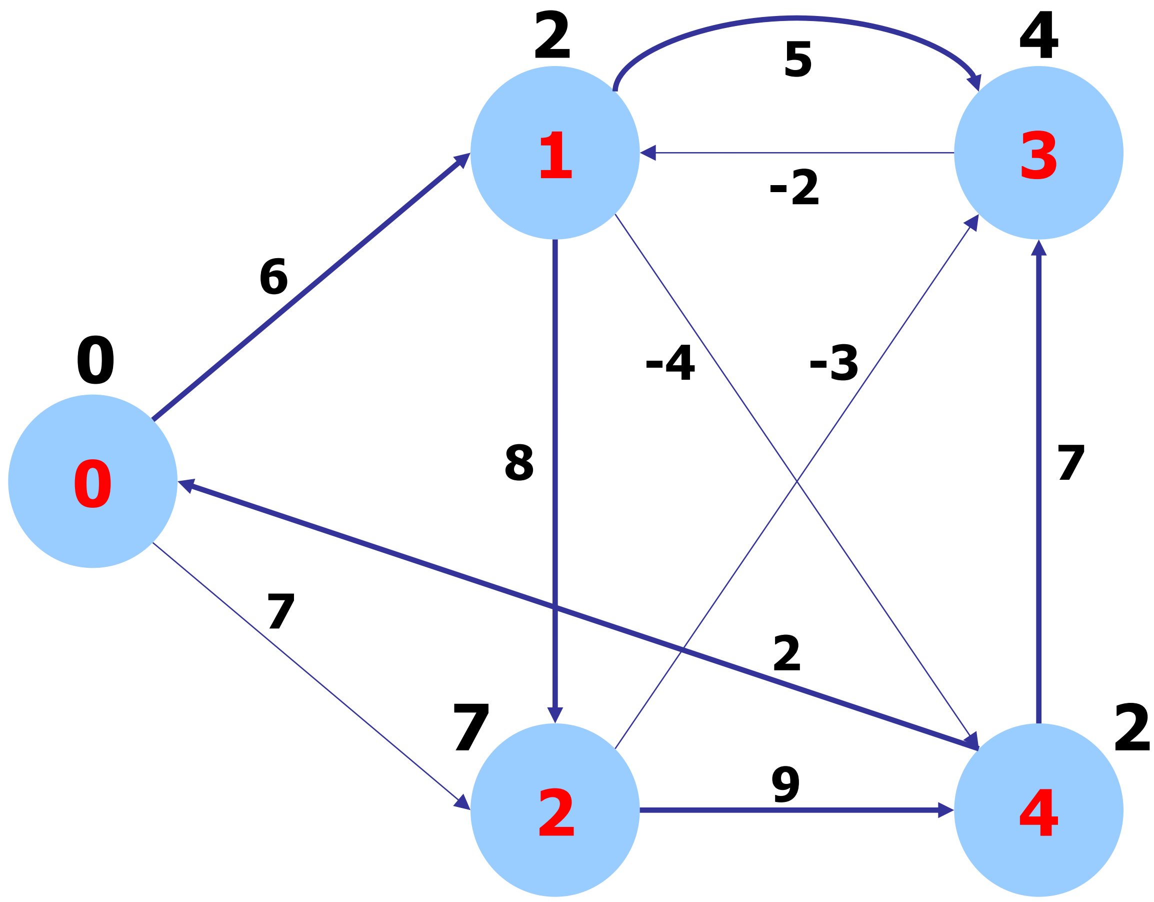
Passo 3

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



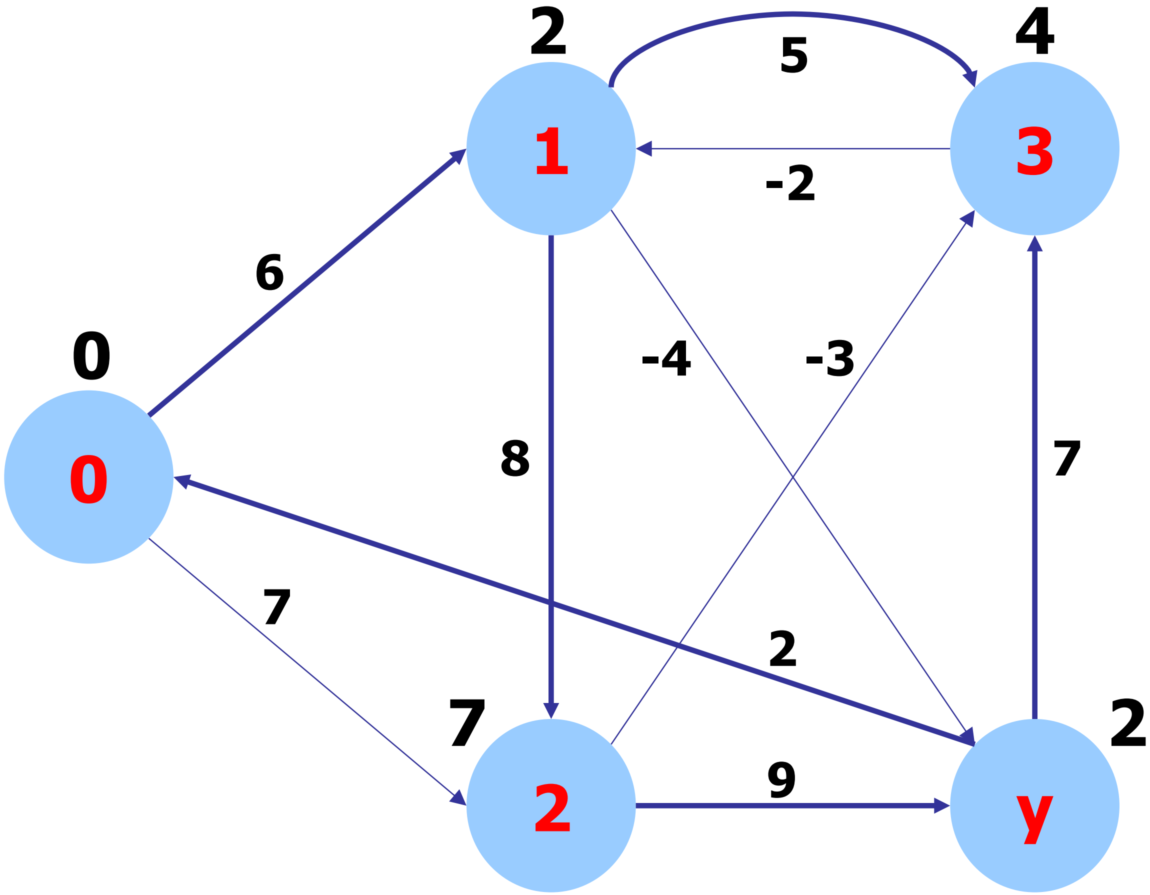
Passo 3

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



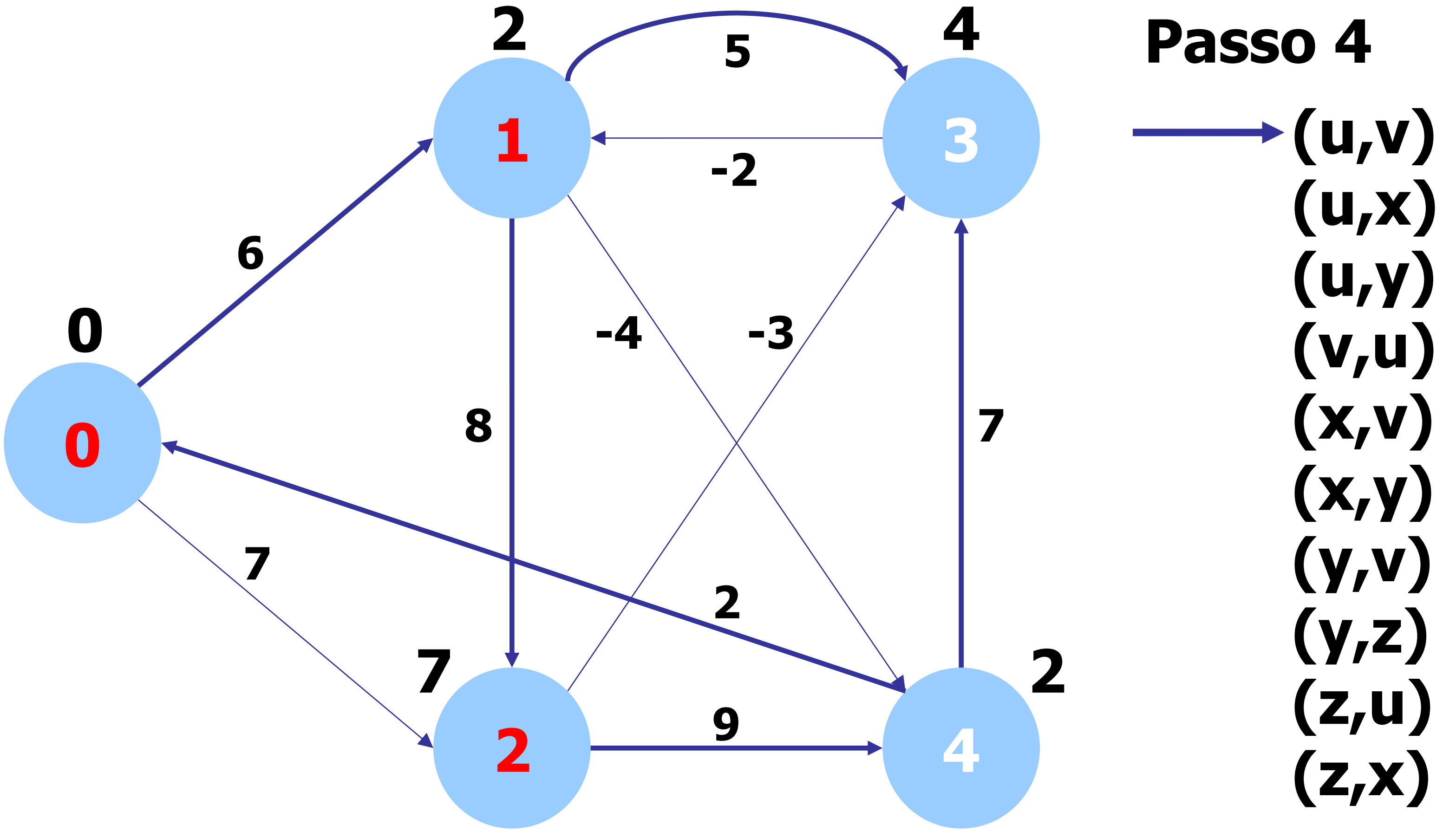
Passo 3

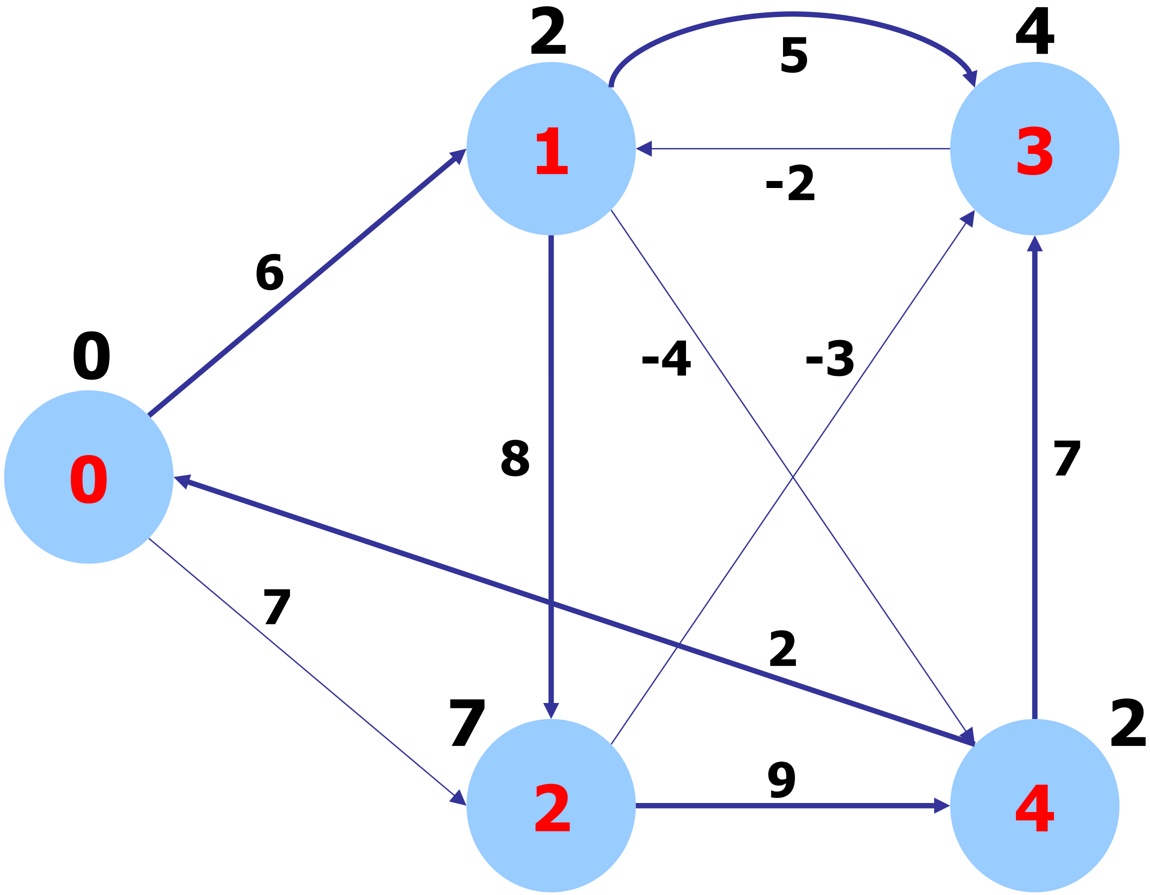
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



Passo 3

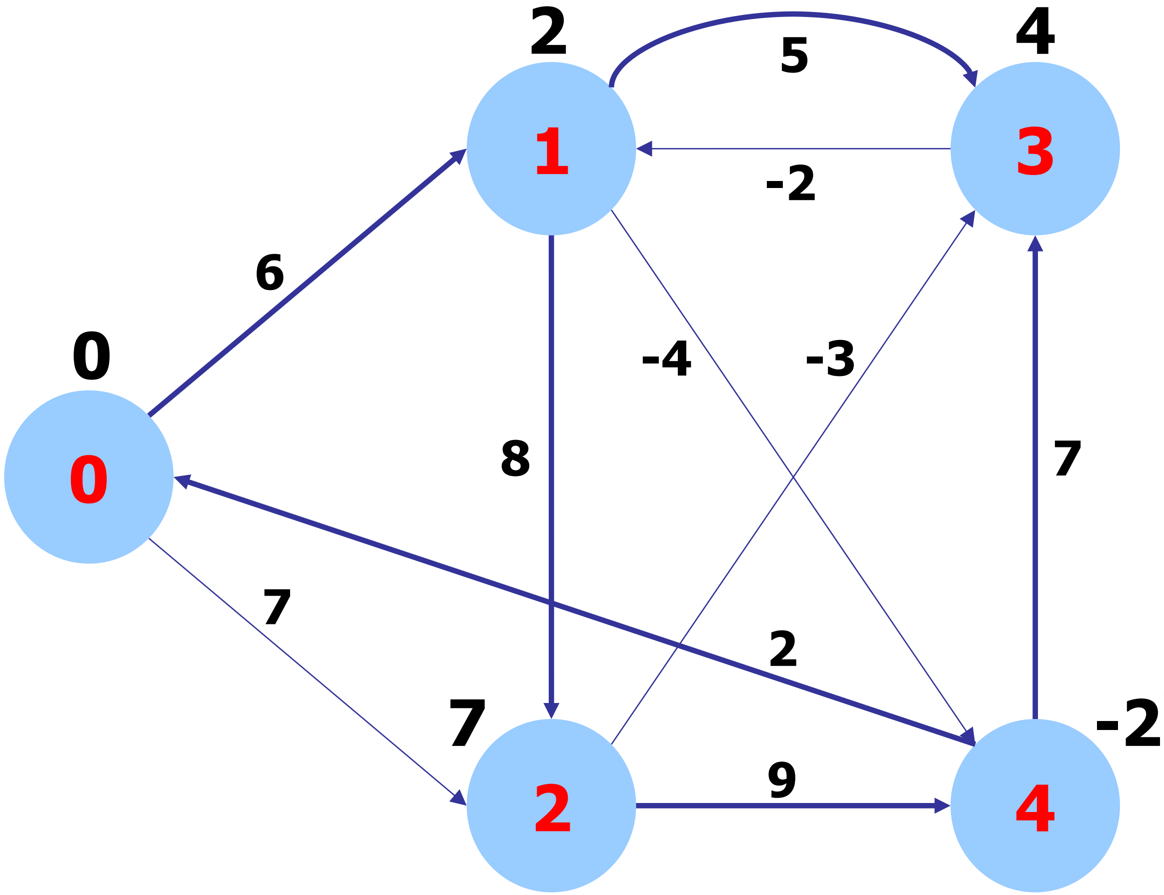
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)


2
5
4
1
3
-2
6
0
-4
-3
8
7
0
7
2
7
2
2
9
4
Passo 4
(u,v)
(u,x)
(u,y)
(v,u)
(x,v)
(x,y)
(y,v)
(y,z)
(z,u)
(z,x)



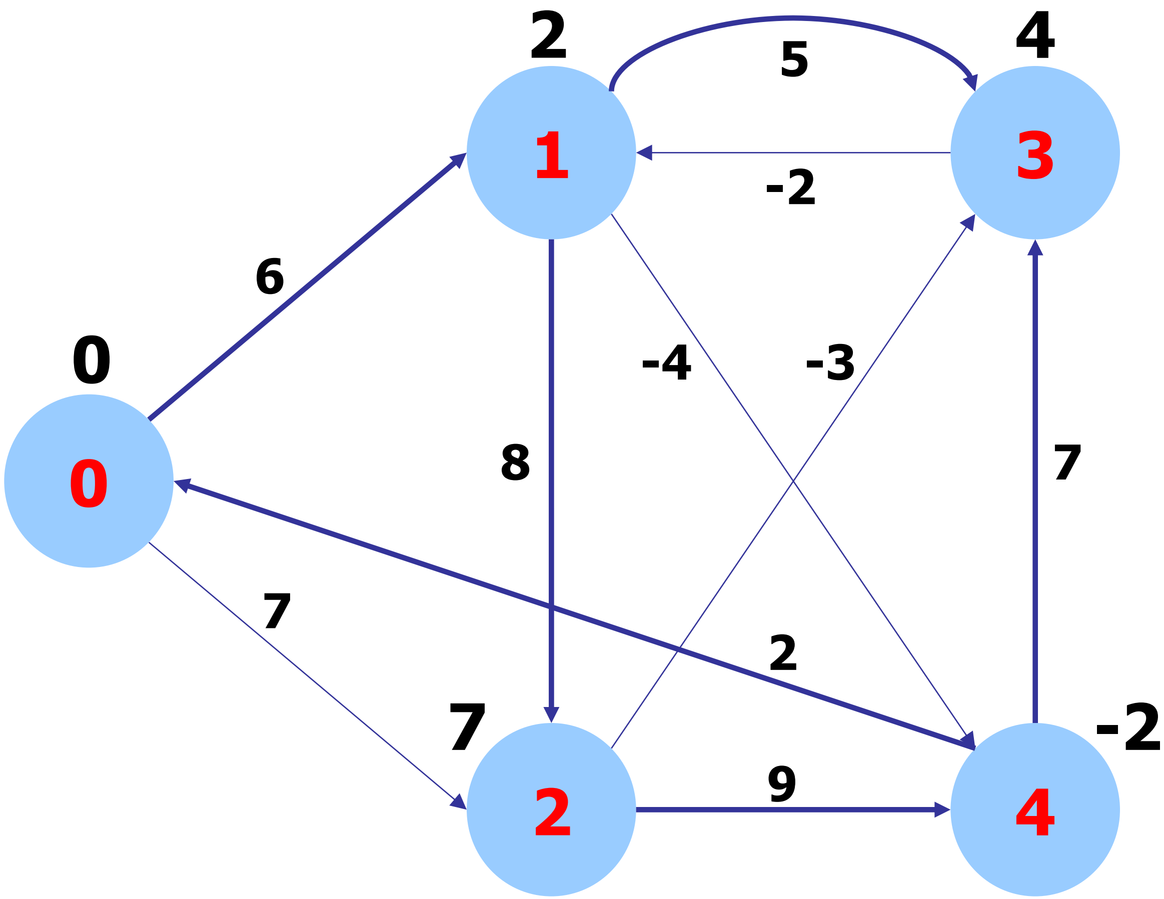
Passo 4

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



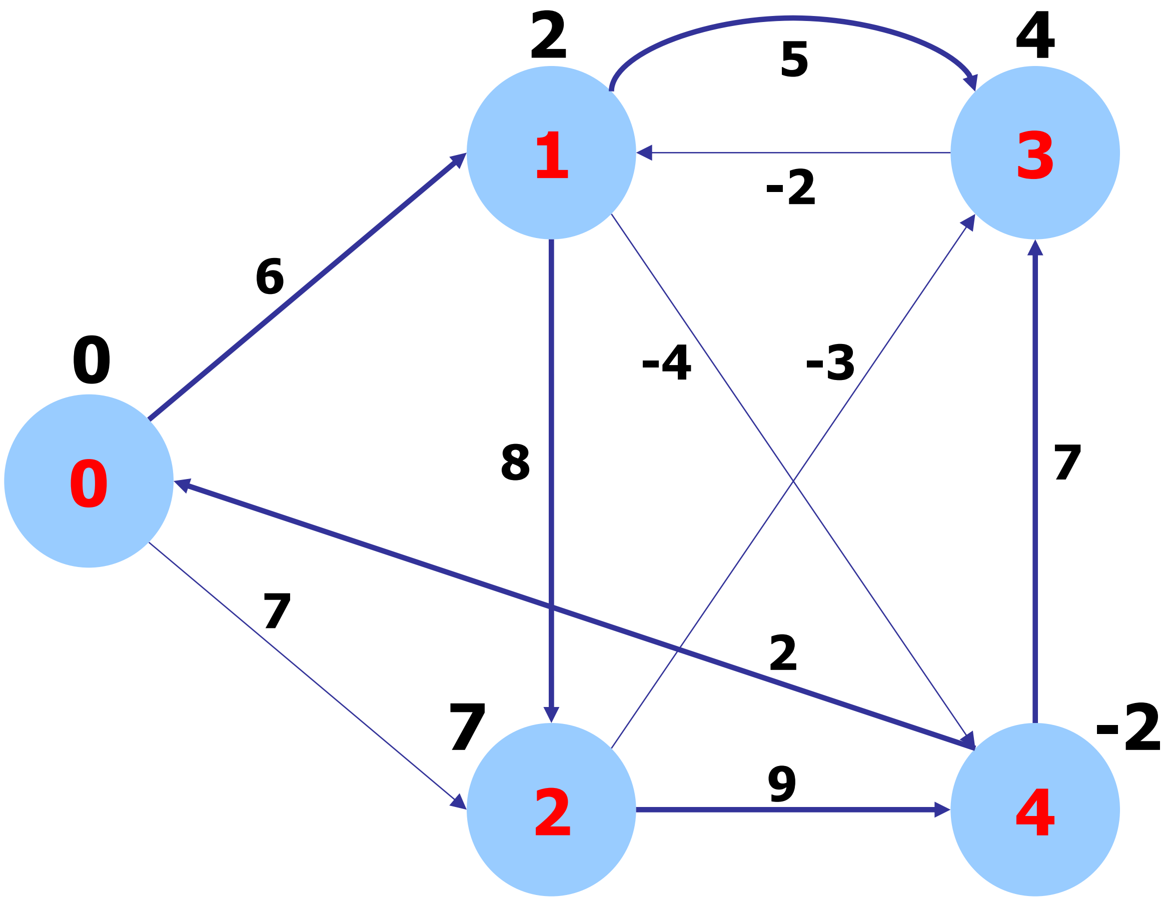
Passo 4

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



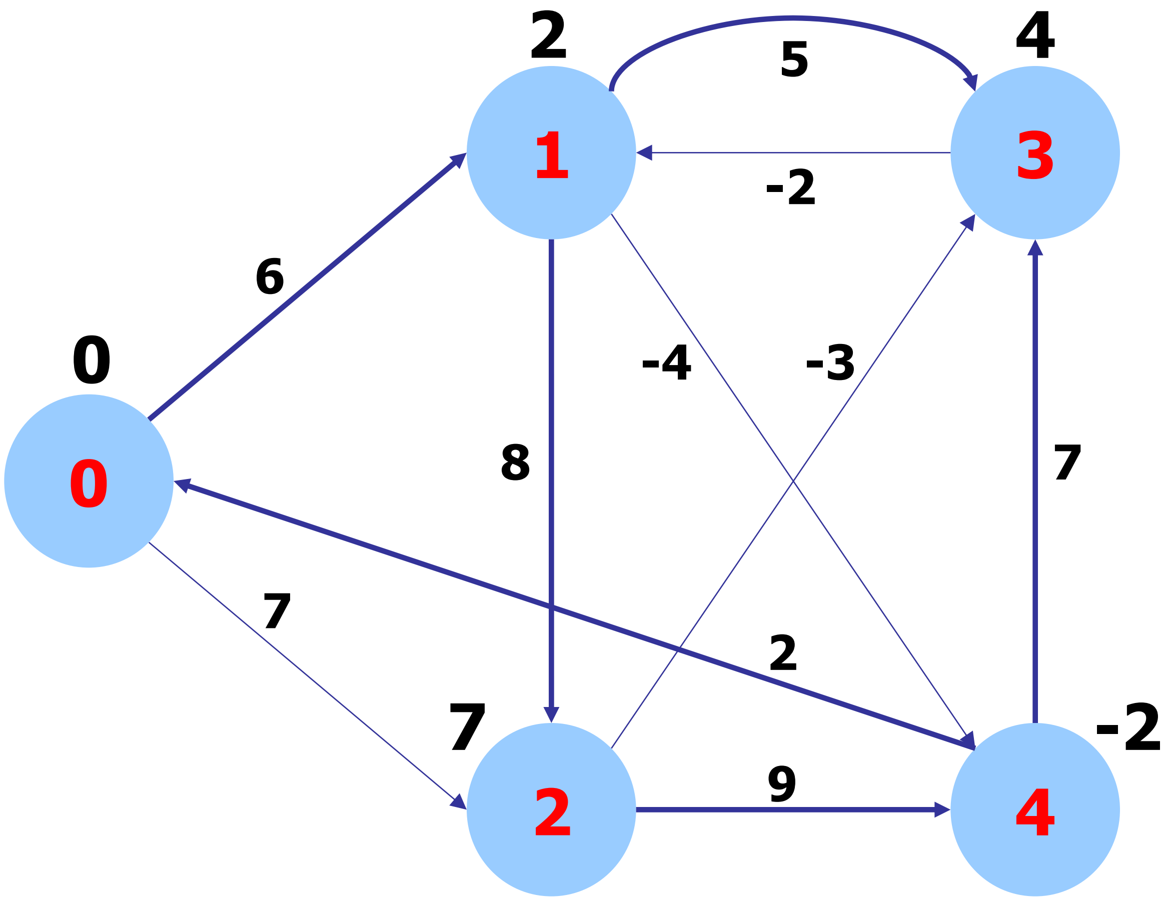
Passo 4

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



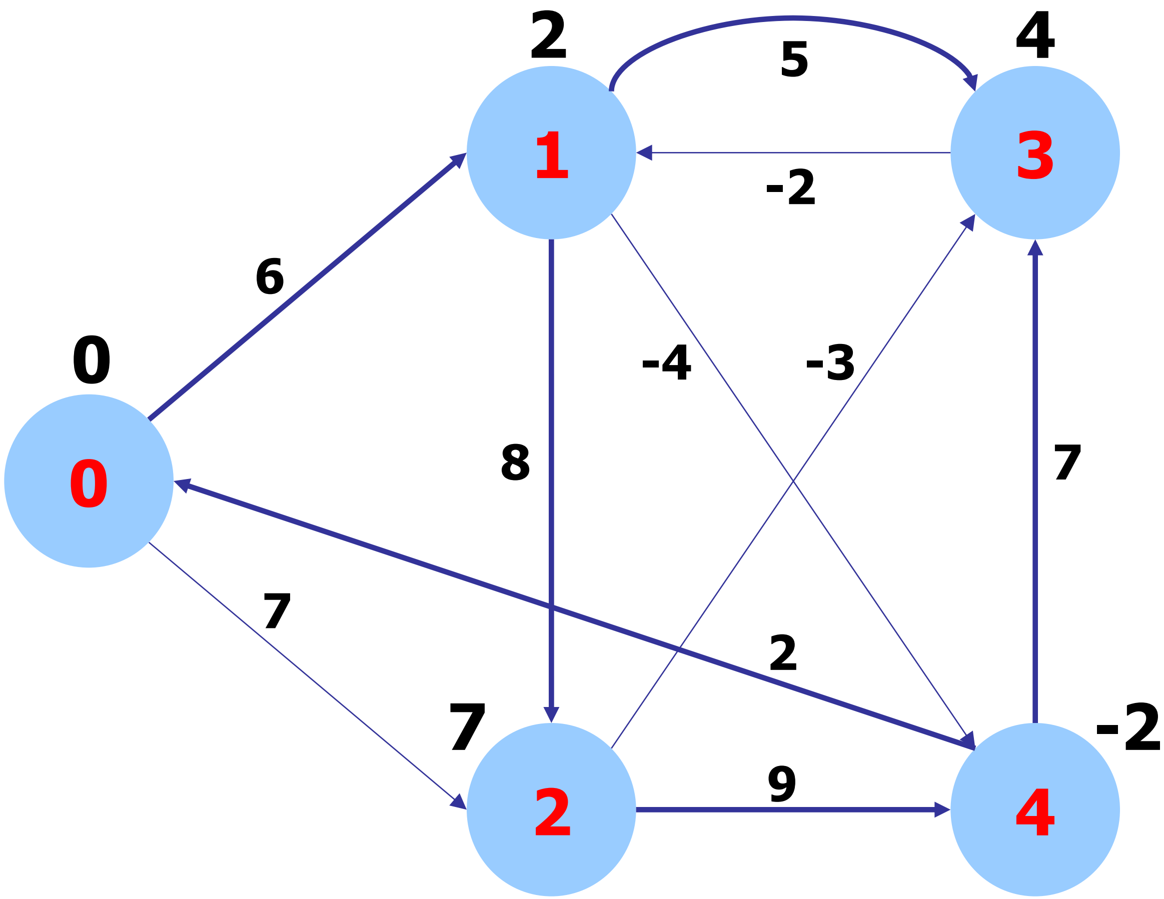
Passo 4

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



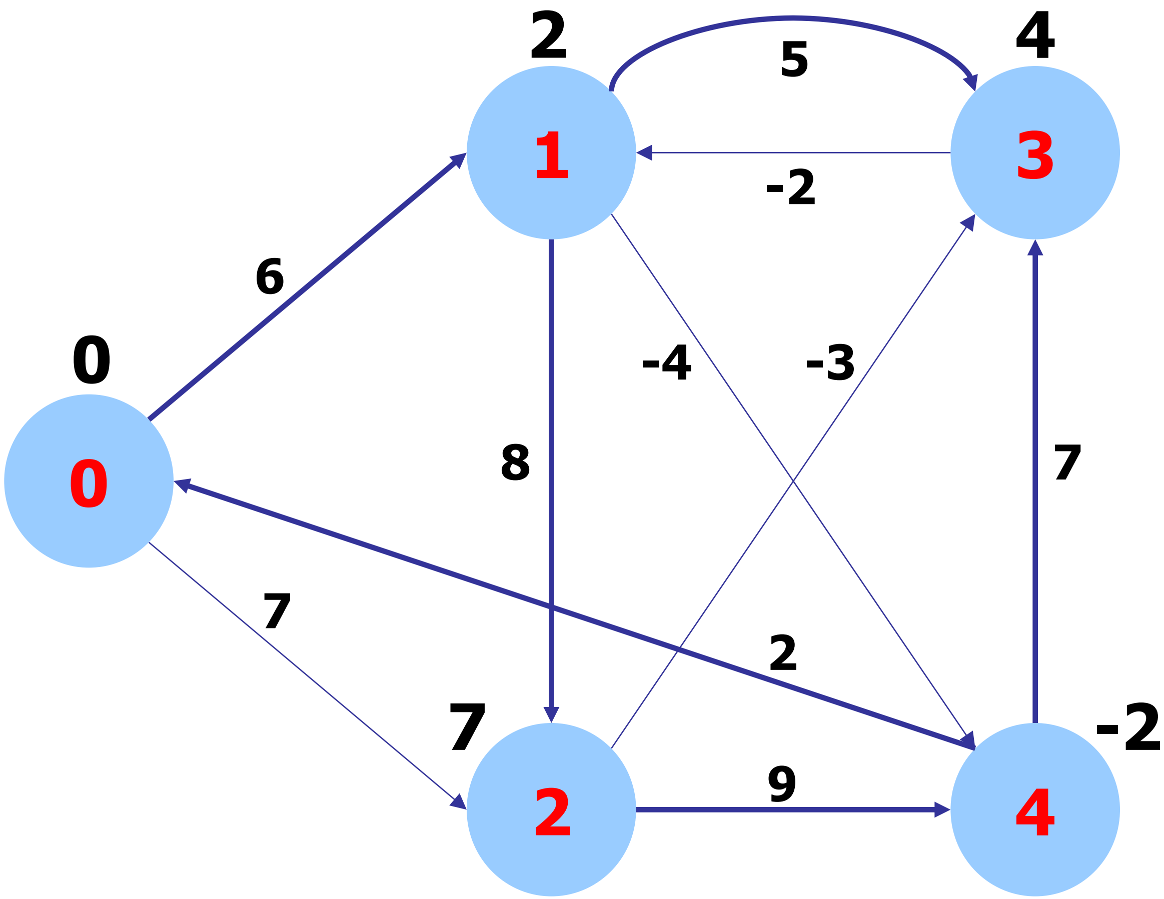
Passo 4

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



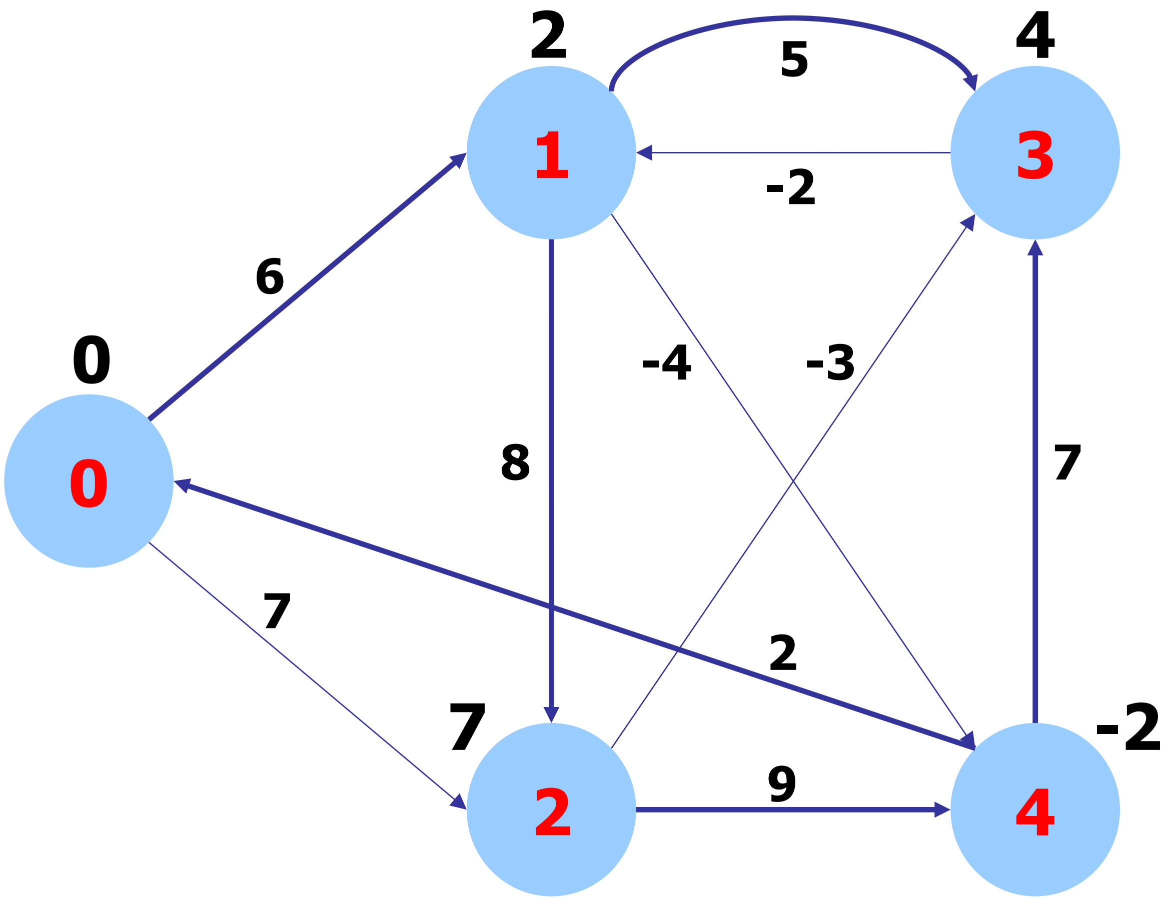
Passo 4

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



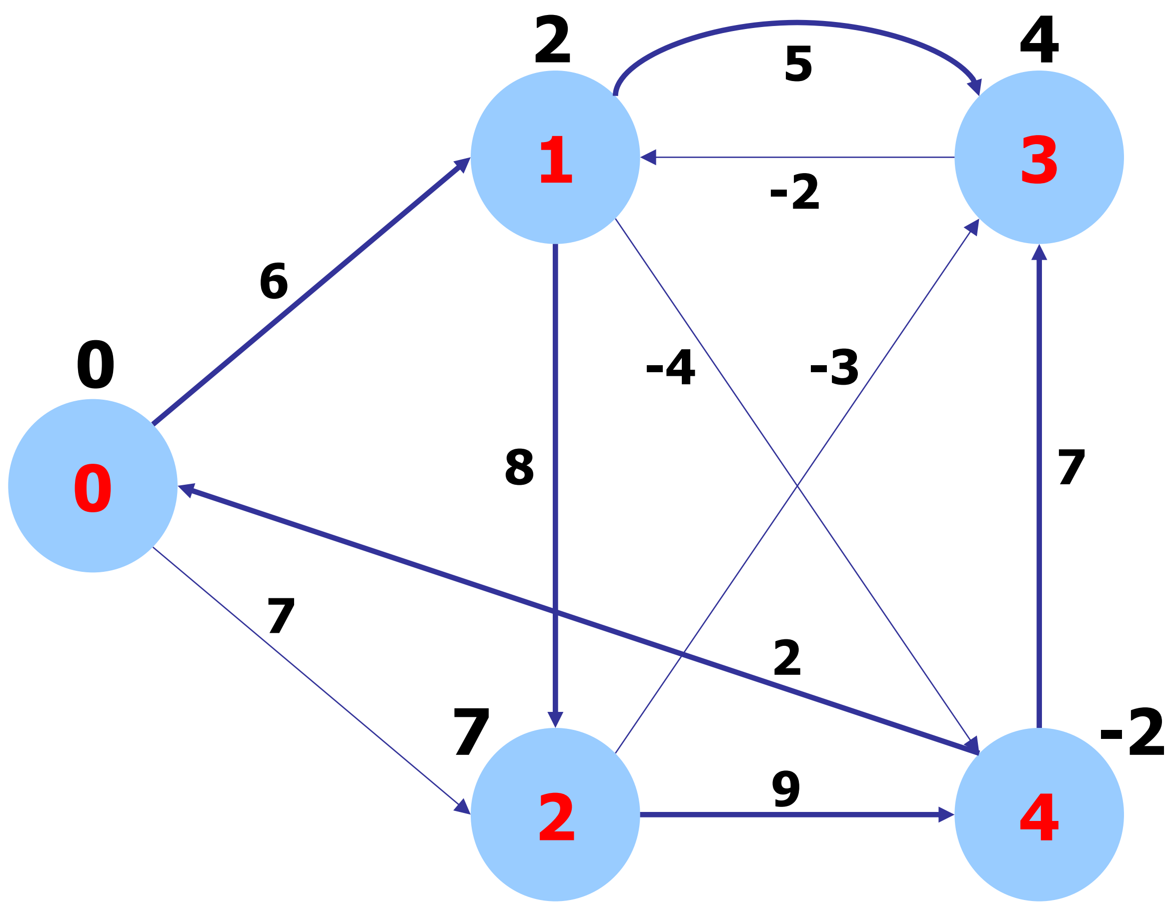
Passo 4

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



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|esimio passo di rilassamento non diminuisce alcuna stima:

terminazione con soluzione ottima.

Complessità

- Inizializzazione
- |V|-1 passi di rilassamento sugli archi
- |V|esimo rilassamento

O(|V|)

O(|V| |E|)

O(|E|)

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

```
void GRAPHspBF(Graph G){
int v, w, negcycfound;
link t;
char name[MAX];
int *st, *mindist;
st = malloc(G->V*sizeof(int));
mindist = malloc(G->V*sizeof(int));

printf("Insert start node: "); scanf("%s", name);

int s = STsearch(G->tab, name);
if (s == -1) { printf("Node doesn't exist\n"); return; }
for ( v = 0; v < G->V; v++) {
  st[v]= -1;
  mindist[v] = maxWT;
}
mindist[s] = 0;
st[s] = s;
```

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

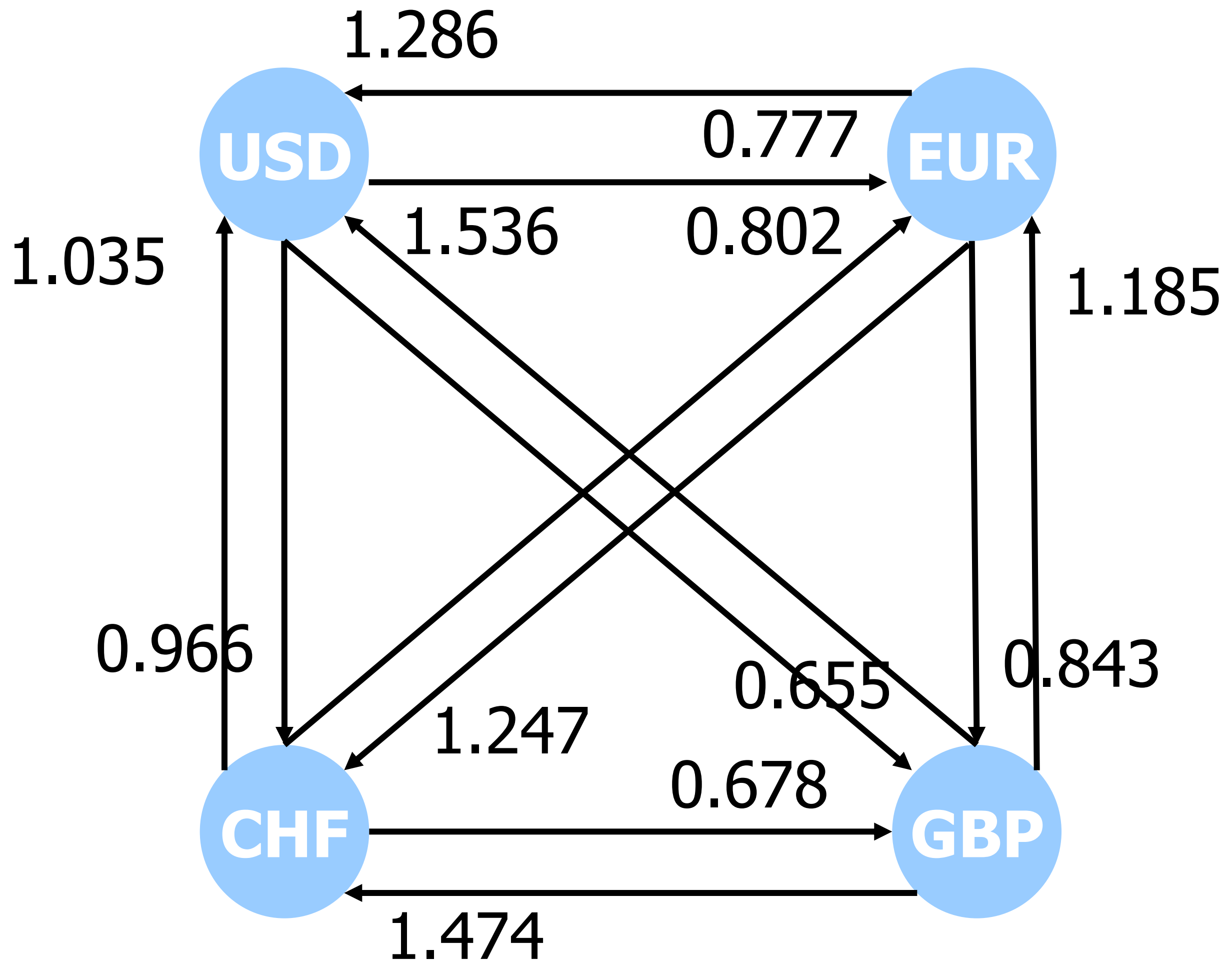
```
  if (negcycfound == 0) {
    printf("\n Shortest path tree\n");
    for (v = 0; v < G->V; v++)
      printf("Parent of %s is %s \n", STretrieve(G->tab, v),
              STretrieve(G->tab, st[v]));
    printf("\n Min.dist. from %s\n", STretrieve(G->tab, s));
    for (v = 0; v < G->V; v++)
     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

Applicazione: arbitrage

In Economia e Finanza si definisce «arbitrage» la possibilità di guadagno a costo zero e senza rischi dovuta alle differenze tra i mercati.

Esempio (semplificato): cambi delle valute:

	EUR	USD	GBP	CHF
EUR	1.000	1.286	0.843	1.247
USD	0.777	1.000	0.655	0.966
GBP	1.185	1.526	1.000	1.474
CHF	0.802	1.035	0.678	1.000


1.286
0.777
USD
EUR
1.536
0.802
1.035
1.185
0.966
0.655
0.843
1.247
0.678
CHF
GBP
1.474

1000 USD = 777 EUR = 655,11 GBP = 1006,10 USD

Guadagno di 6,10 USD $\Rightarrow$ arbitrage

Sul ciclo

USD – EUR – GBP – USD

il prodotto dei tassi di cambio è

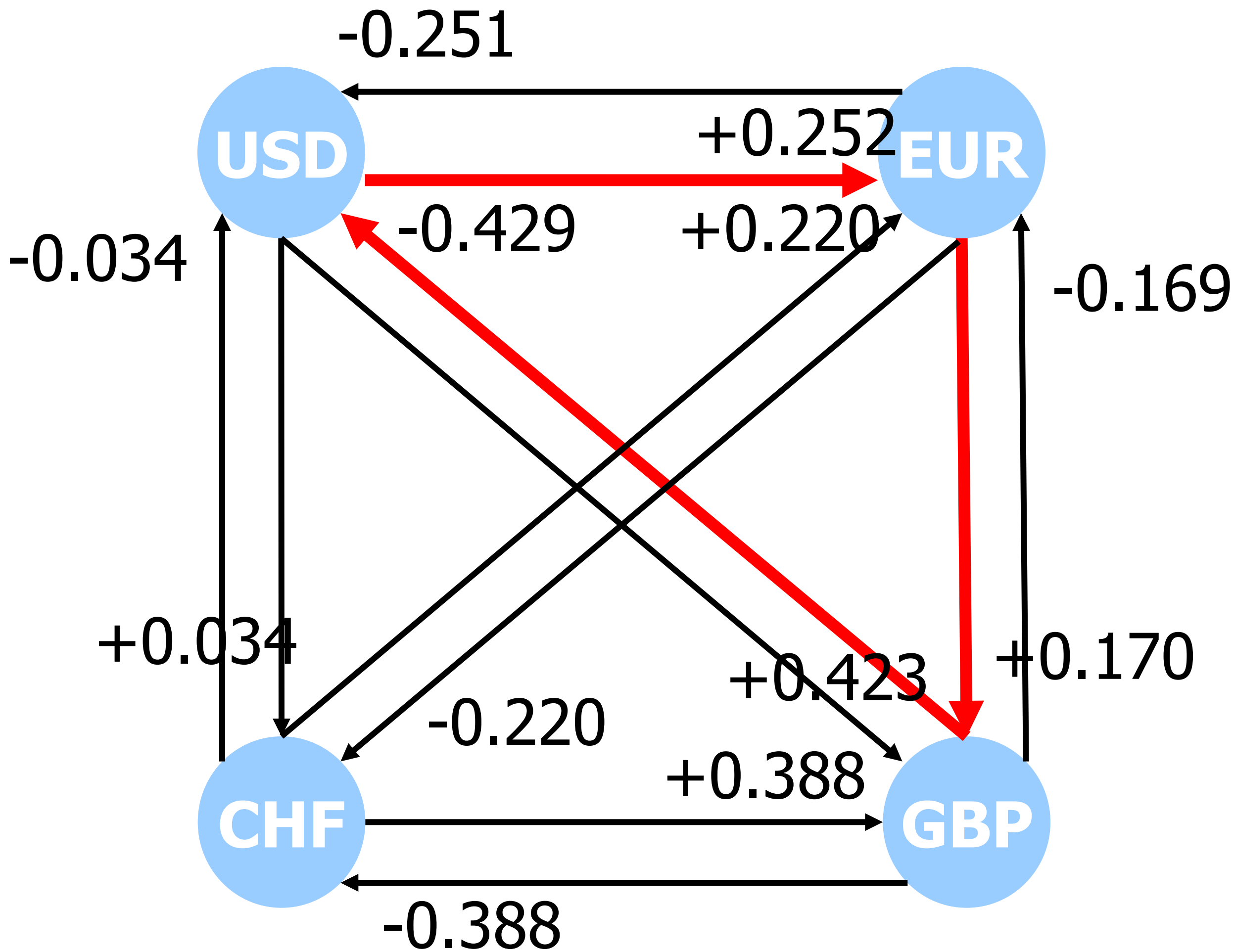
$$0.777 * 0.843 * 1.536 = 1{,}0061 > 1.0$$

$$\Downarrow$$

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
- si può applicare l'algoritmo di Bellman-Ford



+0.220
+0.170
-0.429
=
-0.039

Riferimenti

- Principi:
  - Sedgewick Part 5 21.1
  - Cormen 25.1
- Algoritmo di Dijkstra:
  - Sedgewick Part 5 21.2
  - Cormen 25.2
- Cammini minimi e massimi in DAG:
  - Sedgewick Part 5 21.4
  - Cormen 25.4
- Algoritmo di Bellman-Ford:
  - Sedgewick Part 5 21.7
  - Cormen 25.3