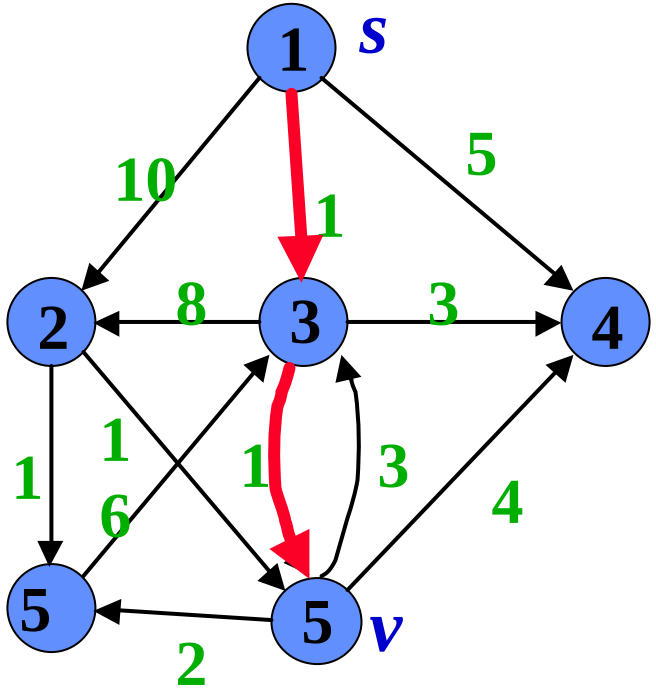
Algoritmi e Strutture dati Mod B

Grafi: Percorsi Minimi

(parte I)

Grafi: Percorsi minimi

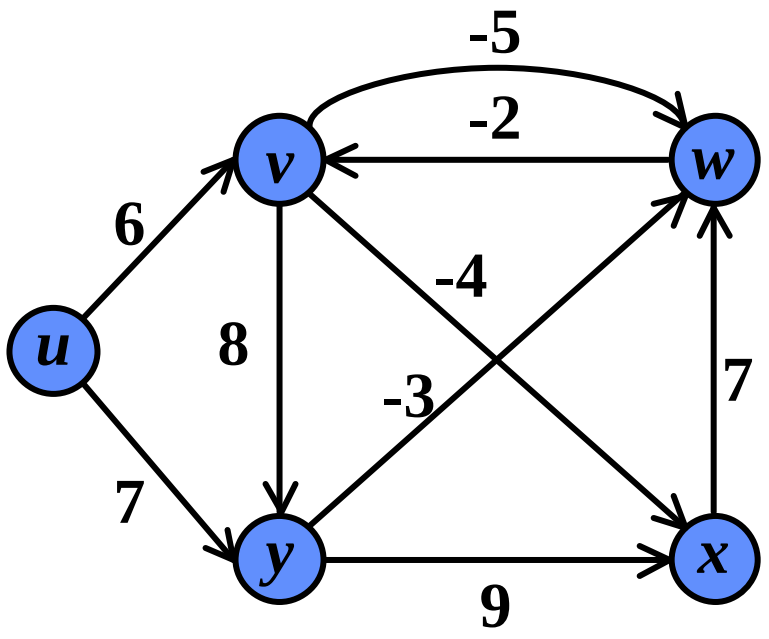
Un *percorso minimo* in un *grafo* $G=<V,E>$ grafo pesato orientato, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali tra due vertici *s* e *v*, è un percorso da *s* a *v* tale che la *somma dei pesi degli archi* che formano il percorso sia *minima*.



Percorsi minimi: pesi negativi

Qual è il *percorso minimo* tra *u* e *x* nel grafo sottostante?

Percorso	Peso
<*u*,*v*,*x*>	2
<*u*,*v*,*w*,*v*,*x*>	-5
<*u*,*v*,*w*,*v*,*w*,*v*,*x*>	-12
<*u*,*v*,*w*,*v*,*w*,*v*,*w*,*v*,*x*>	-19
…	…
…	…



***Non* esiste alcun *percorso minimo* tra *u* e *x*!**

Grafi: Percorsi minimi

Lemma 1: **Dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$, sia $p=\langle v_1,\ldots,v_k\rangle$ il *percorso minimo* tra v_1 e v_k e per ogni i e j ($1 \le i \le j \le k$) sia $p_{ij}=\langle v_i,\ldots,v_j\rangle$ un sottopercorso di p tra v_i e v_j.**
Allora, $p_{ij}=\langle v_i,\ldots,v_j\rangle$ è un *percorso minimo* tra v_i e v_j.

Proprietà di sottostruttura ottima
(per dimostrazione vedere Cormen)

Grafi: Percorsi minim

Per dimostrazioni vedere Cormen

Corollario 1: **Sia $G = (V, E)$ un grafo pesato orientato, con funzione di peso $w: E \to \mathbb{R}$. Supponiamo che un *percorso minimo* p dalla *sorgente* s ad un vertice v possa essere decomposto in $s \xrightarrow{p'} u \to v$ per qualche vertice u e percorso p'. Allora, il *peso del percorso minimo* tra s e v è $d(s,v) = d(s,u) + w(u,v)$.**

Lemma 2: **Dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$, e un vertice sorgente s, allora per ogni arco (u,v) in E vale $d(s,v) \leq d(s,u) + w(u,v)$.**

Albero dei percorsi minimi

Definizione: **Sia $G = (V, E)$ un grafo pesato orientato, con funzione di peso $w: E \to \mathbb{R}$. Un *albero dei percorsi minimi* con radice s è un *sottografo orientato* $G' = (V', E')$ di G con $V' \subseteq V$ e $E' \subseteq E$ e tale che:**

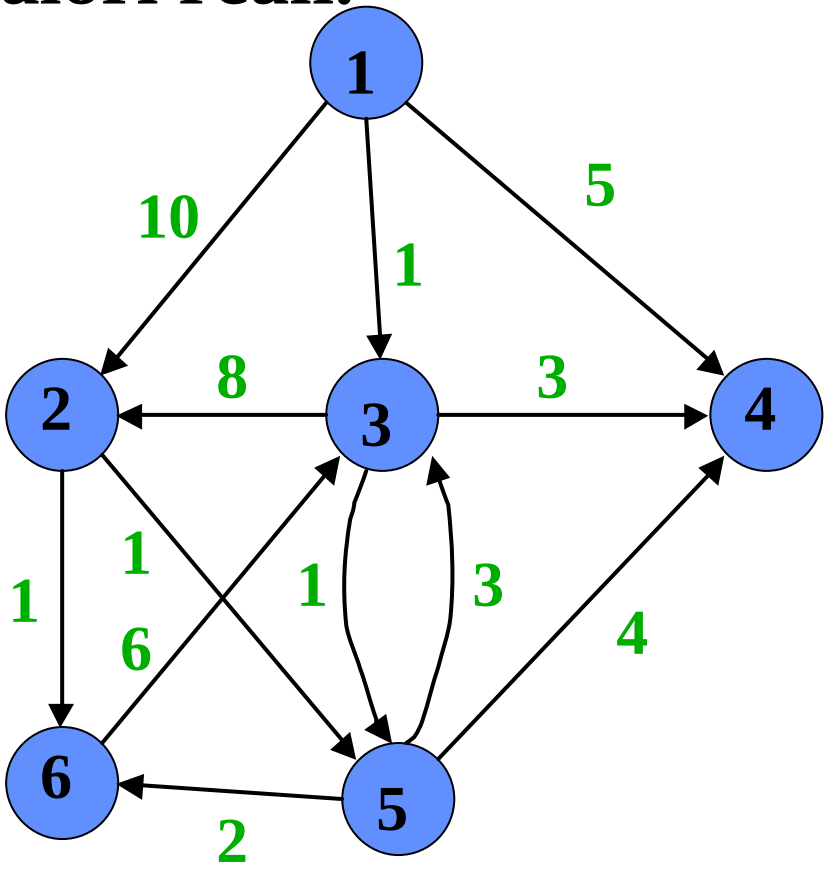
1. **V' è l'insieme di *vertici raggiungibili* da s**
2. **G' forma un *albero radicato* in s**
3. **per ogni $v \in V'$, l'*unico percorso semplice* da s a v è un *percorso minimo*.**

Grafi: Percorsi minimi

- **Peso unitario**
  - **Breadth First Search**
- **Pesi non negativi**
  - **Algoritmo di Dijkstra**
- **Pesi negativi con cicli non negativi**
  - **Algoritmo di Bellman-Ford**
- **Cicli negativi**
  - **Nessuna soluzione**

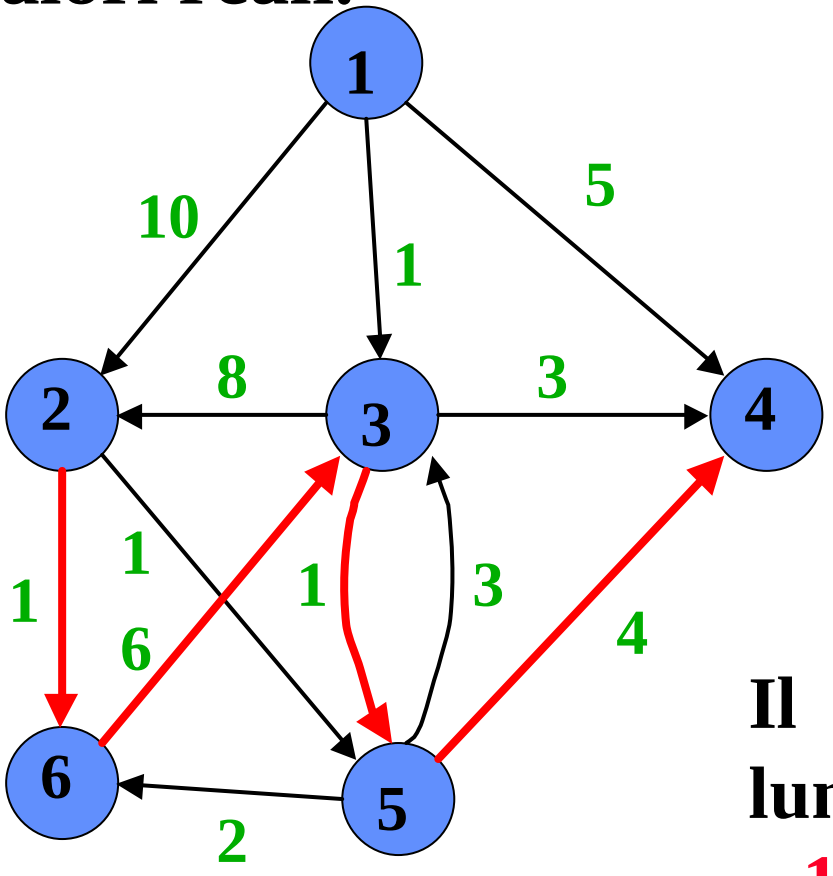
Grafi: Percorsi minimi (Dijkstra)

Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali.



Grafi: Percorsi minimi (Dijkstra)

Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali.



Il *peso* di un percorso

$$p = (v_1, v_2, \ldots, v_k)$$

è

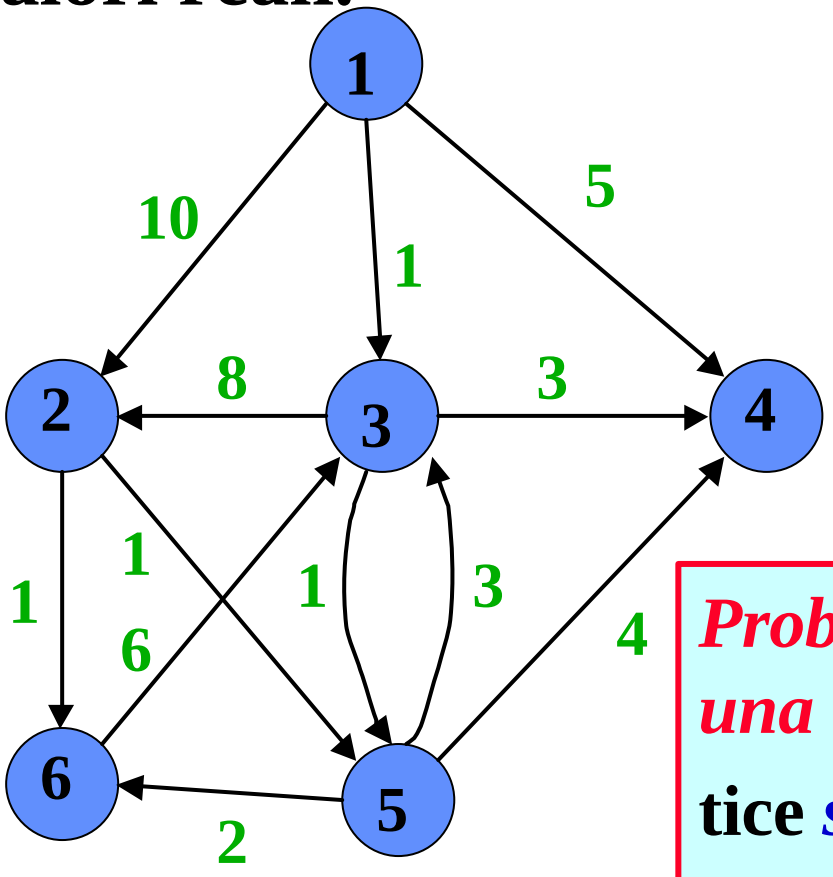
$$\sum_{i=1}^{k-1} w(v_i, v_{i+1}).$$

Il *peso* del percorso lungo gli *archi rossi* è

$$1 + 6 + 1 + 4 = 12.$$

Grafi: Percorsi minimi (Dijkstra)

Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali.



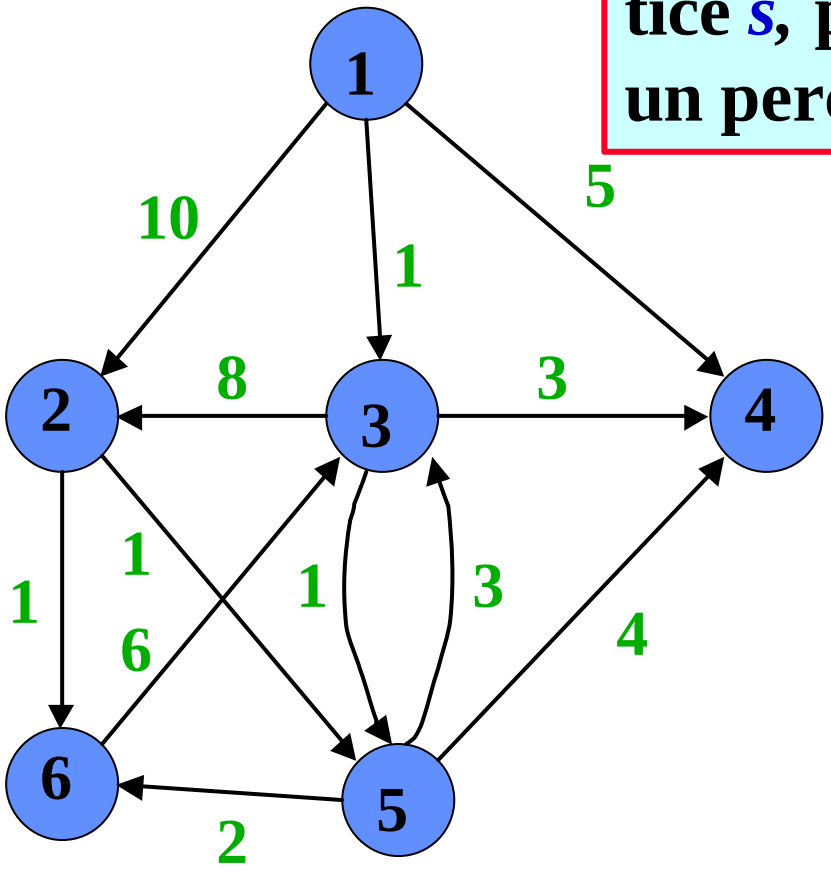
Il *peso* di un percorso $p = (v_1, v_2, \ldots, v_k)$ è

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

***Problema del percorso minimo da una singola sorgente*: dato un vertice s, per ogni vertice $v \in V$ trovare un percorso minimo da s a v.**

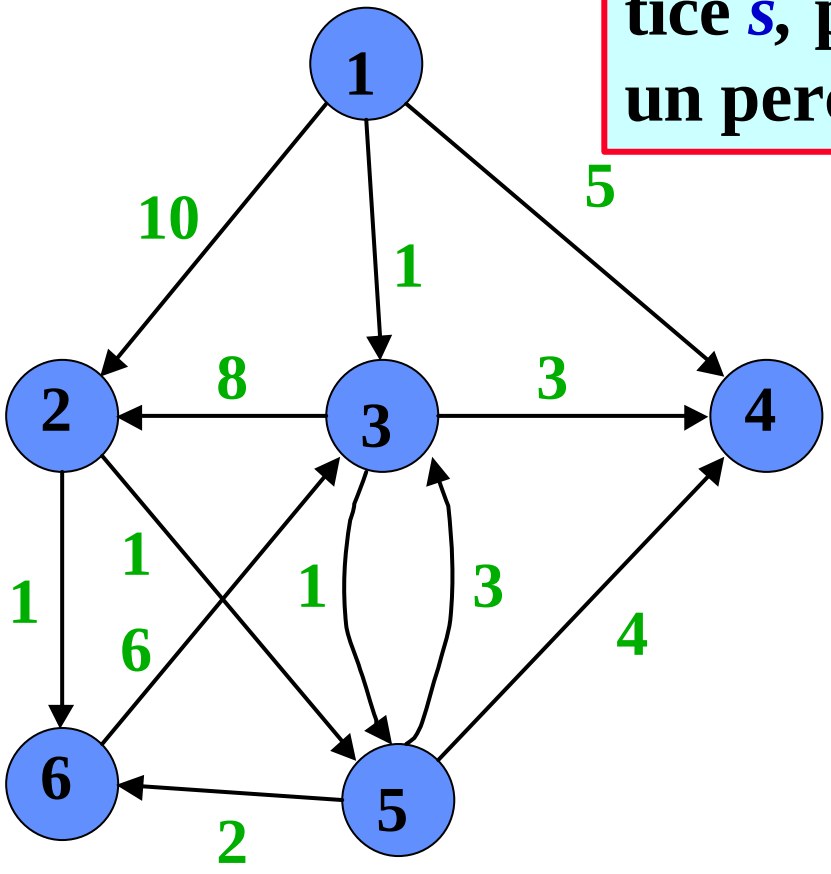
Grafi: Percorsi minimi (Dijkstra)

***Problema del percorso minimo da una singola sorgente*: dato un vertice *s*, per ogni vertice *v* Î *V* trovare un percorso minimo da *s* a *v*.**



***L'algoritmo di Dijkstra* risolve il problema in modo efficiente nel caso in cui *tutti i pesi siano non-negativi*, come nell'esempio del grafo.**

Grafi: Percorsi minimi (Dijkstra)



***Problema del percorso minimo da una singola sorgente*: dato un vertice *s*, per ogni vertice *v* Î *V* trovare un percorso minimo da *s* a *v*.**

***L'algoritmo di Dijkstra* risolve il problema in modo efficiente nel caso in cui *tutti i pesi siano non-negativi*, come nell'esempio del grafo.**

- Utilizza un campo *d*[*v*] per la stima della *distanza minima*

- Utilizza un campo *p*[*v*] per il *nodo predecessore* di *v*

Sottografo dei predecessori (Dijkstra)

- **L'*algoritmo di Dijkstra* sul grafo $G=<V,E>$ costruisce in $p[]$ il *sottografo dei predecessori* denotato con $G_p=<V_p,E_p>$, dove:**

$$V_p = \{ v \in V : p[v] \neq Nil \} \cup \{s\}$$

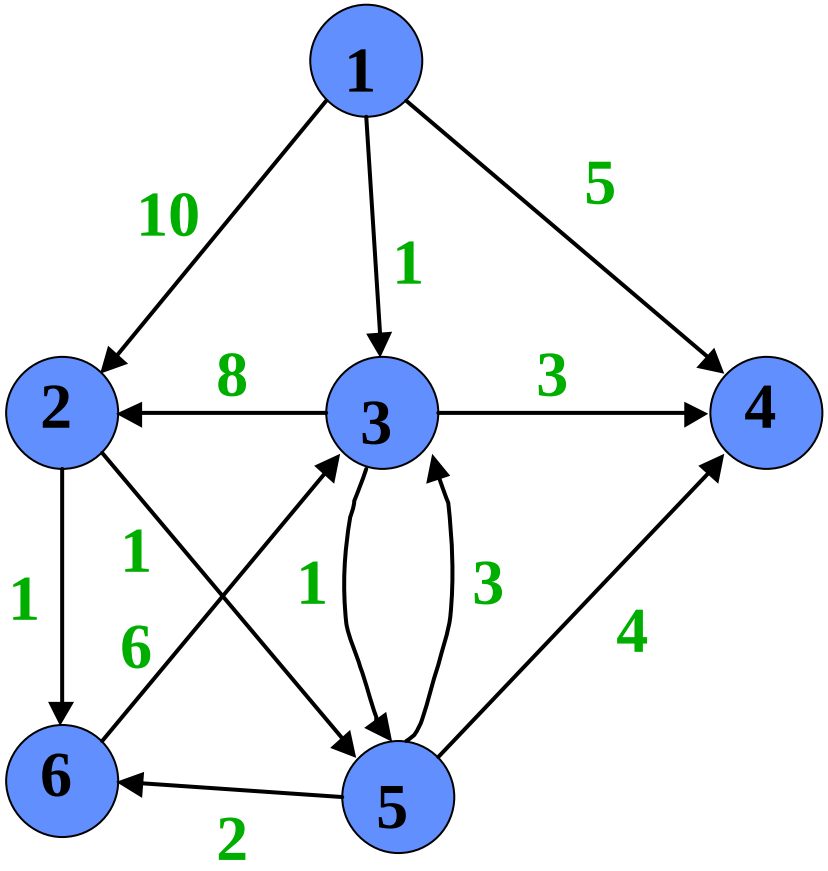
$$E_p = \{ (p[v],v) \in E : v \in V_p - \{s\} \}$$

Il *sottografo dei predecessori* è definito come per *BFS*
La differenza è che ora verrà costruito in modo che i *percorsi che individua* siano quelli con *peso minimo* (*non* col numero minimo di archi)

Si dimostra che il *sottografo dei predecessori* costruito dall'*algoritmo di Dijkstra* è un *albero dei percorsi minimi*

Grafi: Percorsi minimi (Dijkstra)

Inizializzazione del grafo



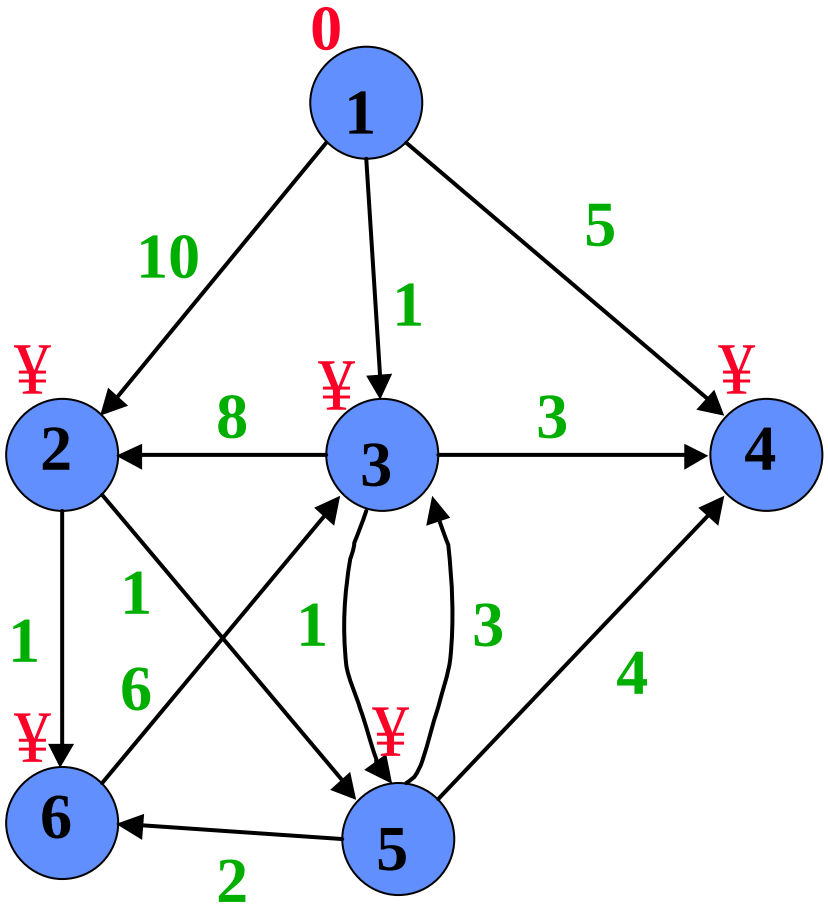
`Inizializza(G,s)`

1 La stima della distanza d[s] viene posta a 0

2 Tutte le altre stime delle disanze d[v] sono poste a ¥

Grafi: Percorsi minimi (Dijkstra)

Inizializzazione del grafo




```
Inizializza(G,s)
```

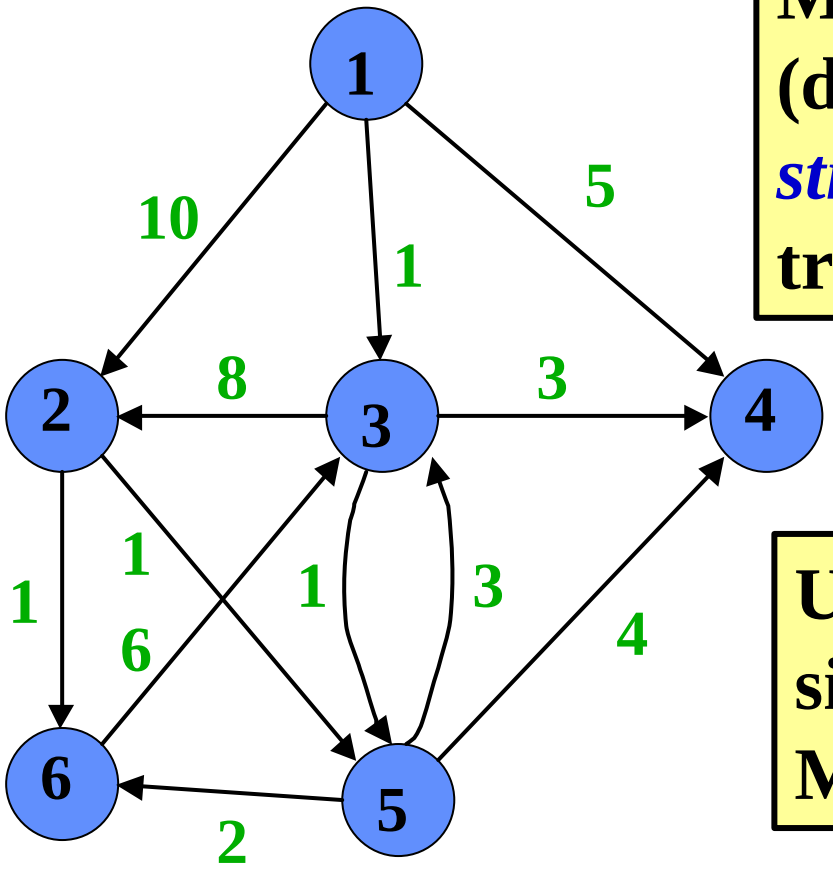
1 La stima della distanza d[s] viene posta a 0

2 Tutte le altre stime delle disanze d[v] sono poste a ¥

3 I predecessori p[v] sono posti a Nil

Grafi: Percorsi minimi (Dijkstra)

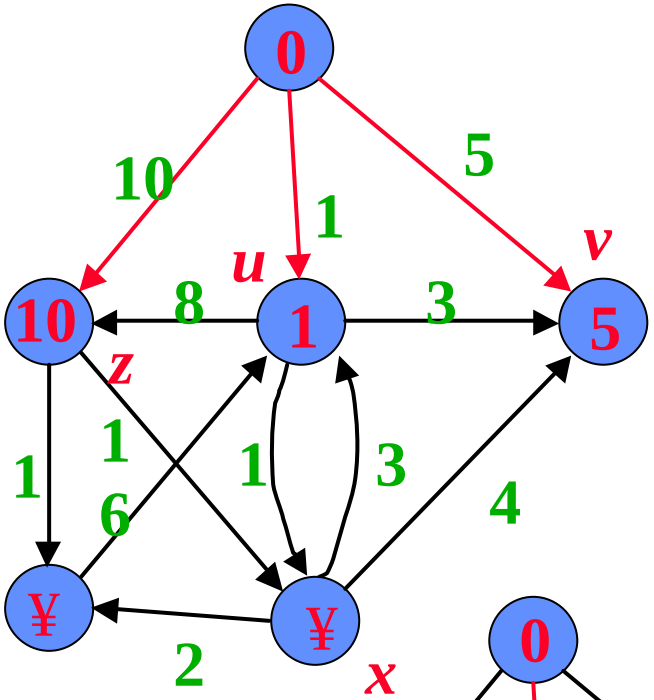
Rilassamento degli archi

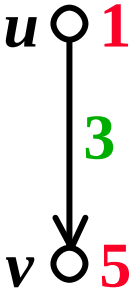


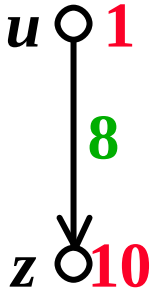
Meccanismo di *aggiustamento* (diminuzione) progressivo *delle stime d[v]* delle *distanze minime* tra *s* e gli altri nodi *v*.

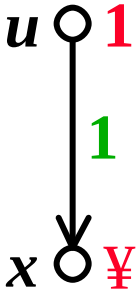
Utilizza la funzione di peso *w* e si applica agli *archi del grafo*. Modifica sia *d[v]* che *p[v]*.

Rilassamento

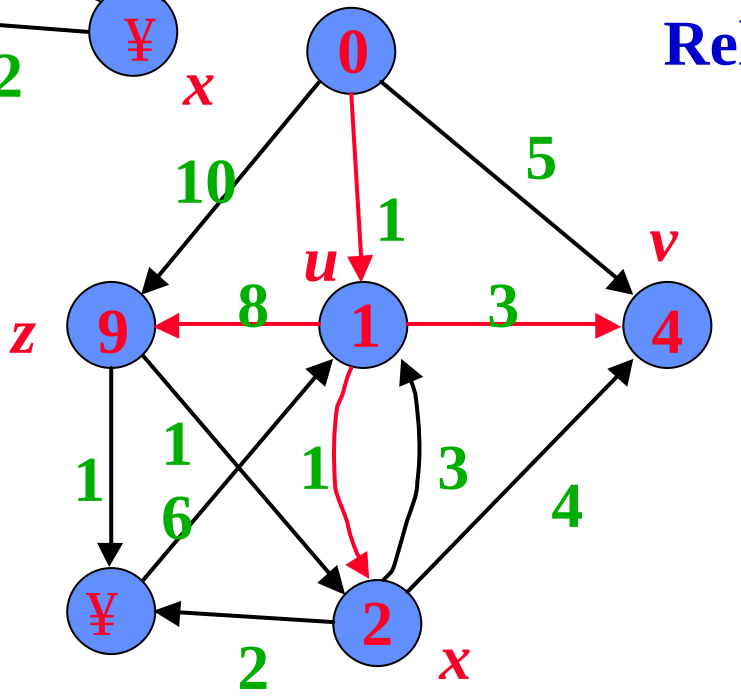


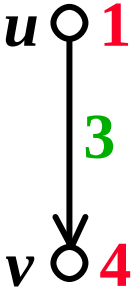


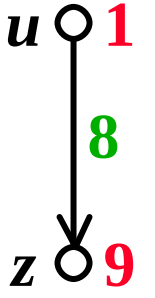


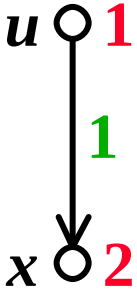


Relax(*u*,*v*,*w*) **Relax(*u*,*z*,*w*)** **Relax(*u*,*x*,*w*)**

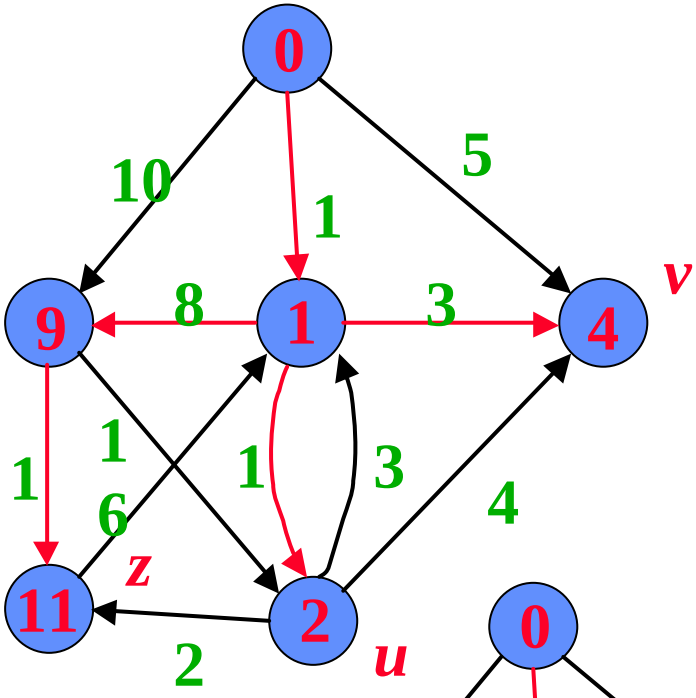


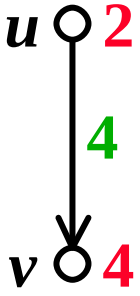


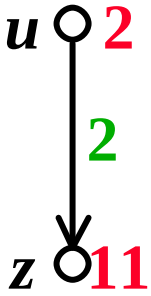




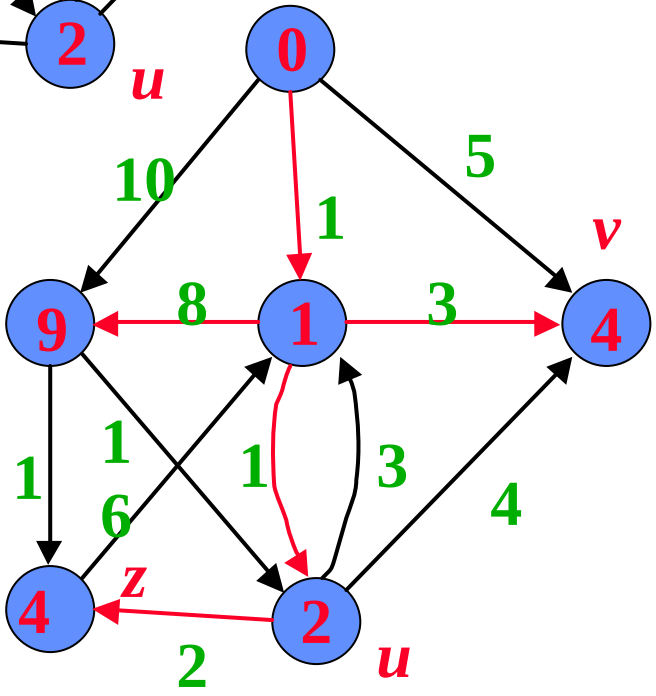
Rilassamento

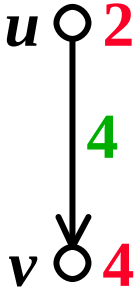


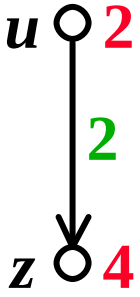




Relax(*u*,*v*,*w*) **Relax(*u*,*z*,*w*)**







Rilassamento

Verifica **se è possibile** ***ottenere un percorso migliore*** **tra** ***s*** **e** ***v*** **passando per il vertice** ***u***

- ***d*[*v*]: *estremo superiore*** **della lunghezza del** ***percorso minimo*** **tra** ***s*** **a** ***v*** **(*s* è la *sorgente*)**
- ***p*[*v*]: il *vertice predecessore*** **di** ***v*** **nel** ***percorso minimo corrente*** **tra** ***s*** **e** ***v*** **(padre di *v*)**
- ***w*(*u*,*v*): *peso*** **dell'arco** ***(u,v)***

```
Relax(u,v,w)
    if d[v] > d[u] + w(u,v)
      then d[v] = d[u] + w(u,v)
           p[v] = u
```

Rilassamento: proprietà

Lemma 3: **Sia $G = (V, E)$ un grafo orientato pesato, con funzione di peso $w: E \to \mathbb{R}$.**
Sia inoltre $(u,v) \in E$ un arco di G. Allora, immediatamente dopo un rilassamento di (u,v) (`Relax(u,v,w)`), varrà $d[v] \leq d[u] + w(u,v)$.

Rilassamento: proprietà

Lemma 4: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Sia s la sorgente e il grafo sia inizializzato con una chiamata a `Inizializza(G,s)`.**

Allora, vale $d[v] \geq \delta(s,v)$ per ogni vertice v di G e tale *invariante* viene mantenuto da ogni sequenza di operazioni di *rilassamento*. Inoltre, appena $d[v] = \delta(s,v)$, $d[v]$ *non cambia più*.

Rilassamento: proprietà

Corollario 2: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Supponiamo che in G non esistano percorsi tra s e un vertice v.**

Allora dopo che grafo è stato inizializzato da `Inizializza(G,s)`, vale $d[v] = \delta(s,v)$ e questa *uguaglianza* viene mantenuto da ogni sequenza di operazioni di *rilassamento*.

Rilassamento: proprietà

Lemma 5: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Sia s la sorgente e $s \xrightarrow{p} u \to v$ sia un *percorso minimo* per qualche $u,v \in V$. Il grafo sia inizializzato con una chiamata a `Inizializza(G,s)` e venga applicata una sequenza di operazioni di *rilassamento* che includa `Relax(u,v,w)`.**

Se $d[u] = \delta(s,u)$ in qualunque momento *prima della chiamata*, allora vale $d[v] = \delta(s,v)$ *sempre dopo la chiamata*.

Rilassamento e percorsi minimi

Lemma 6: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ e sia s la sorgente.**

Allora, dopo che il grafo è stato inizializzato con una chiamata a `Inizializza(G,s)`, il *sottografo dei predecessori* G_p forma un *albero con radice* s, e qualunque sequenza di operazioni di *rilassamento* su G mantiene questa proprietà.

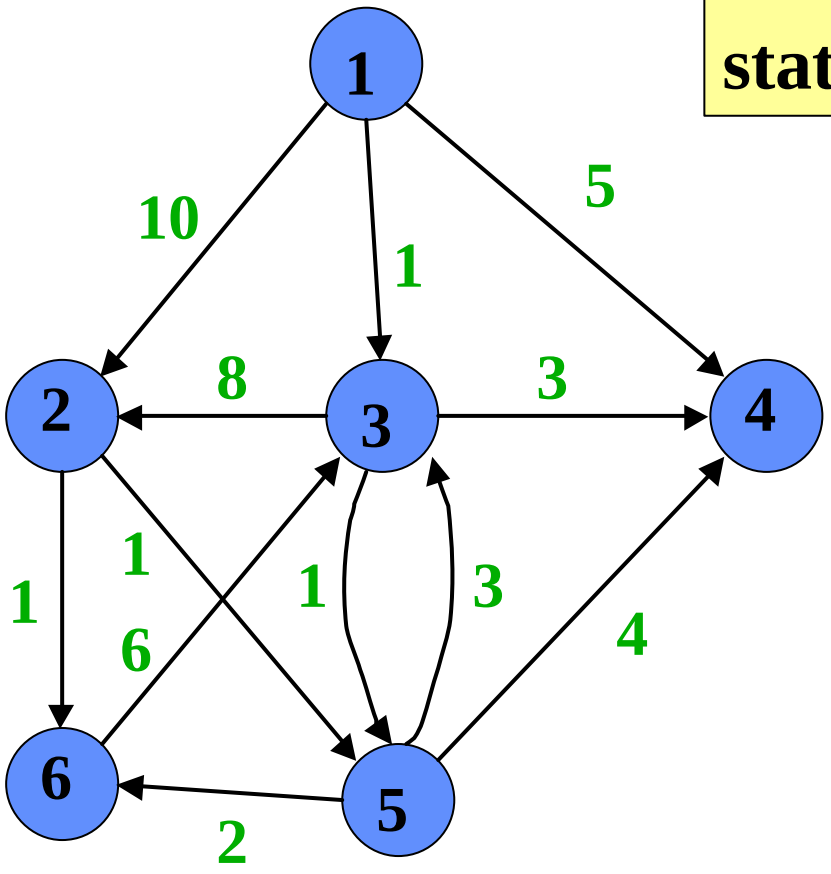
Rilassamento e percorsi minimi

Lemma 7: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Sia s la sorgente e G non contenga cicli di peso negativo.**

Il grafo sia inizializzato con una chiamata a `Inizializza(G,s)` e venga applicata una sequenza di operazioni di *rilassamento* che includa `Relax(u,v,w)` tale che $d[v] = \delta(s,v)$.

Allora il *sottografo dei predecessori* G_p è un *albero di cammini minimi* con radice s.

Grafi: Percorsi minimi (Dijkstra)

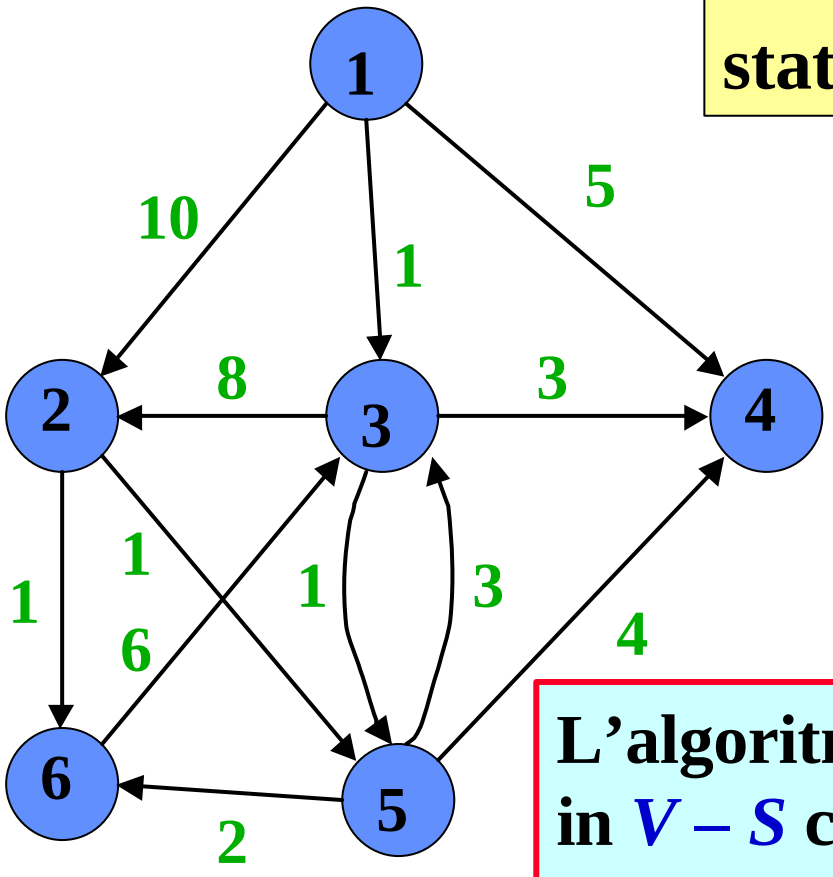


L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

Utilizza anche, per ogni vertice *v* non in *S*, un campo *d*[*v*] contenente ad ogni passo l'*estremo superiore* del peso del percorso minimo da *s* a *v*.

Grafi: Percorsi minimi (Dijkstra)

L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

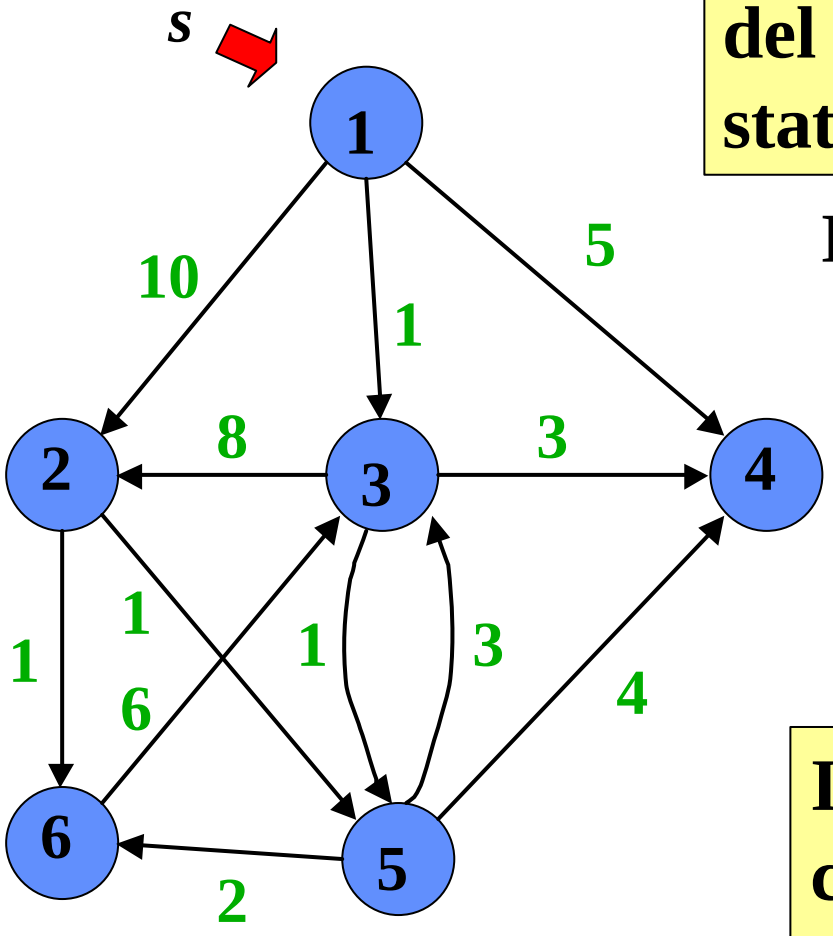


Utilizza anche, per ogni vertice *v* non in *S*, un campo *d*[*v*] contenente ad ogni passo l'*estremo superiore* del peso del percorso minimo da *s* a *v*.

L'algoritmo seleziona a turno il vertice *u* in *V* – *S* col *minimo valore d*[*u*], inserisce *u* in *S*, e *rilassa* tutti gli archi uscenti da *u*.

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

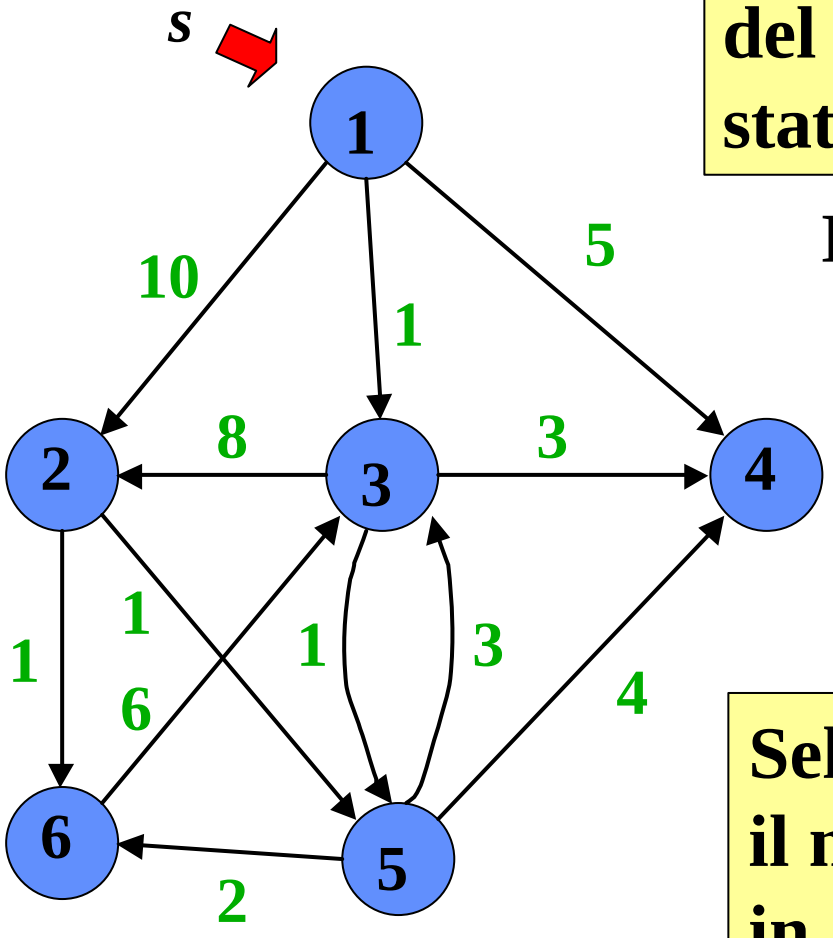
Inizialmente: *S* = { }.

v	1	2	3	4	5	6
d	0	¥	¥	¥	¥	¥

Inizializza una *coda a priorità* con tutti i vertici e i loro estremi superiori *d*[].

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

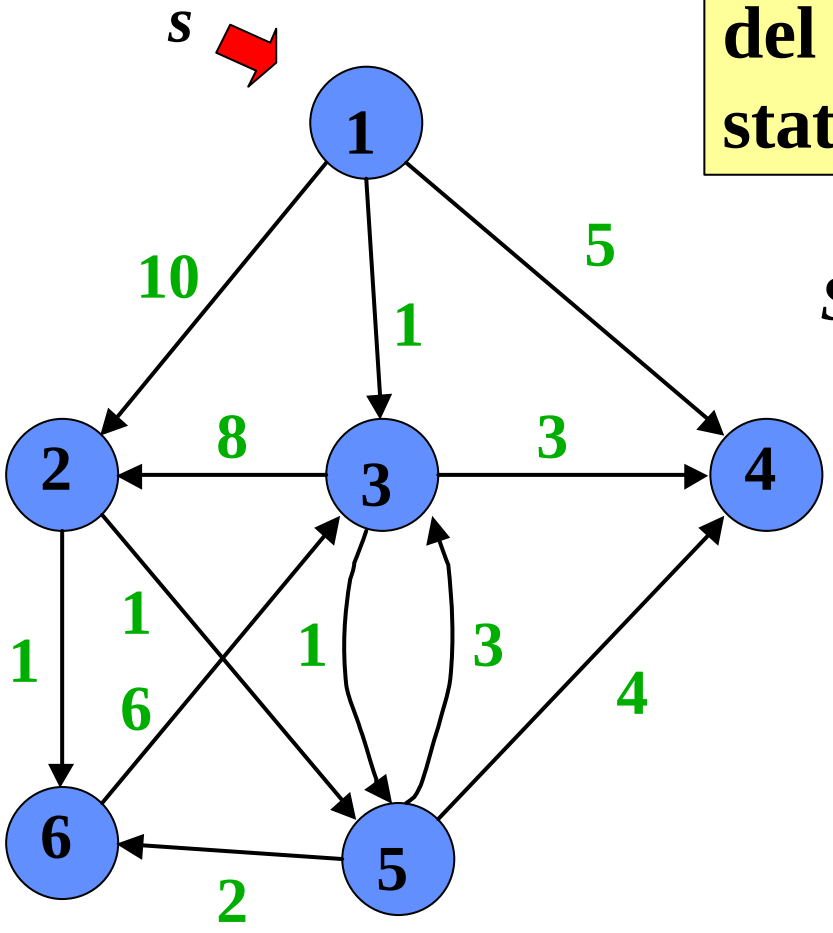
Inizialmente: *S* = { }.

v	1	2	3	4	5	6
d	0	¥	¥	¥	¥	¥

Seleziona il vertice *u* Î *V* – *S* con il minimo valore *d*[*v*], inserisce *u* in *S*, . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

1
0

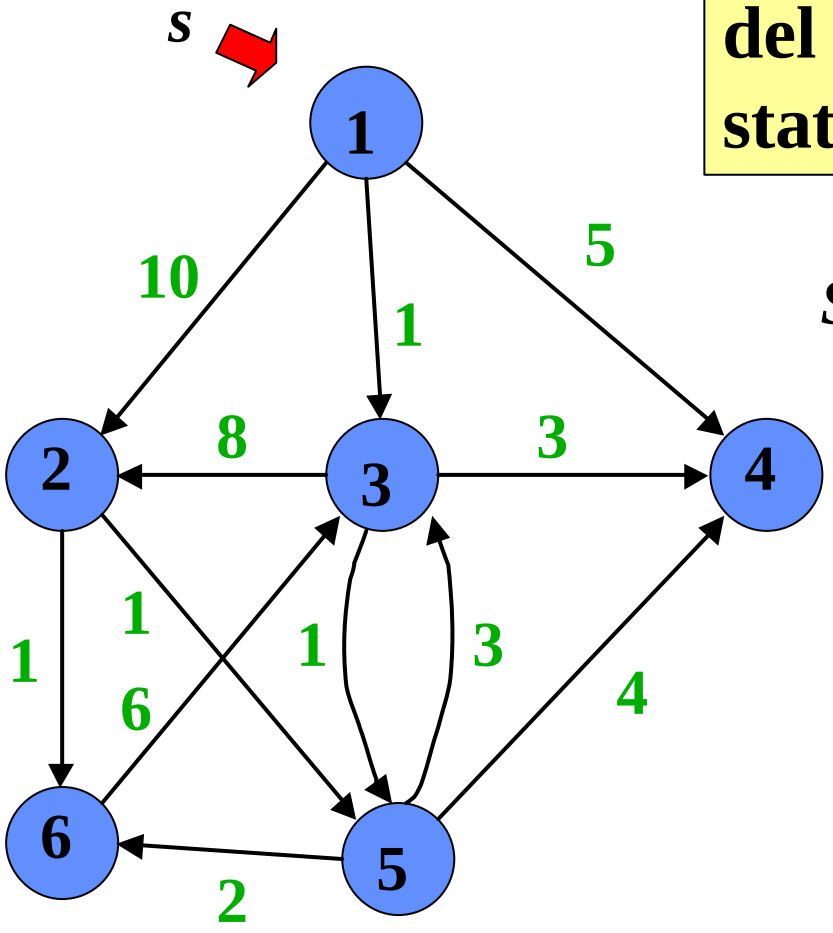
(qui manteniamo la *stima* con il *vertice* in *S*)

v	2	3	4	5	6
d	¥	¥	¥	¥	¥

. . . rimuovi *u* dalla *coda*, inserisci *u* in *S*,. . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

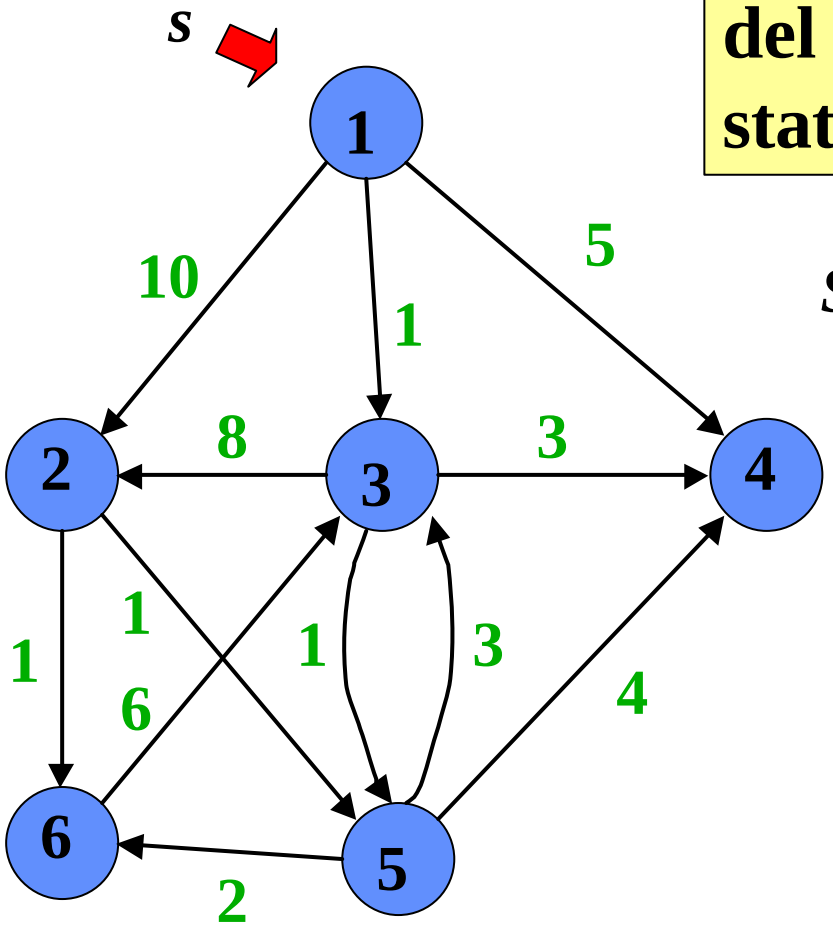
1
0

v	2	3	4	5	6
d	10	1	5	¥	¥

. . . e *rilassa* gli *archi uscenti* da *u*.

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

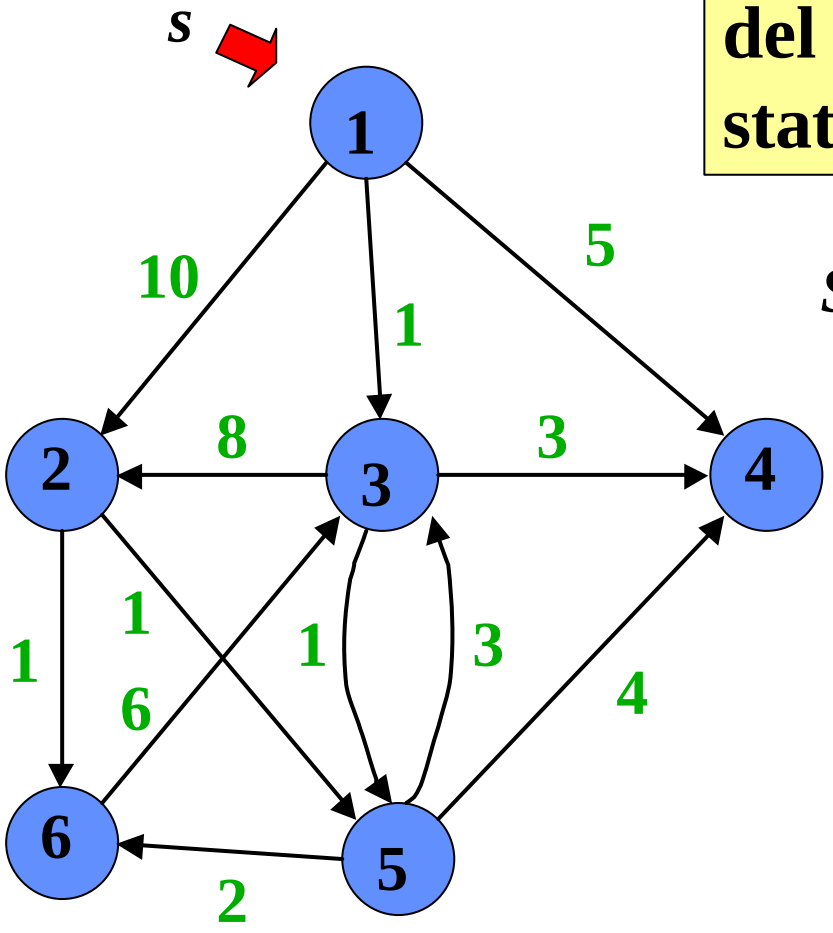
1
0

v	3	4	2	5	6
d	1	5	10	¥	¥

. . . *riordina* la *coda* rispetto alle nuove *stime*.

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

***S* =**

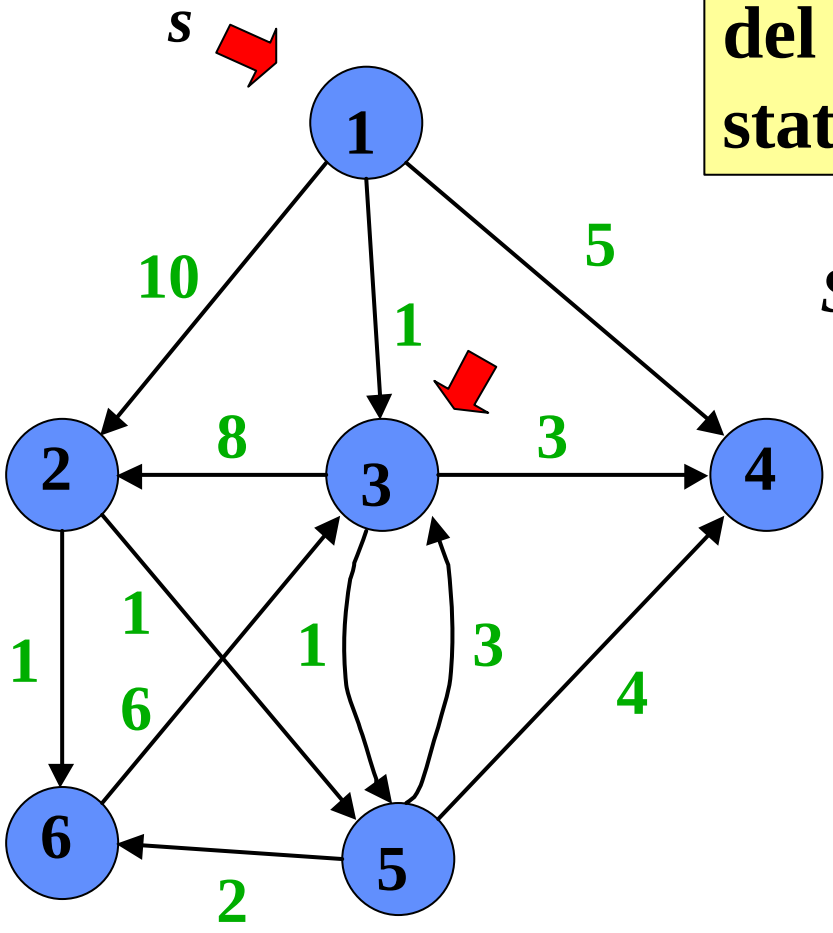
1
0

v	3	4	2	5	6
d	1	5	10	¥	¥

***Ripeti* . . .**

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

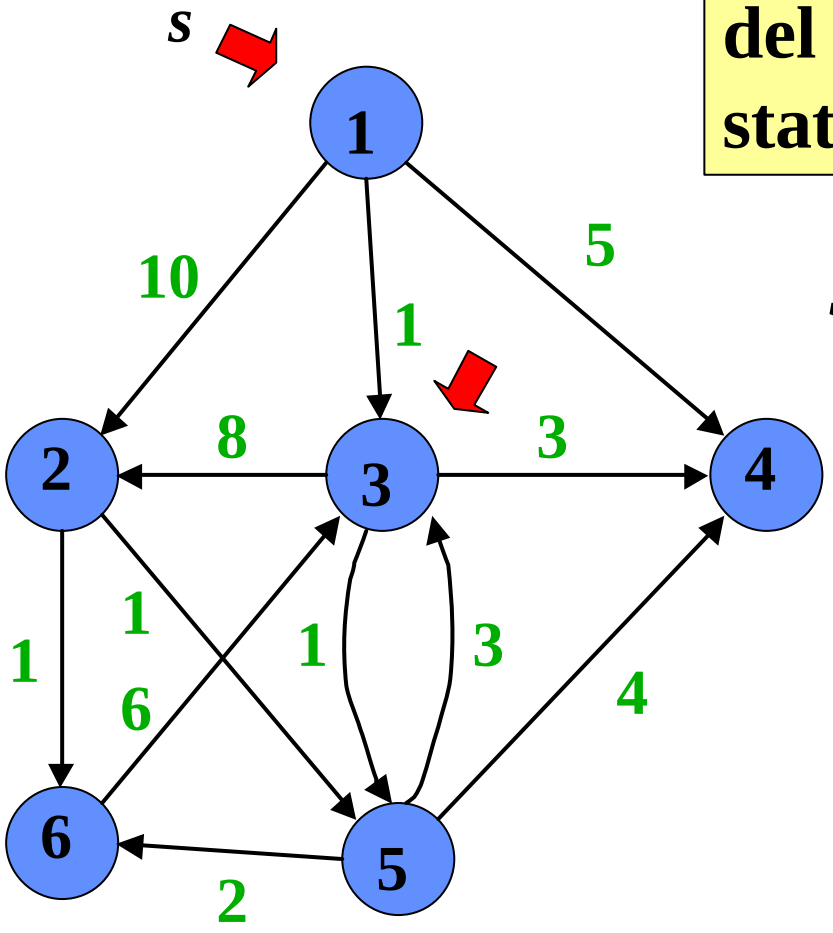
1	3
0	1

v	4	2	5	6
d	5	10	¥	¥

. . . rimuovi dalla *coda* e inserisci in *S*

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



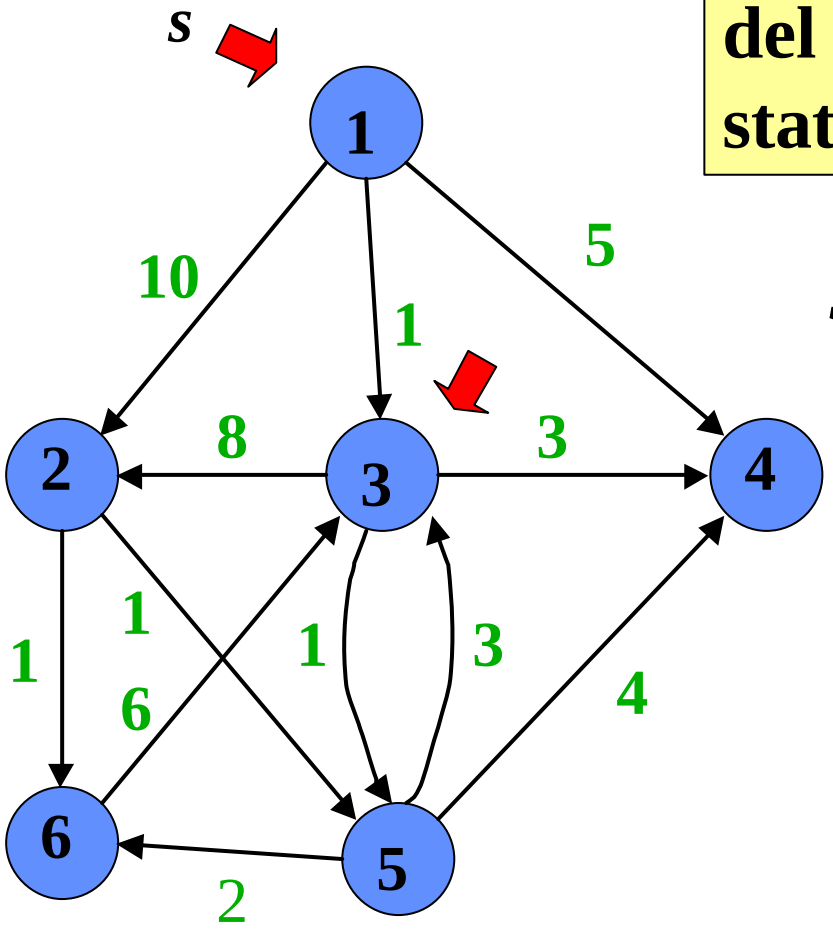
L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =	1	3		
	0	1		
v	4	2	5	6
d	4	9	2	¥

. . . rilassa gli archi . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



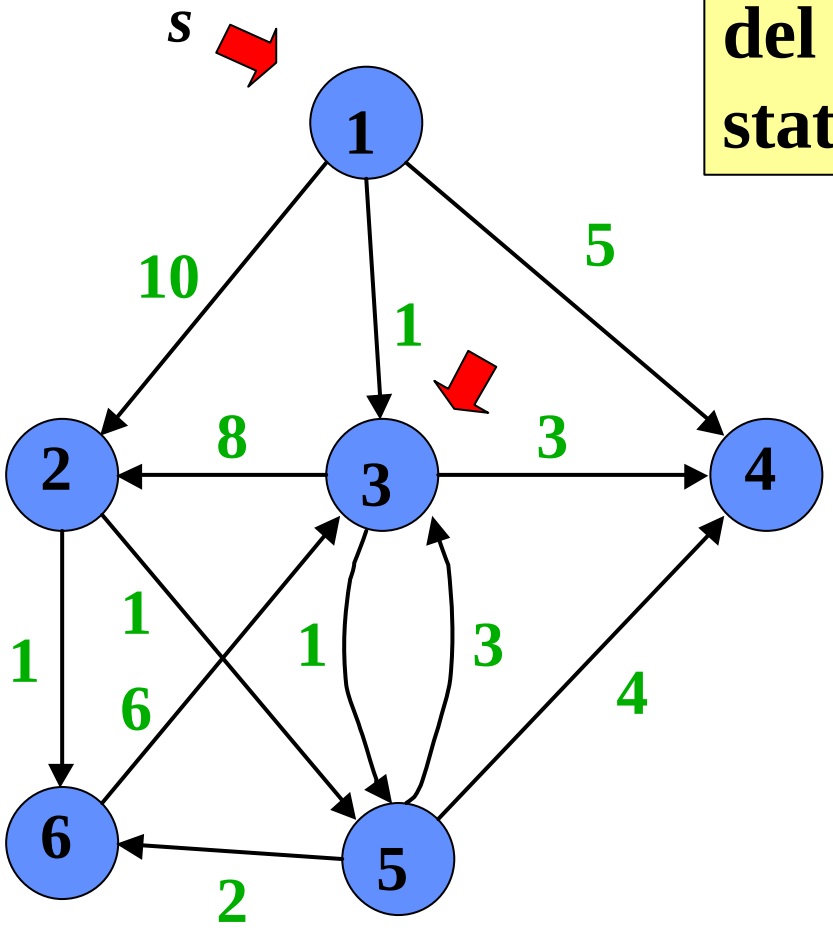
L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =	1	3		
	0	1		
v	5	4	2	6
d	2	4	9	¥

. . . e riordina la coda. . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



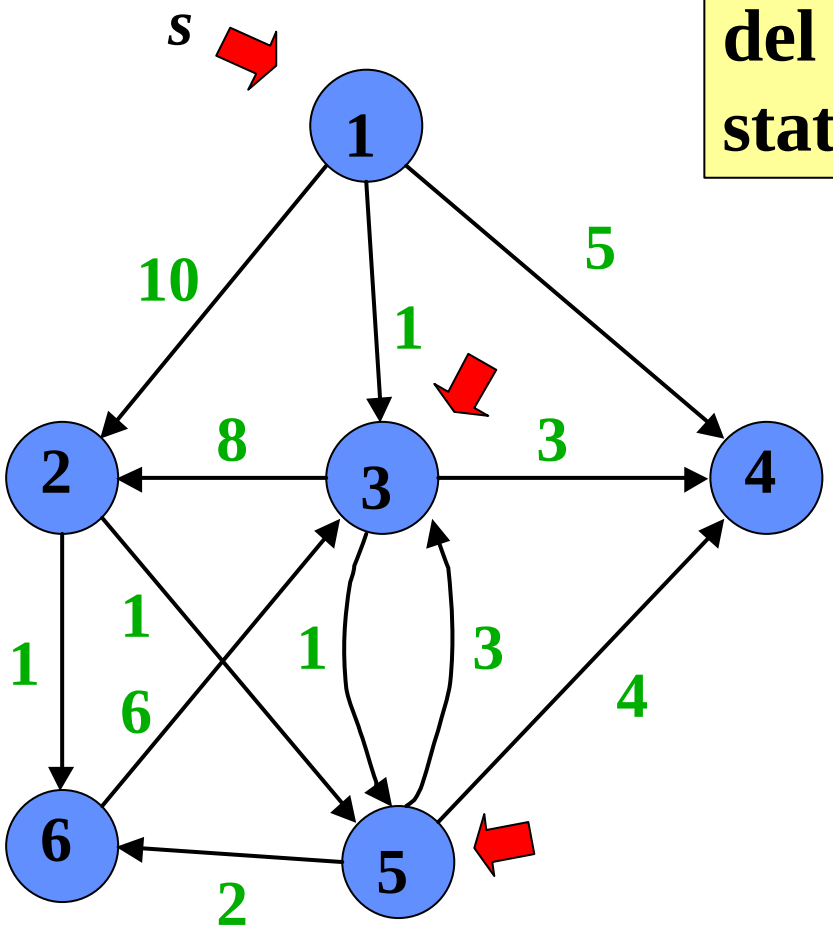
L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =	1	3		
	0	1		
v	5	4	2	6
d	2	4	9	¥

Ripeti . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

1	3	5
0	1	2

v

4	2	6

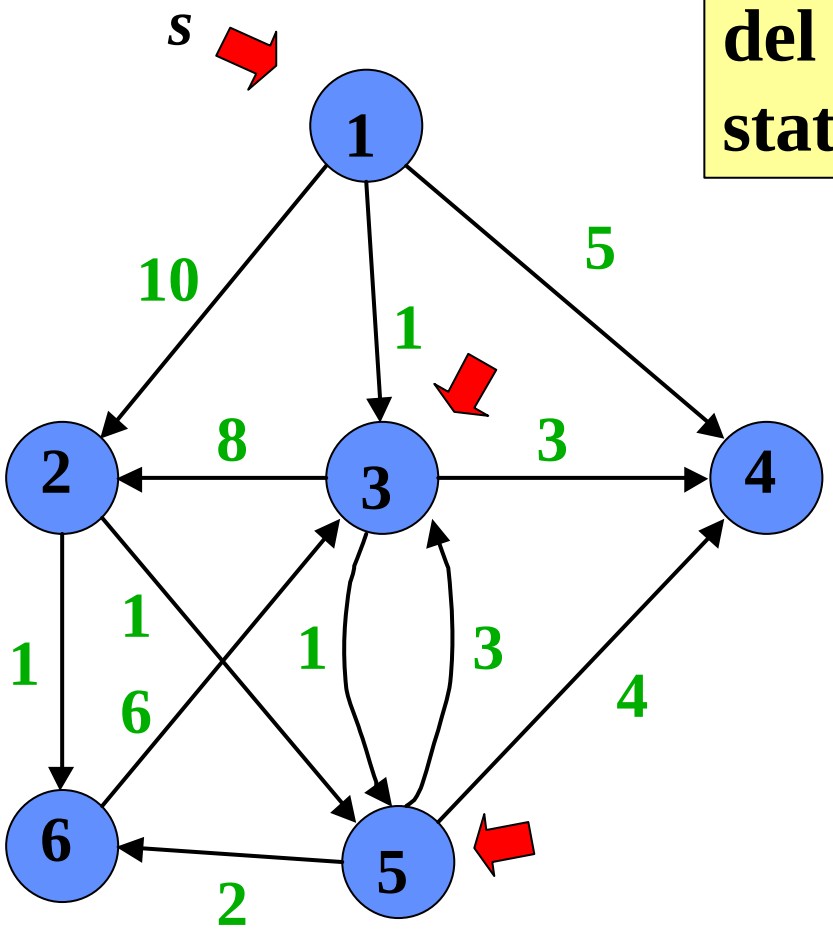
d

4	9	¥

. . . rimuovi dalla coda e inserisci in *S*

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



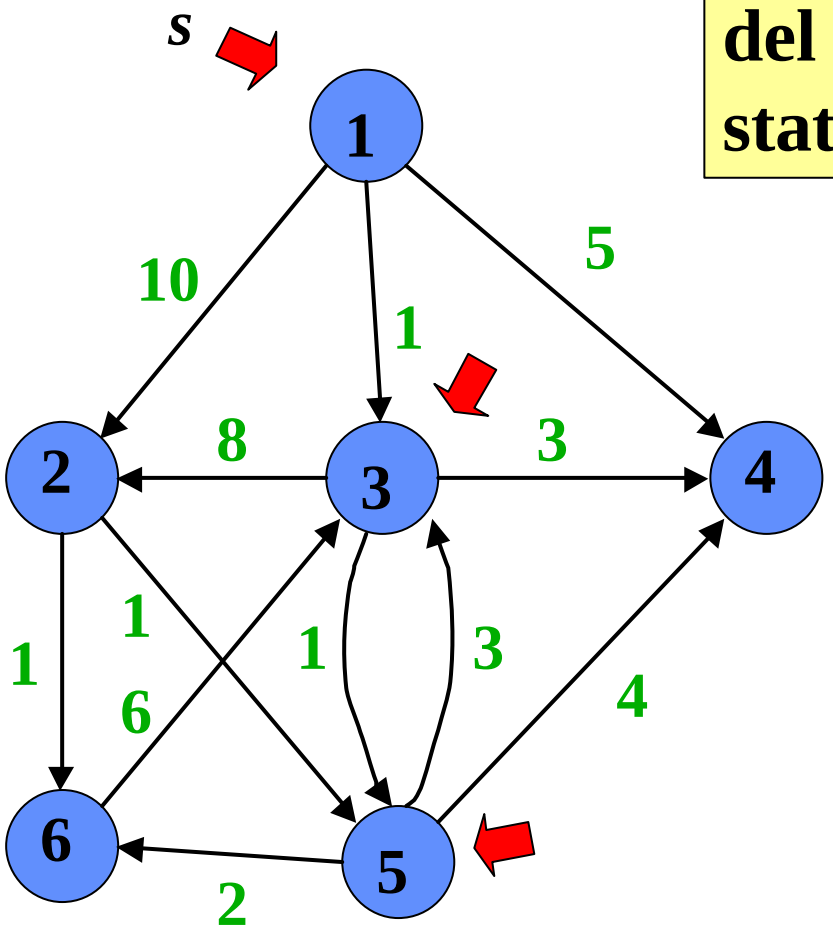
L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =	1	3	5		
	0	1	2		
v	4	2	6		
d	4	9	4		

. . . rilassa gli archi . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



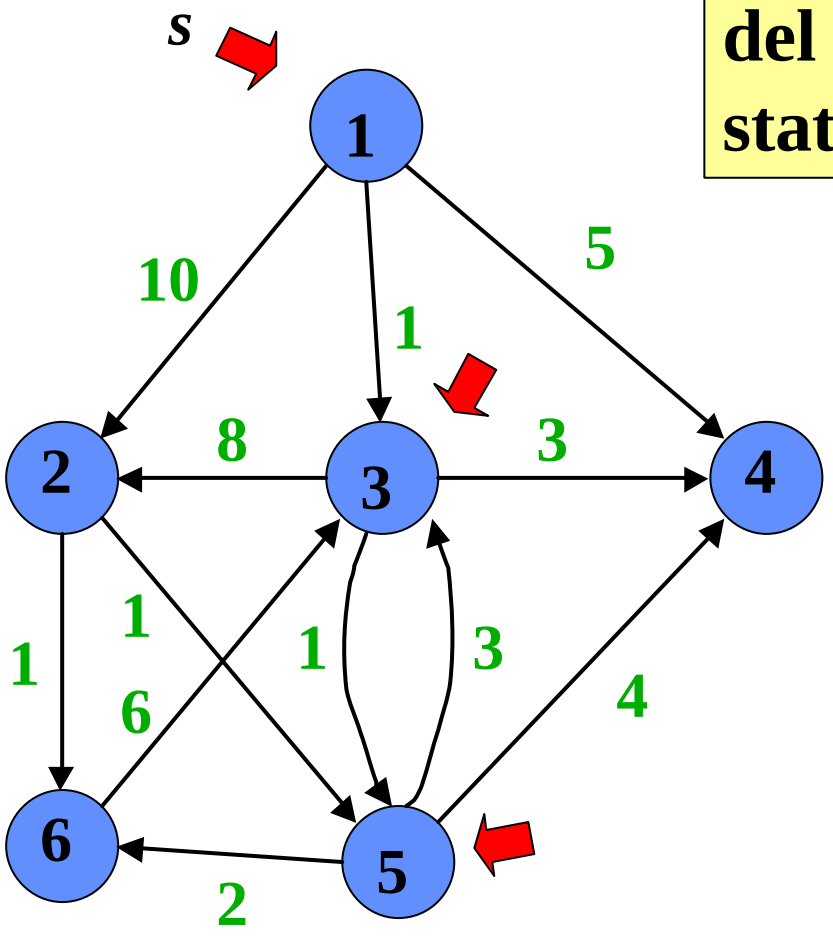
L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =	1	3	5
	0	1	2
v	4	6	2
d	4	4	9

. . . e riordina la coda. . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



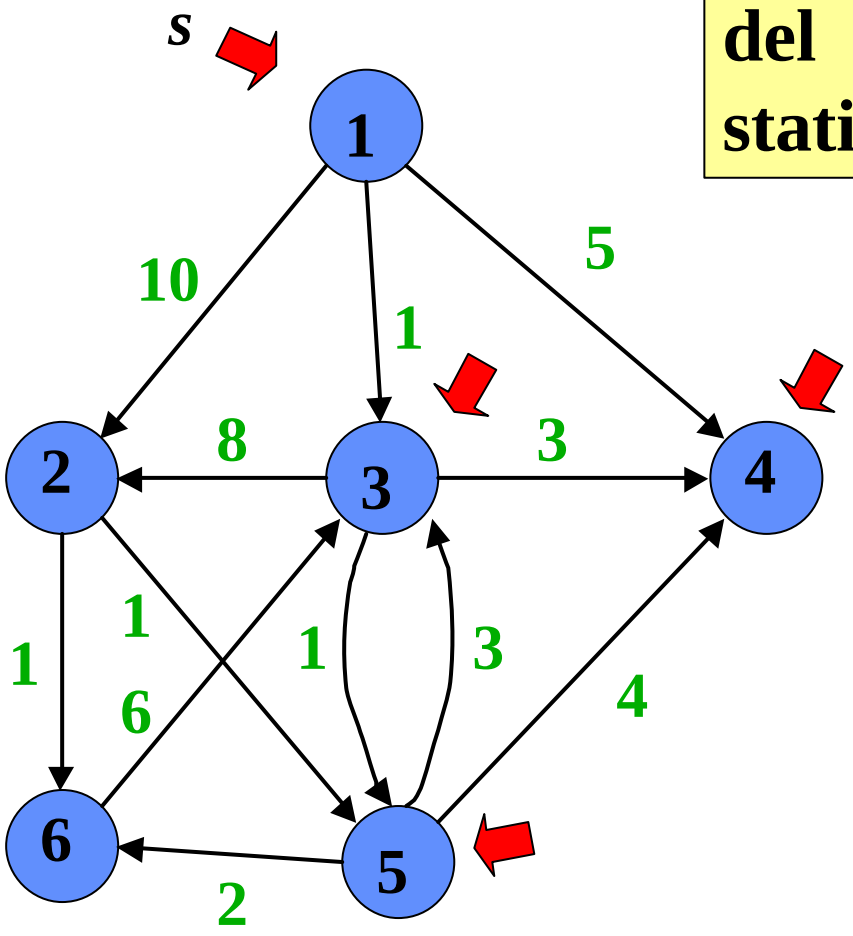
L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =	1	3	5
	0	1	2
v	4	6	2
d	4	4	9

Ripeti . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

1	3	4	5
0	1	4	2

v

6	2

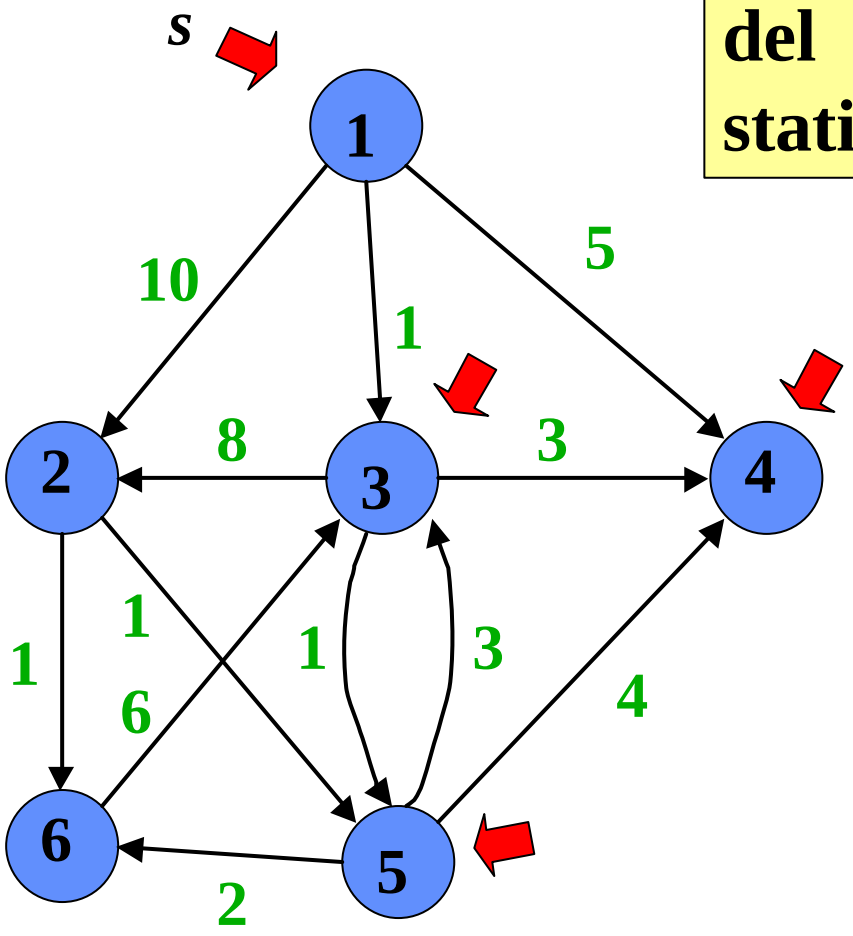
d

4	9

. . . rimuovi dalla coda e inserisci in *S*

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

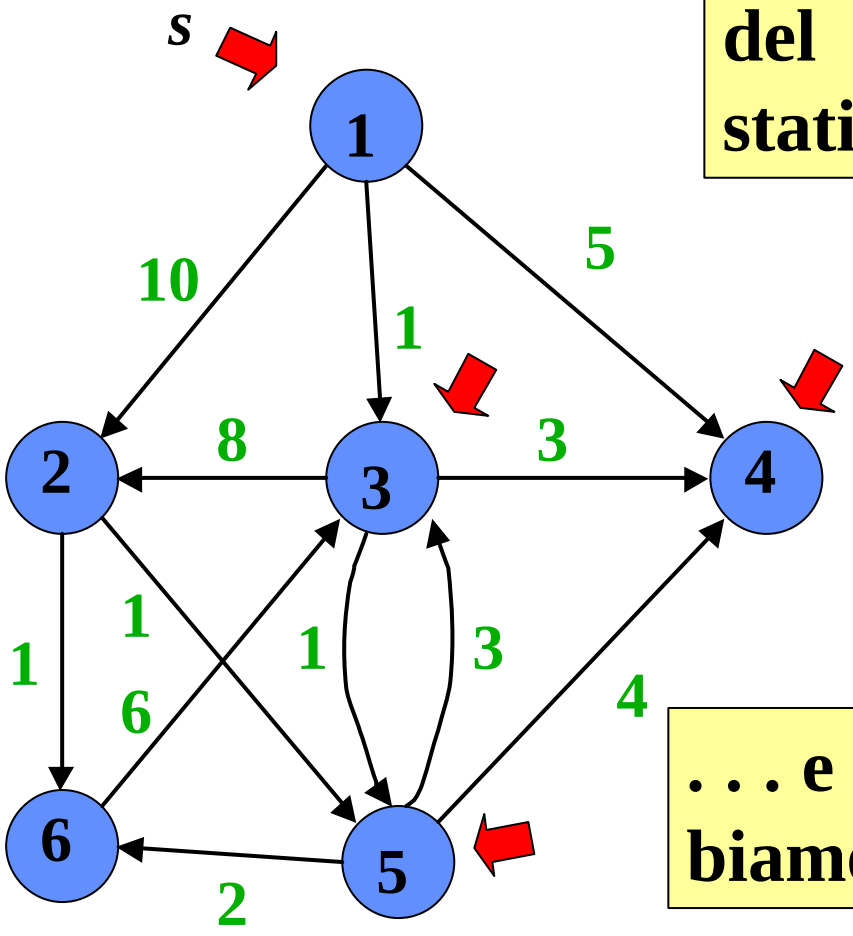
S =	1	3	4	5
	0	1	4	2

v	6	2
d	4	9

. . . rilassa gli archi . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

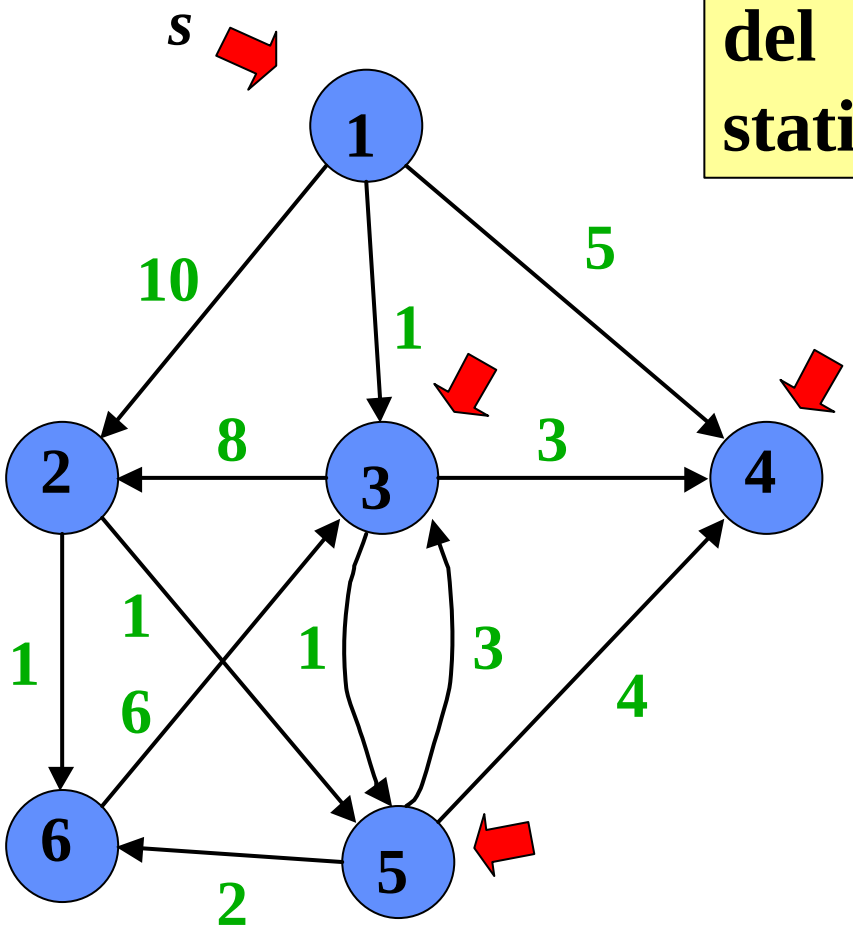
S =	1	3	4	5
	0	1	4	2

v	6	2
d	4	9

. . . e riordina la coda (nessun cambiamento in questo caso) . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

S =

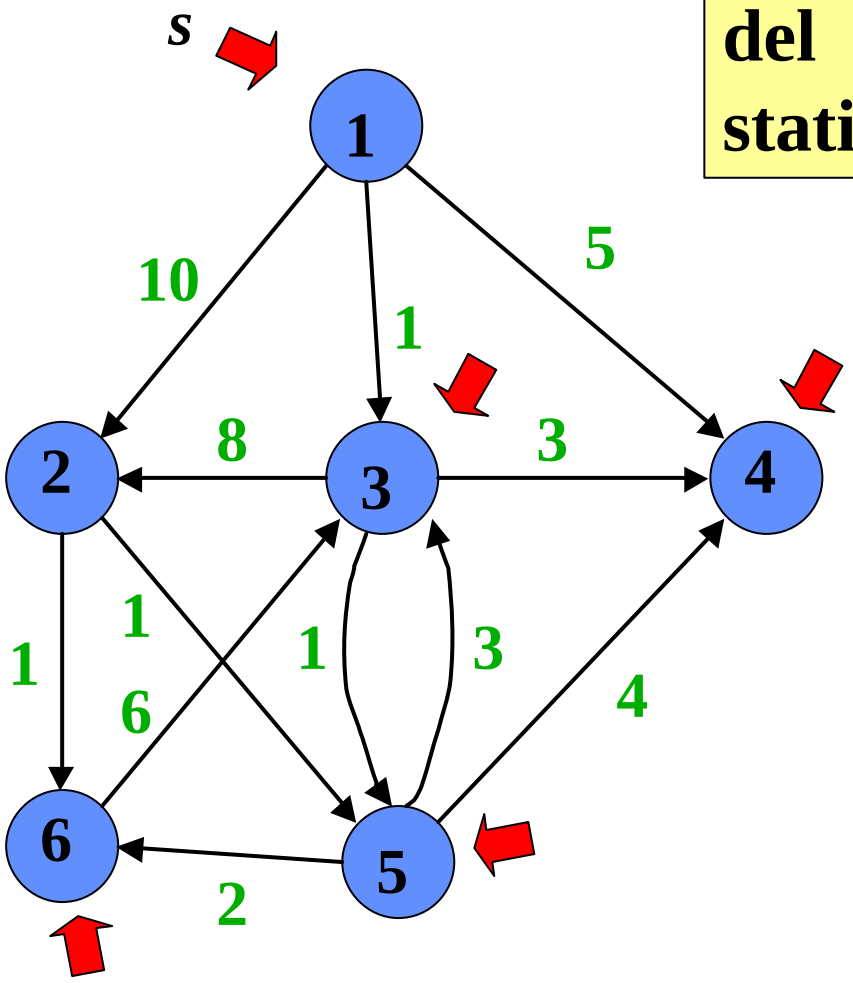
1	3	4	5
0	1	4	2

v	6	2
d	4	9

Ripeti . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

***S* =**

1	3	4	5	6
0	1	4	2	4

v

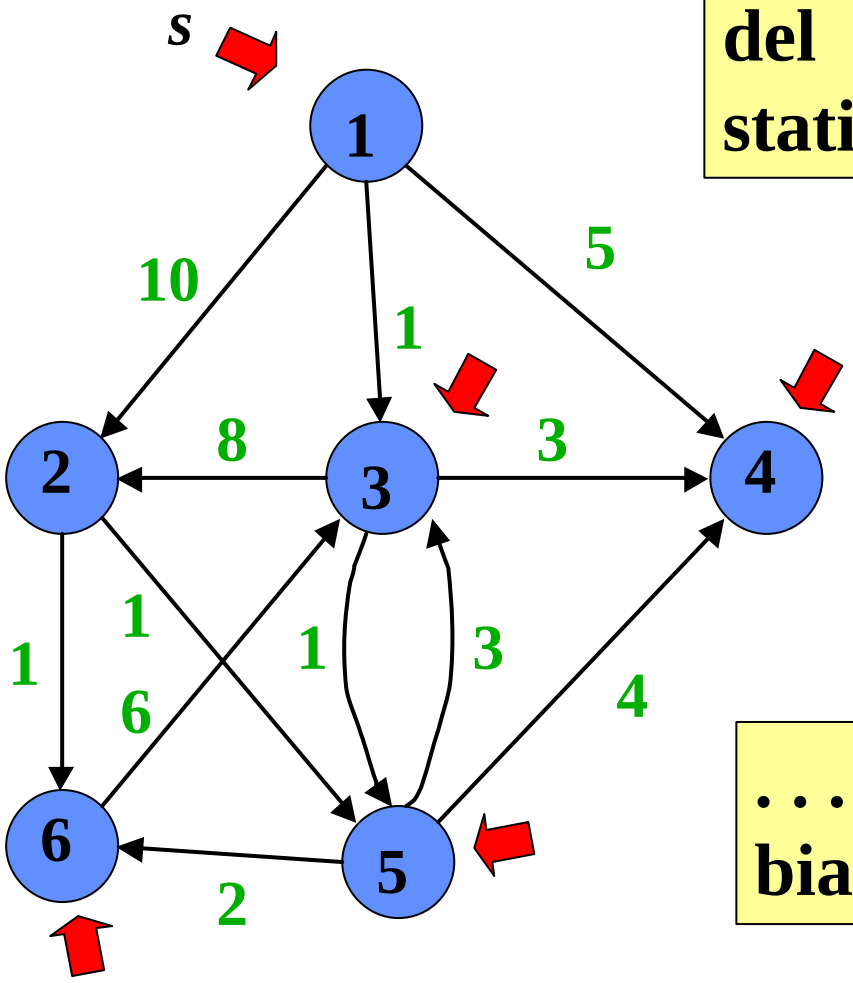
2
9

d

. . . rimuovi dalla coda e inserisci in *S*

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

$S =$

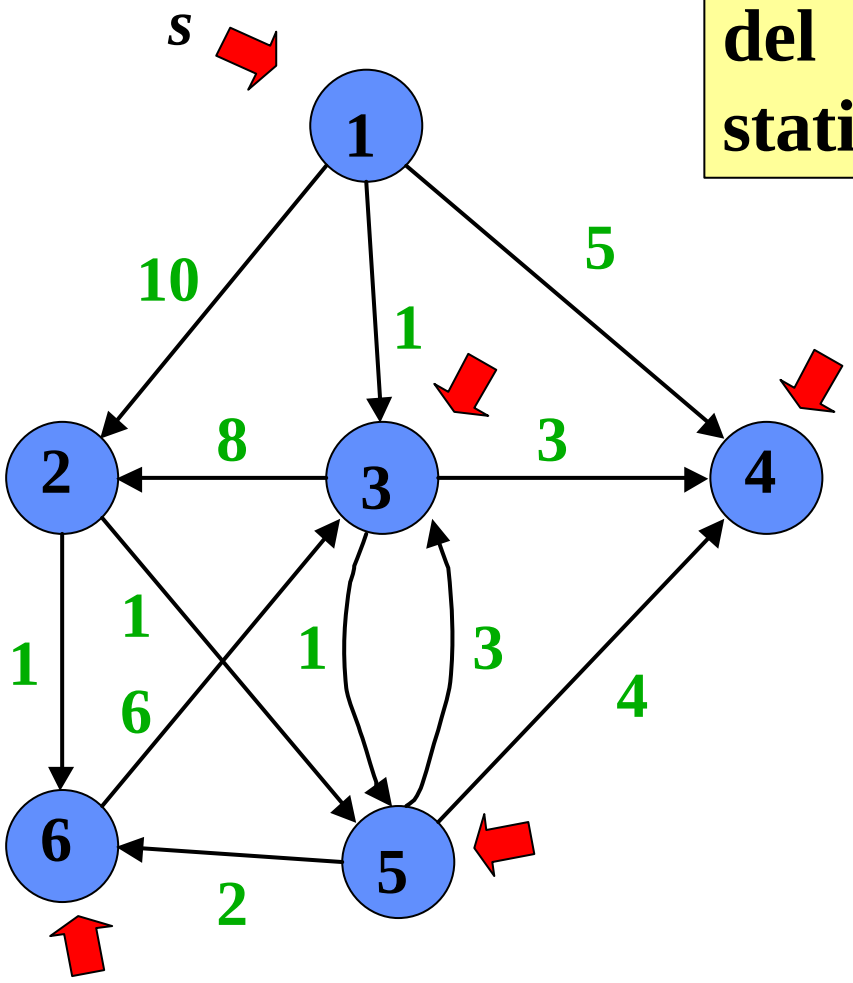
1	3	4	5	6
0	1	4	2	4

v	2
d	9

. . . rilassa gli archi (nessun cambiamento in questo caso) . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

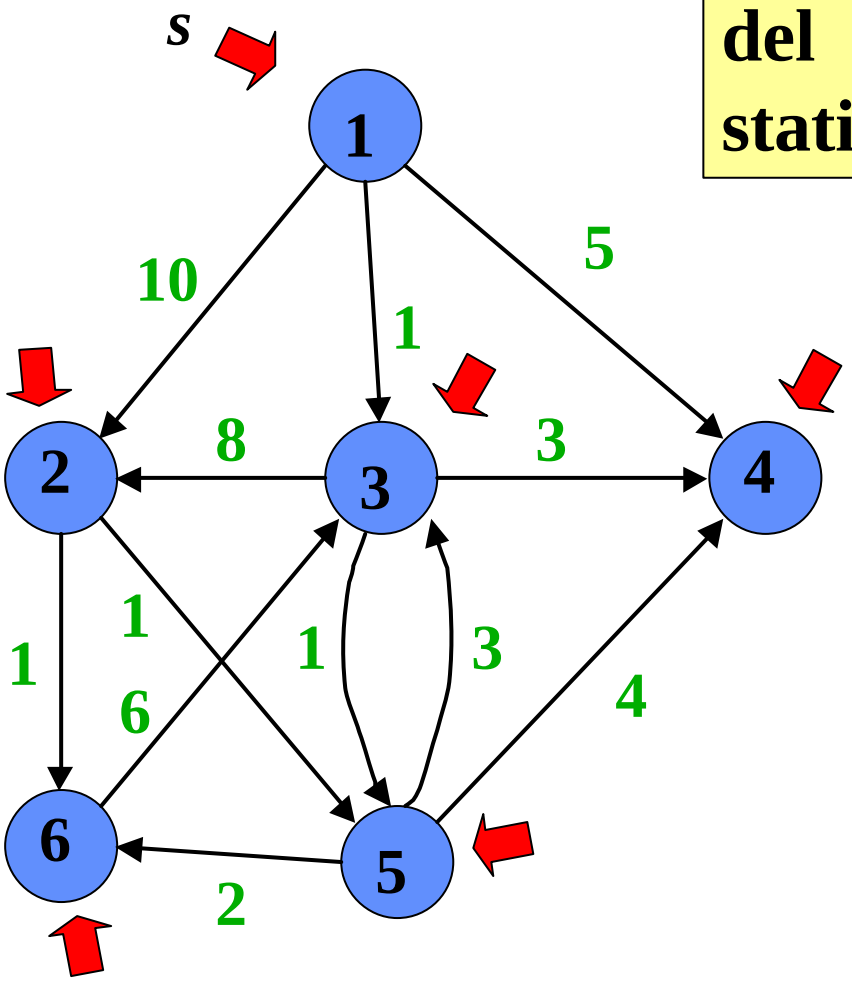
S =	1	3	4	5	6
	0	1	4	2	4

v	2
d	9

Ripeti . . .

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.



L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

***S* =**

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

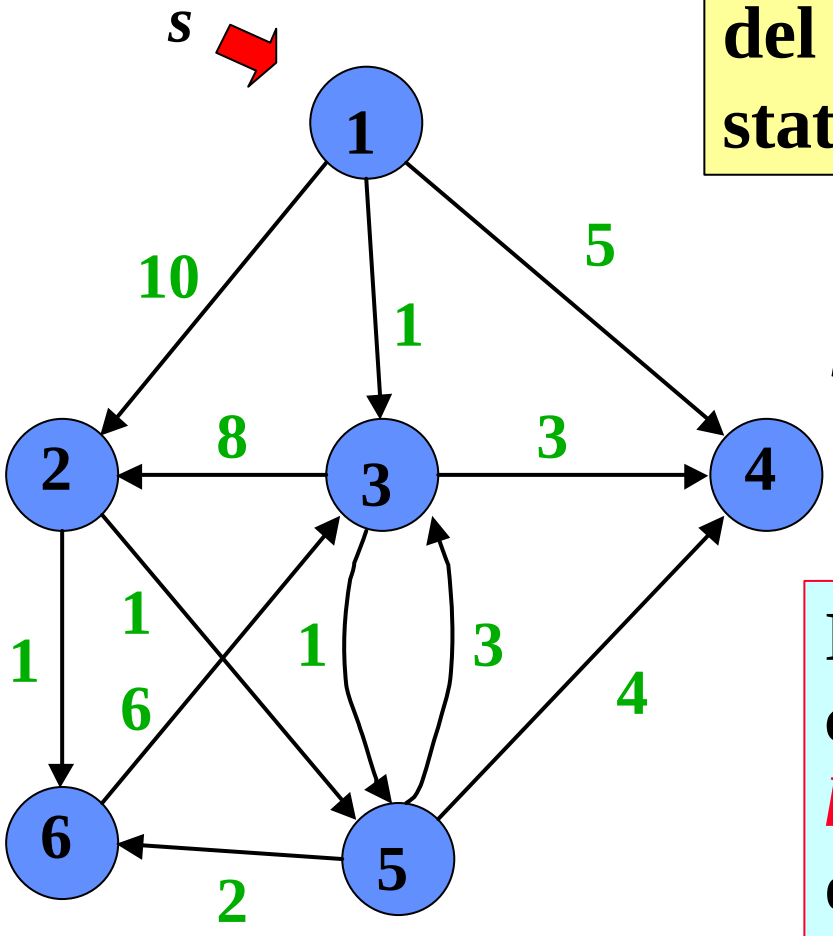
v

d

Fatto!

Grafi: Percorsi minimi (Dijkstra)

Sia 1 il *vertice sorgente*.

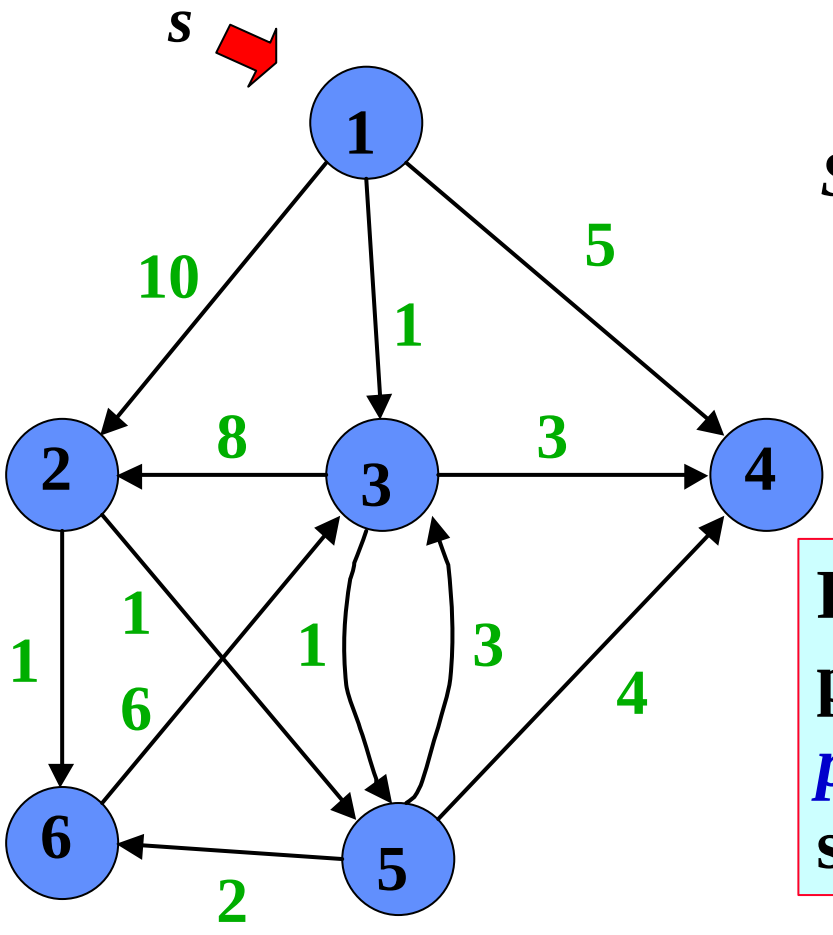


L'*algoritmo di Dijkstra* utilizza un insieme *S* di vertici i cui pesi del percorso minimo sono già stati determinati.

v	1	2	3	4	5	6
d	0	9	1	4	2	4

Il risultato è la riga in basso che contiene in ciascuna cella la *lunghezza del percorso minimo* dal nodo *s* al nodo indicato nella riga sopra.

Grafi: Percorsi minimi (Dijkstra)

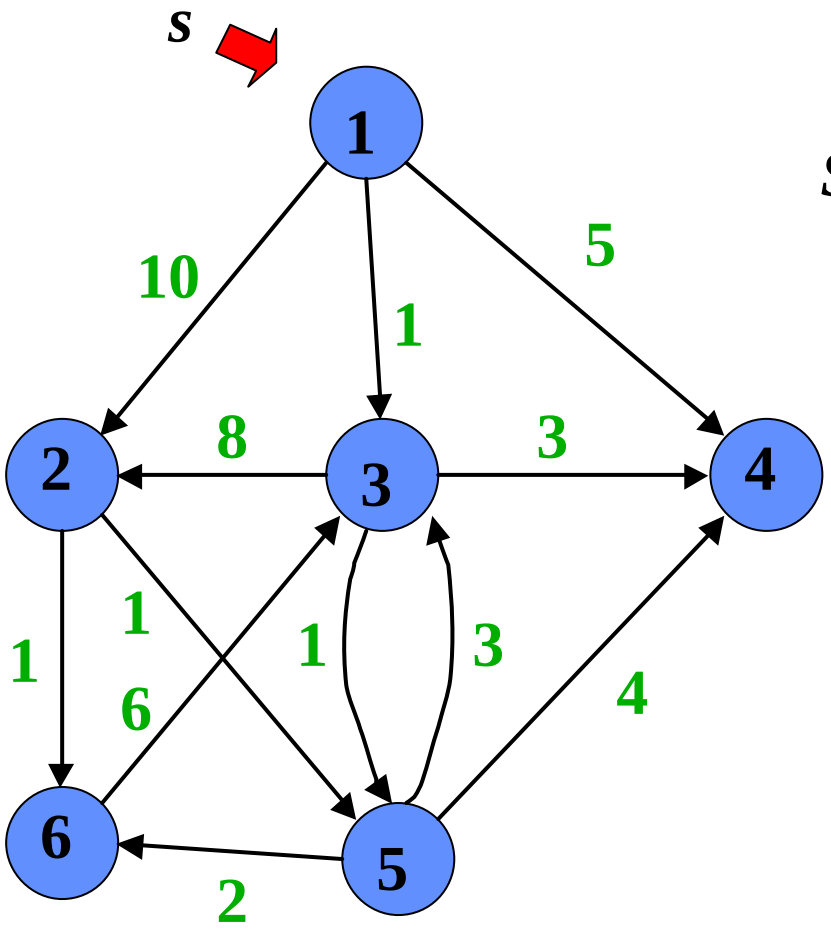


S =

v	1	2	3	4	5	6
d	0	9	1	4	2	4
p		3	1	3	3	5

Per calcolare i percorsi corrispondenti, utilizziamo il campo $p[v]$, che contiene il predecessore v lungo il percorso minimo.

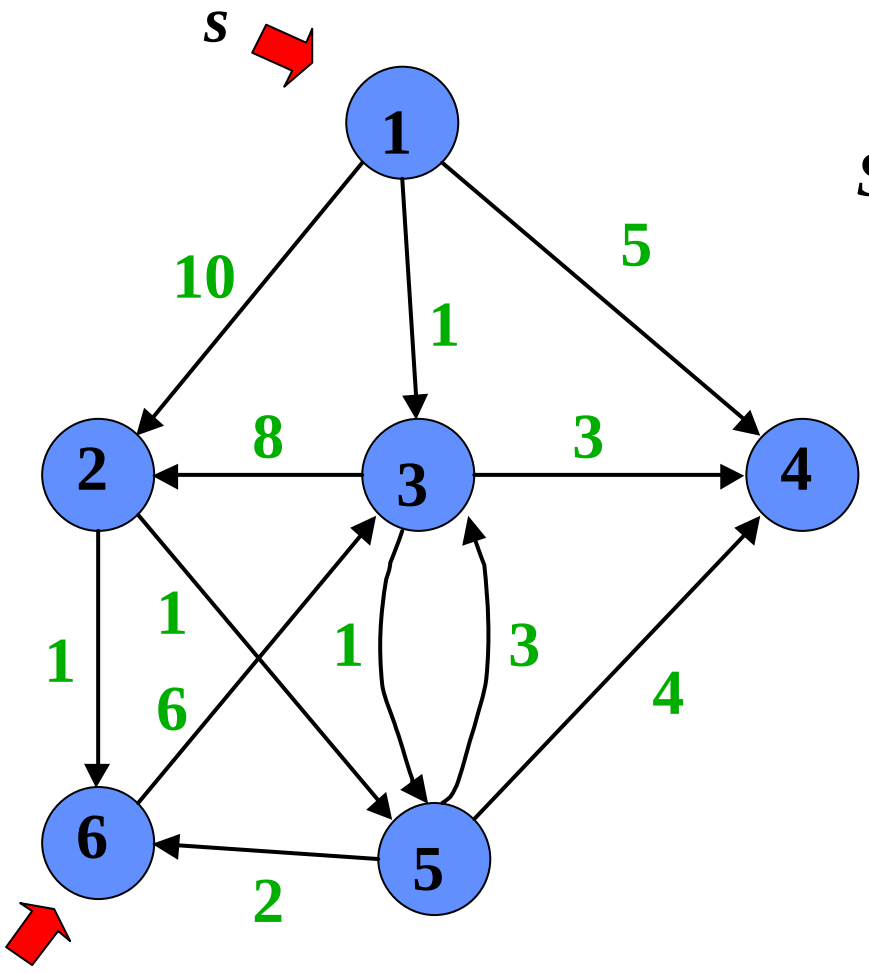
Grafi: Percorsi minimi (Dijkstra)



S = v	1	2	3	4	5	6
d	0	9	1	4	2	4
p		3	1	3	3	5

Utilizzando il predecessore *p*[*v*], si può facilmente ricostruire il percorso fino a *v*, *procedendo all'indietro* da *v* a *s*.

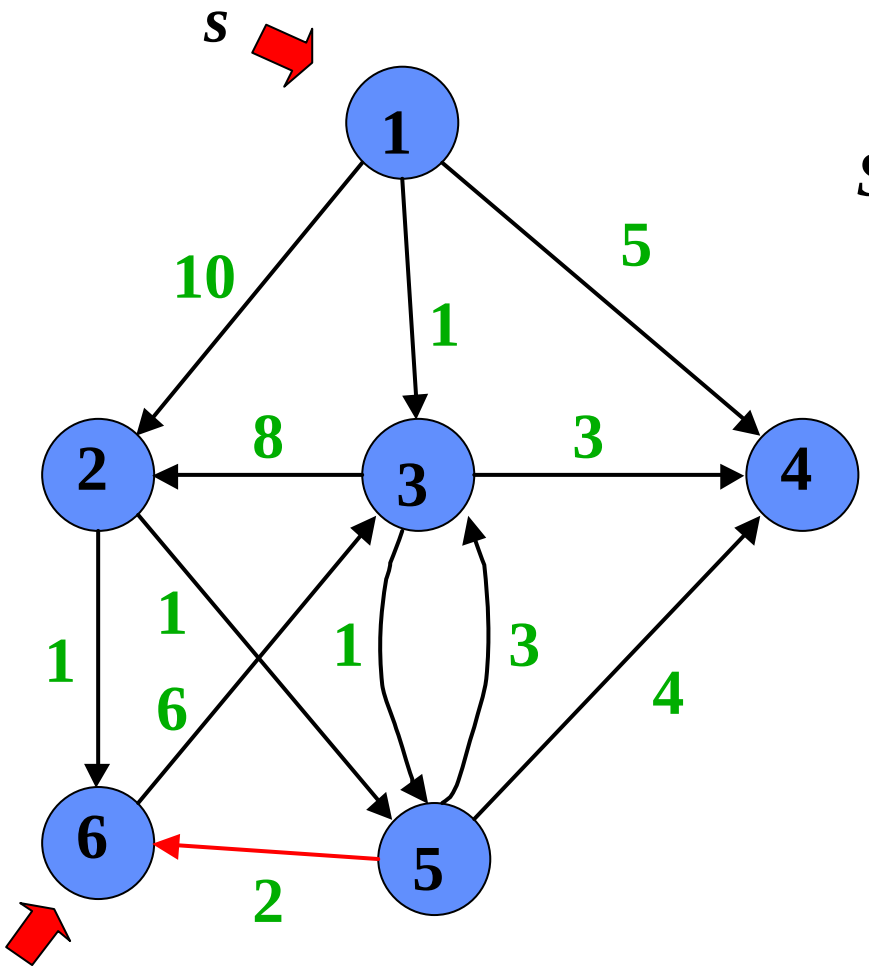
Grafi: Percorsi minimi (Dijkstra)



S = v	1	2	3	4	5	6
d	0	9	1	4	2	4
p		3	1	3	3	5

Es., $v = 6$.

Grafi: Percorsi minimi (Dijkstra)

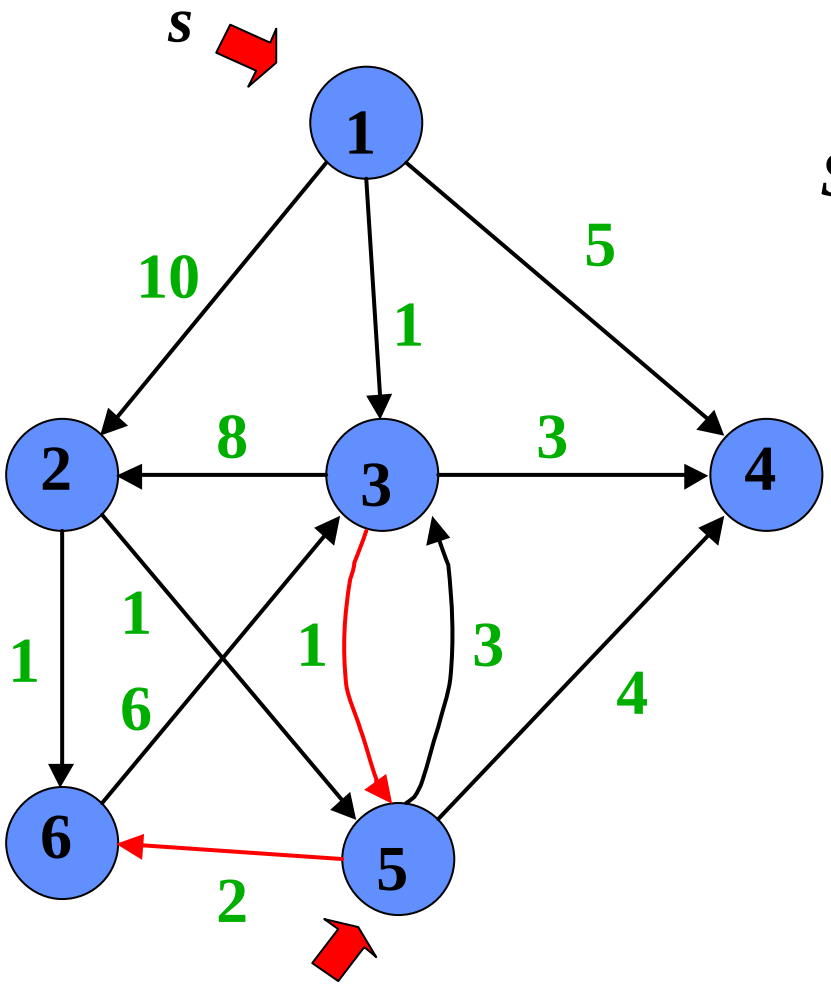


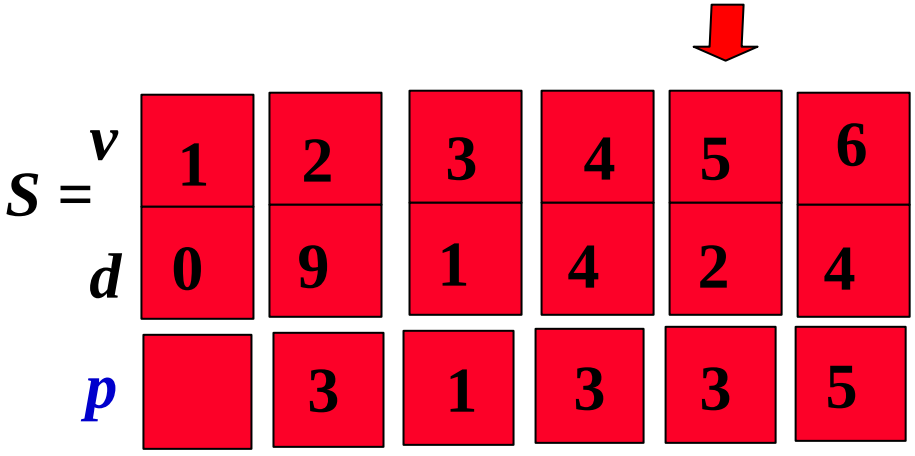
S = v	1	2	3	4	5	6
d	0	9	1	4	2	4
p		3	1	3	3	5

Es., $v = 6$.

$p(6) = 5$,

Grafi: Percorsi minimi (Dijkstra)



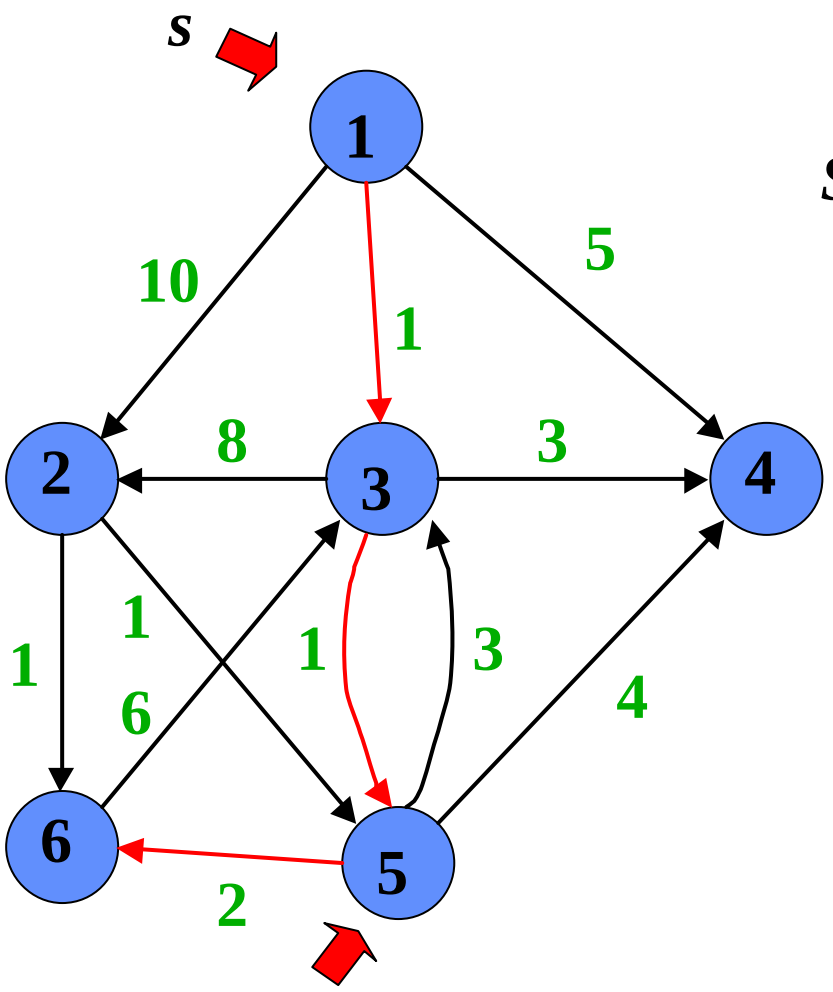


Es., $v = 6$.

$p(6) = 5$,

$p(5) = 3$,

Grafi: Percorsi minimi (Dijkstra)



S = v	1	2	3	4	5	6
d	0	9	1	4	2	4
p		3	1	3	3	5

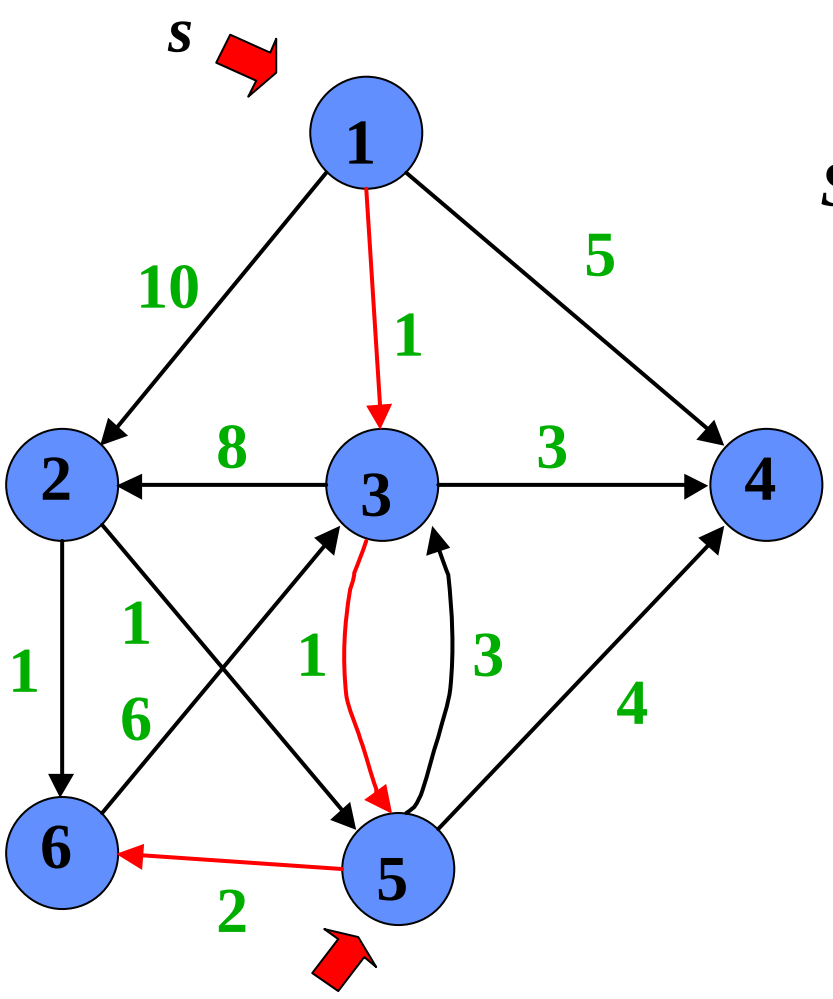
Es., $v = 6$.

$p(6) = 5$,

$p(5) = 3$,

$p(3) = 1$.

Grafi: Percorsi minimi (Dijkstra)



$S =$ v	1	2	3	4	5	6
d	0	9	1	4	2	4
p		3	1	3	3	5

Es., $v = 6$.

$p(6) = 5$,

$p(5) = 3$,

$p(3) = 1$.

Percorso: 1, 3, 5, 6.

Peso: 4.

L'algoritmo di Dijkstra

Coda di priorità

```
Dijkstra(G,s)
    Inizializza(G,s)
    S = Æ
    Q = V(G)
    while ( Q 1 Æ )
        u = Delete_Min(Q)
        S = S È {u}
        for each vertice v adiecente a u
            relax(u,v,w)
```

L'algoritmo di Dijkstra

Coda di priorità

Operazione riduzione del valore di un elemento

```
Dijkstra(G,s)
    Inizializza(G,s,d)
    S = Æ
    Q = V(G)
    while ( Q 1 Æ )
        u = Delete_Min(Q)
        S = S È {u}
        for each vertice v adiecente a u
            relax(u,v,w)
```

```
Relax(u,v,w)
    if d[v] > d[u] + w(u,v)
      then Decrase_key(d[v], d[u]+w(u,v))
           p[v] = u
```

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*. Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.**

- **`Delete_Min` quante volte viene eseguita?**

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*. Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.**

- **`Delete_Min` viene eseguita $O(|V|)$ volte.**

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*. Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.**

- **`Delete_Min` viene eseguita $O(|V|)$ volte.**
- **`Decrease_key` viene eseguita $O(|E|)$ volte.**

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*. Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.**

- **`Delete_Min` viene eseguita $O(|V|)$ volte.**
- **`Decrease_key` viene eseguita $O(|E|)$ volte.**
- ***Tempo totale* = $|V|\, T_{\text{Delete\_min}} + |E|\, T_{\text{Decrease\_key}}$**

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*.**
Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.

- `Delete_Min` **viene eseguita $O(|V|)$ volte.**
- `Decrease_key` **viene eseguita $O(|E|)$ volte.**
- ***Tempo totale*** **= $|V|\, T_{\text{Delete\_min}} + |E|\, T_{\text{Decrease\_key}}$**

Coda a priorità	$T_{\text{Delete\_min}}$	$T_{\text{Decrease\_key}}$	*Tempo Totale*
Array non ordinato			
Heap binario			

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*. Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.**

- **`Delete_Min` viene eseguita $O(|V|)$ volte.**
- **`Decrease_key` viene eseguita $O(|E|)$ volte.**
- ***Tempo totale* = $|V|\ T_{\text{Delete\_min}} + |E|\ T_{\text{Decrease\_key}}$**

Coda a priorità	$T_{\text{Delete\_min}}$	$T_{\text{Decrease\_key}}$	*Tempo Totale time*
Array non ordinato	$O(\|V\|)$	$O(1)$	
Heap binario			

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*. Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.**

- **`Delete_Min` viene eseguita $O(|V|)$ volte.**
- **`Decrease_key` viene eseguita $O(|E|)$ volte.**
- ***Tempo totale* = $|V|\ T_{\text{Delete\_min}} + |E|\ T_{\text{Decrease\_key}}$**

Coda a priorità	$T_{\text{Delete\_min}}$	$T_{\text{Decrease\_key}}$	*Tempo Totale*
Array non ordinato	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Heap binario	$O(\log \|V\|)$	$O(\log \|V\|)$	

Tempo di esecuzione: Dijkstra

***Tempo di esecuzione*: verifichiamo il tempo in relazione a *differenti implementazioni* della *coda di priorità*.**
Differenti implementazioni danno *differenti costi* per le *operazioni sulla coda*.

- `Delete_Min` **viene eseguita $O(|V|)$ volte.**
- `Decrease_key` **viene eseguita $O(|E|)$ volte.**
- ***Tempo totale*** **$= |V| \, T_{\text{Delete\_min}} + |E| \, T_{\text{Decrease\_key}}$**

Coda a priorità	$T_{\text{Delete\_min}}$	$T_{\text{Decrease\_key}}$	*Tempo Totale*
Array non ordinato	$O(\|V\|)$	$O(1)$	$O(\|V\|^2)$
Heap binario	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(E \log \|V\|)$

Coda di priorità: decrease_key

```
DECREASE-KEY(A,i,key)
1.  if key > A[i] then
2.     error “key > A[i]"
3.  A[i] = key
4.  while i>1 and A[PARENT(i)]>A[i] do
5.     scambia A[i] con A[PARENT(i)]
6.     i = PARENT(i)
```

Algoritmo di Dijkstra: correttezza

Teorema: **Se eseguiamo l'*algoritmo di Dijkstra* su un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali *non-negativi*, e un vertice sorgente s, allora, alla terminazione, $d[u] = \delta(s,u)$ per tutti i vertici u in V.**

Rilassamento: proprietà

Lemma 4: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Sia s la sorgente e il grafo sia inizializzato con una chiamata a `Inizializza(G,s)`. Allora, vale $d[v] \geq \delta(s,v)$ per ogni vertice v di G e tale *invariante* viene mantenuto lungo ogni sequenza di operazioni di *rilassamento*. Inoltre, appena $d[v] = \delta(s,v)$, $d[v]$ *non cambia più*.**

Rilassamento: proprietà

Corollario 2: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Supponiamo che in *G non esistano percorsi tra s* e *un vertice v*. Allora dopo che grafo è stato inizializzato con `Inizializza(G,s)`, vale $d[v] = \delta(s,v)$ e questa *uguaglianza* viene mantenuta *da ogni sequenza* di operazioni di *rilassamento*.**

Rilassamento: proprietà

Lemma 5: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Sia s la sorgente e $s \xrightarrow{p} u \to v$ sia un *percorso minimo* per qualche $u,v \in V$. Il grafo sia inizializzato con una chiamata a `Inizializza(G,s)` e venga applicata una sequenza di operazioni di *rilassamento* che includa `Relax(u,v,w)`.**

Se, $d[u] = \delta(s,u)$ in qualunque momento *prima della chiamata*, allora vale $d[v] = \delta(s,v)$ *sempre dopo la chiamata*.

Algoritmo di Dijkstra: correttezza

Dimostriamo che per ogni $u \hat{I} V$, quando u viene inserito in S, vale $d[u] = d(s,u)$.

Procediamo per contraddizione. Sia u il primo nodo per cui $d[u] \neq d(s,u)$ quando viene inserito in S.

Consideriamo la situazione all'inizio del *while* quando u viene inserito in S e otteniamo la contraddizione $d[u] = d(s,u)$, esaminando il *percorso minimo* da s a u.

Sappiamo che:

1 $u \neq s$ poiché $d[s] = 0 = d(s,s)$ all'inizio del *while*.

2 Ma allora deve valere anche $S \neq Æ$ prima che u sia inserito.

3 Ci deve essere un percorso da s a u altrimenti $d[u] = ¥ = d(s,u)$ per il *Corollario 2*, e quindi contraddizione.

4 Se c'è un percorso, c'è anche un *percorso minimo p*.

Algoritmo di Dijkstra: correttezza

Dimostrazione: **Sia *u* il primo nodo per cui *d*[*u*] ¹ d(*s*,*u*) quando viene inserito in *S*.**

Sappiamo che:

1 ***u*** **¹** ***s***

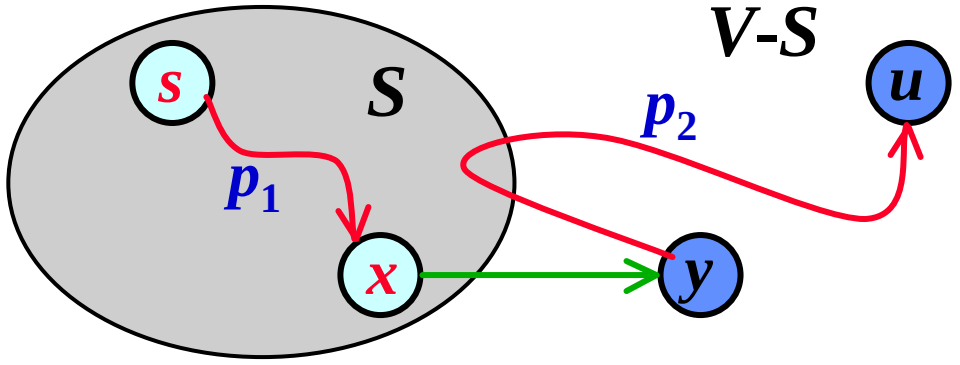
2 ***S*** **¹ Æ**

3 C'è un percorso da *s* a *u*.

4 C'è un *percorso minimo p*.

p **connette un nodo in *S* con un nodo (*u*) in *V-S***

Sia *y* il primo vertice in *V-S* lungo *p* e *x* il suo predecessore. *p* è scomponibile in $s \xrightarrow{p_1} x \to y \xrightarrow{p_2} u$



Algoritmo di Dijkstra: correttezza

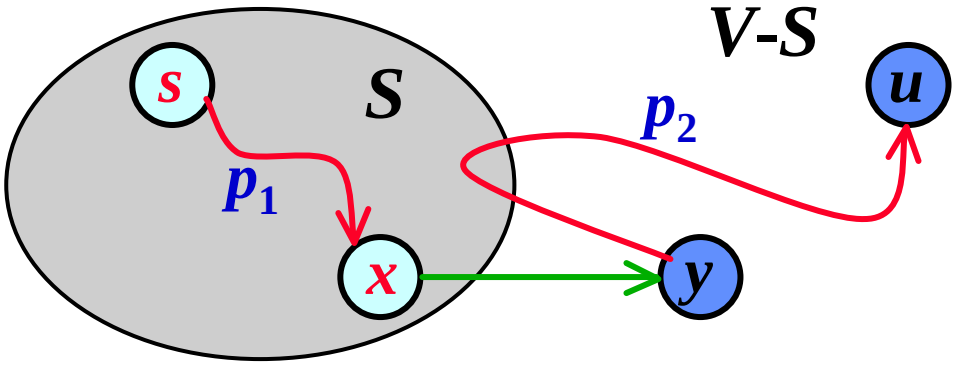
Dimostrazione: **Possiamo asserire che quando *u* è inserito in *S*, vale *d*[*y*] = d(*s*,*y*).**

Infatti sappiamo che *u* è il primo nodo per cui *d*[*u*] $\geq$ d(*s*,*u*), quando viene inserito in *S*.

x **appartiene ad *S*, quindi avevamo *d*[*x*] = d(*s*,*x*) quando è stato inserito in *S*.**

Ma l'algoritmo allora rilassa l'arco *(x,y)*.

Quindi per il *Lemma 5*, deve essere *d*[*y*] = d(*s*,*y*), dopo la chiamata a *Relax(x,y,w)*.



Algoritmo di Dijkstra: correttezza

Dimostrazione: **Possiamo ora ottenere la nostra contraddizione.**

Poiché *y* compare nel *percorso minimo* tra *s* e *u* e i *pesi* (ed in particolare quelli in p_2) sono tutti *non-negativi*,

$$d(s,y) \leq d(s,u)$$

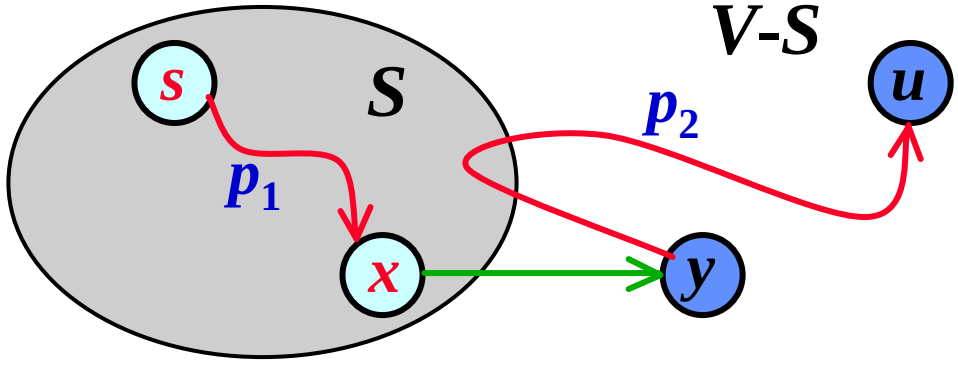
e inoltre $d[y] = d(s,y) \leq d(s,u) \leq d[u]$ **(*Lemma 4*)**

Poiché sia *u* che *y* sono in *V-S* quando *u* viene estratto dalla coda (*linea 5*), vale anche $d[u] \leq d[y]$.

Cioè $d[y] = d[u]$, e quindi

$$d[y] = d(s,y) = d(s,u) = d[u].$$ **(*contraddizione*!)**

Lemma 4 **ci garantisce inoltre che l'uguaglianza vale sempre dopo l'inserimento di *u* in *S*.**



Algoritmo di Dijkstra: correttezza 2

Corollario: **Se eseguiamo l'*algoritmo di Dijkstra* su un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali *non-negativi*, e un vertice sorgente s, allora, alla terminazione il *sottografo dei predecessori* G_p corrisponde all'*albero dei percorsi minimi* con *radice* s.**

Rilassamento e percorsi minimi

Lemma 6: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ e sia s la sorgente.**

Allora, dopo che il grafo è stato inizializzato con una chiamata a `Inizializza(G,s)`, il *sottografo dei predecessori* G_p forma un *albero con radice* s, e qualunque sequenza di operazioni di *rilassamento* su G mantiene questa proprietà.

Rilassamento e percorsi minimi

Dimostrazione: **Per dimostrare il *lemma* è necessario dimostrare che:**

- **sempre G_p è un *grafo aciclico*;**
- **sempre *esiste almeno un percorso* da s a $v \hat{I} V_p$;**
- **sempre *esiste al più un percorso* da s a $v \hat{I} V_p$.**

***Dimostreremo solo la prima*, per la dimostrazione delle altre due vedere Cormen**

Rilassamento e percorsi minimi

Dimostrazione: **Dimostriamo che G_p è sempre un albero (cioè G_p è un *grafo aciclico*).**

Appena eseguita l'inizializzazione la proprietà è ovviamente vera (unico nodo in G_p è s).

Sia stata eseguita una sequenza di rilassamenti.

Supponiamo che l'ultimo rilassamento abbia introdotto un ciclo $<v_0,\ldots,v_k>$ (dove $v_k=v_0$).

Possiamo assumere che sia stato il rilassamento dell'arco (v_{k-1},v_k) a creare il ciclo. (*perché*?)

- Tutti i nodi del ciclo sono raggiungibili da s.

Infatti, ogni nodo v_i nel ciclo ha predecessore $p[v_i]$ non-*Nil* quindi questi nodi hanno valore finito di $d[v_i]$ quando $p[v_i]$ viene assegnato.

Per *Lemma 4* ogni v_i ha percorso minimo finito, quindi è raggiungibile da s.

Rilassamento e percorsi minimi

Dimostrazione: **Supponiamo che l'ultimo rilassamento abbia introdotto un ciclo $<v_0,\ldots,v_k>$ (dove $v_k=v_0$).**

Possiamo assumere che sia stato il rilassamento dell'arco (v_{k-1},v_k) ad creare il ciclo. (*perché*?)

- ***Tutti i nodi del ciclo sono raggiungibili da s*.**

Appena prima del rilassamento di (v_{k-1},v_k) abbiamo che $p[v_i]=v_{i-1}$ per $i=1,\ldots,k-1$ e l'ultimo assegnamento di $d[v_i]$ è stato $d[v_i] = d[v_{i-1}] + w(v_{i-1},v_i)$. Se $d[v_{i-1}]$ è cambiato da allora, può solo essere diminuito.

Allora prima del rilassamento certamente vale

$$d[v_i] \geq d[v_{i-1}] + w(v_{i-1},v_i) \text{ per } i=1,\ldots,k-1$$

Poiché $p[v_k]$ viene assegnato rilassando (v_{k-1},v_k), prima del rilassamento deve valere $d[v_k] > d[v_{k-1}] + w(v_{k-1},v_k)$.

Sommando le due disuguaglianze otteniamo

$$\sum_{i=1}^{k} d[v_i] > \sum_{i=1}^{k} [d[v_{i-1}] + w(v_{i-1},v_i)]$$

Rilassamento e percorsi minimi

Dimostrazione: **Supponiamo che l'ultimo rilassamento abbia introdotto un ciclo $\langle v_0,\ldots,v_k\rangle$ (dove $v_k=v_0$).**

Possiamo assumere che sia stato il rilassamento dell'arco (v_{k-1},v_k) ad creare il ciclo. (*perché*?)

- *Tutti i nodi del ciclo sono raggigibili da s*.

Appena prima del rilassamento di (v_{k-1},v_k) abbiamo che

$$d[v_i] \geq d[v_{i-1}] + w(v_{i-1},v_i) \text{ per } i=1,\ldots,k-1$$

$$d[v_k] > d[v_{k-1}] + w(v_{k-1},v_k).$$

Contraddizione!

Sommando le due disuguaglianze otteniamo

$$\sum_{i=1}^{k} d[v_i] > \sum_{i=1}^{k} [d[v_{i-1}] + w(v_{i-1},v_i)]$$

$$\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$$

$$> \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1},v_i)$$

$$\sum_{i=1}^{k} w(v_{i-1},v_i) < 0$$

Perché ogni vertice del ciclo occorre una sola volta in ogni sommatoria

Rilassamento e percorsi minimi

Lemma 7: **Sia dato un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$. Sia s la sorgente e G non contenga cicli di peso negativo.**

Il grafo sia inizializzato con una chiamata a `Inizializza(G,s)` e venga applicata una sequenza di operazioni di *rilassamento* che includa `Relax(u,v,w)` tale che $d[v] = \delta(s,v)$.

Allora il *sottografo dei predecessori* G_p è un *albero di cammini minimi* con radice s.

Rilassamento e percorsi minimi

Dimostrazione: **Per dimostrare il *lemma* è necessario dimostrare che:**

- V_p **contiene solo *vertici raggiungibili* da *s*;**
- G_p **è un *albero* con *radice s*;**
- **i percorsi in G_p sono *percorsi minimi* da *s* a $v \hat{I} V_p$.**

***Dimostreremo solo la terza*, per la dimostrazione delle altre due vedere Cormen**

Rilassamento e percorsi minimi

***Dimostreremo solo la terza*: Sia $p=\langle v_0,\ldots,v_k\rangle$ un percorso in G_p, dove $v_0=s$ e $v_k=v$. Per $i=1,\ldots,k$ abbiamo le seguenti:**

$$d[v_i] = \delta(s,v_i)$$

$$d[v_i] \geq d[v_{i-1}] + w(v_{i-1},v_i)$$

poiché se fosse minore, il predecessore di v_i in G_p non potrebbe certo essere v_{i-1}.

Ma allora possiamo concludere che

$$w(v_{i-1},v_i) \geq \delta(s,v_i) - \delta(s,v_{i-1})$$

Rilassamento e percorsi minimi

***Dimostreremo solo la terza*: Sia $p=\langle v_0,\ldots,v_k\rangle$ un percorso in G_p, dove $v_0=s$ e $v_k=v$.**

Ma allora possiamo concludere che

$$w(v_{i-1},v_i) \leq d(s,v_i) - d(s,v_{i-1})$$

Calcoliamo il peso del percorso p.

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

$$\leq \sum_{i=1\ldots k} d(s,v_i) - d(s,v_{i-1})$$

serie telescopica

$$\leq d(s,v_k) + d(s,v_0)$$

$$\leq d(s,v_k)$$

poiché $d(s,v_0)=0$. Ma $d(s,v_k)$ è un limite per ogni percorso da s a v_k. Quindi $w(p) = d(s,v_k)$.

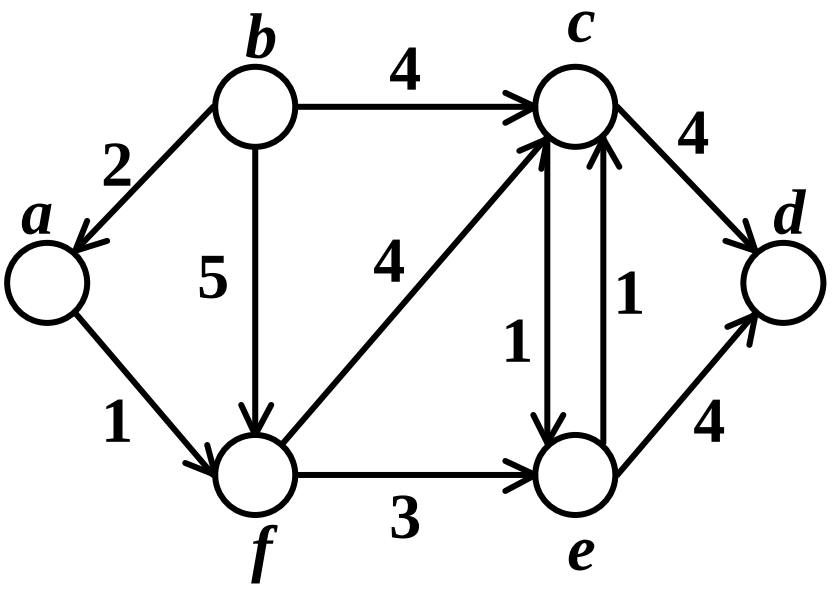
Algoritmo di Dijkstra: correttezza 2

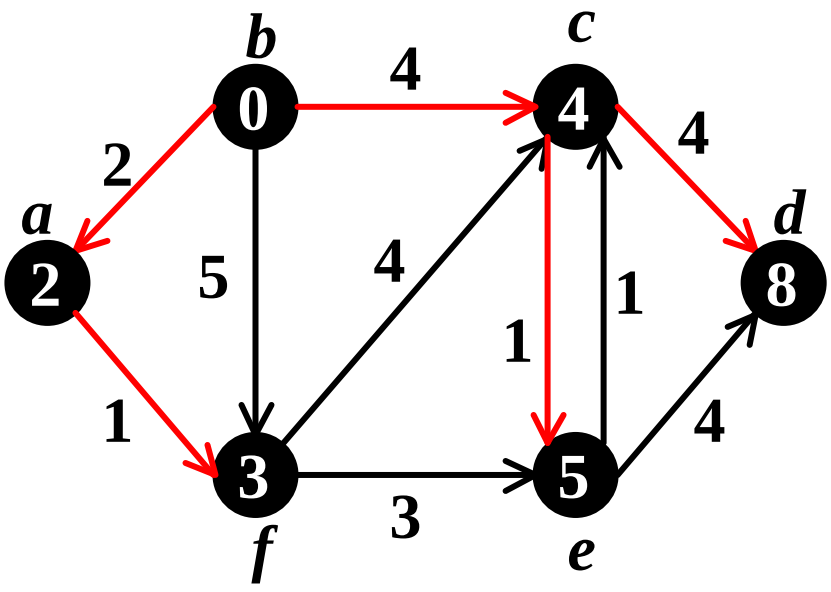
Corollario: **Se eseguiamo l'*algoritmo di Dijkstra* su un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$ che mappa archi in pesi a valori reali *non-negativi*, e un vertice sorgente s, allora, alla terminazione il *sottografo dei predecessori* G_p corrisponde all'*albero dei percorsi minimi* con *radice* s.**

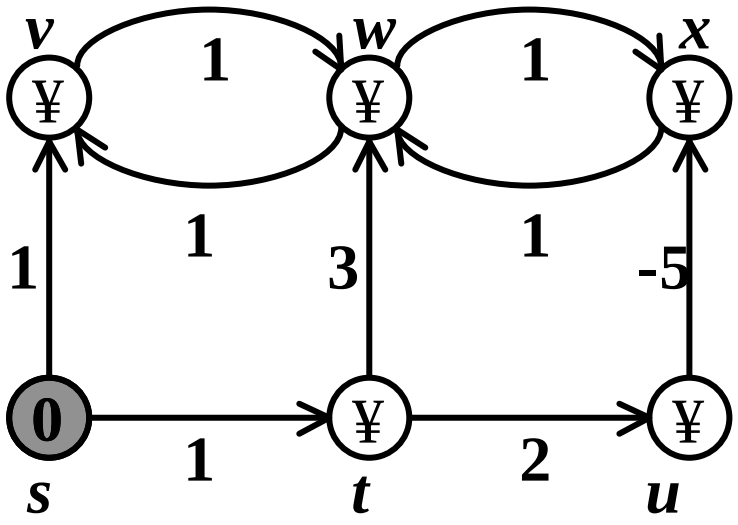
***Dimostrazione Corollario*: Conseguenza immediata del *Teorema di correttezza* e del *Lemma 7*.**

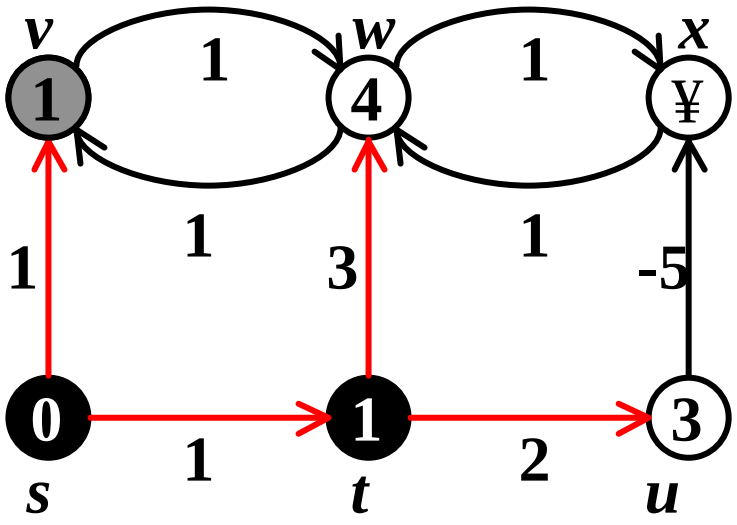
Esercizio

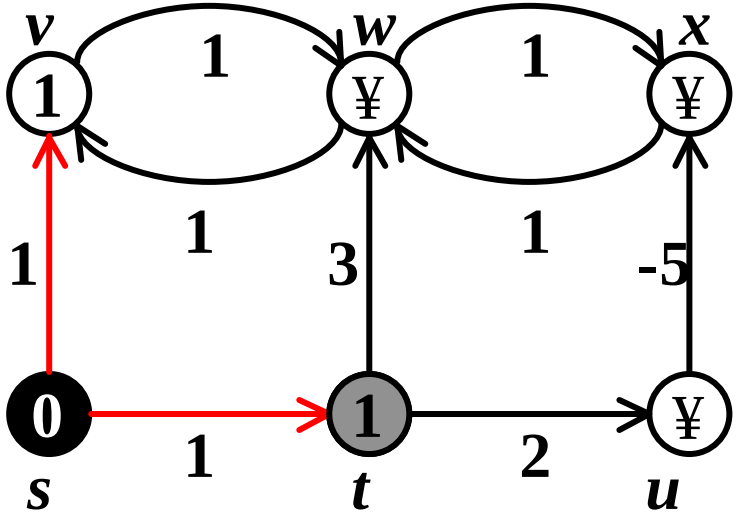
- **Trovare il percorso minimo da *b* ad ogni altro vertice**

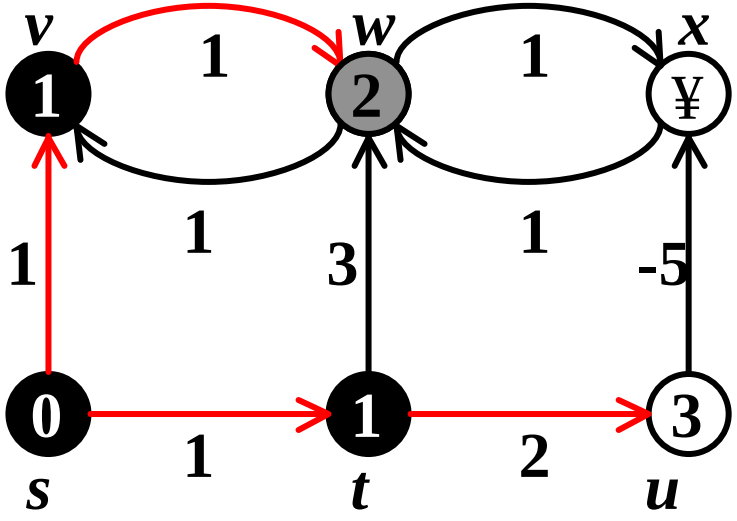


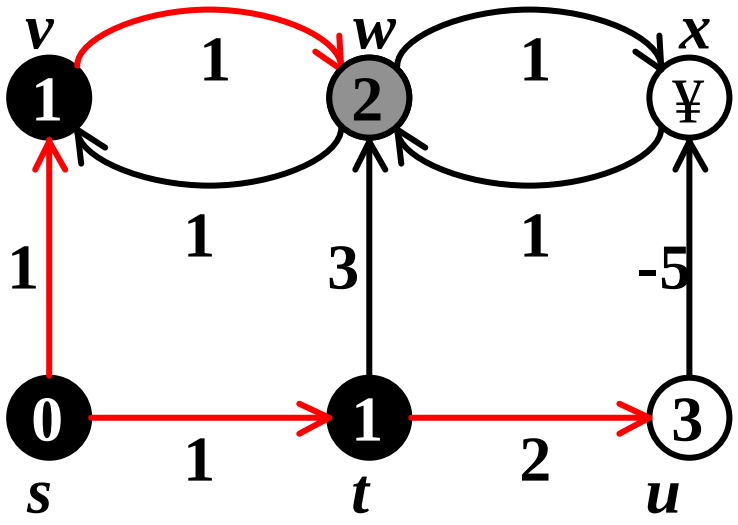


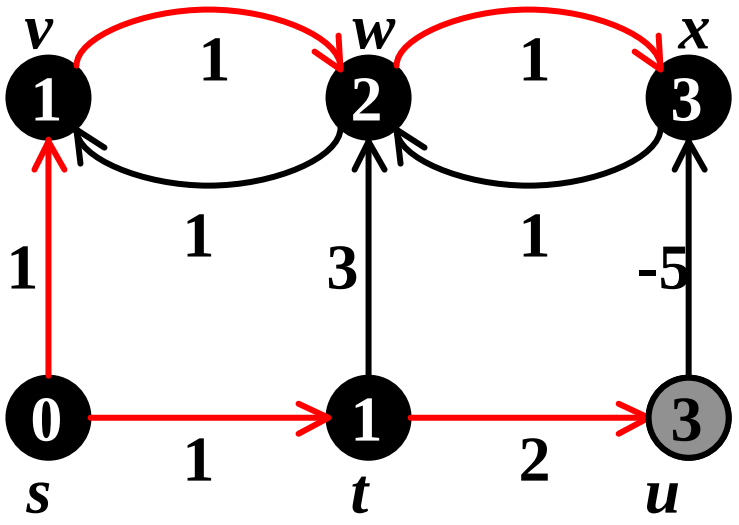

v
w
x
¥
¥
¥
1
1
1
1
3
1
-5
0
¥
¥
1
2
s
t
u

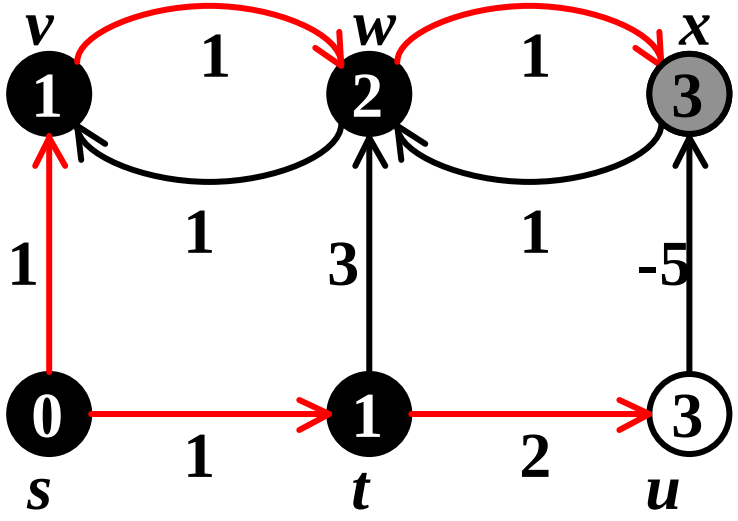

v
w
x
1
4
¥
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u

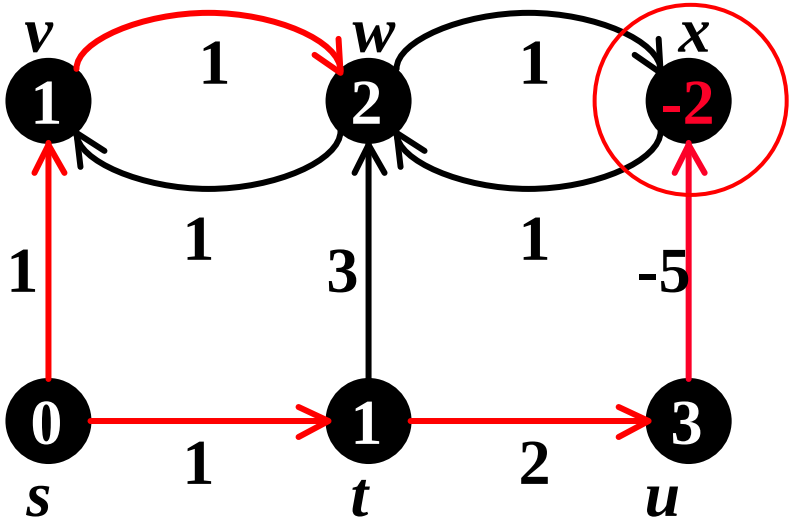

v
w
x
1
¥
¥
1
1
1
1
3
1
-5
0
1
¥
1
2
s
t
u


v
w
x
1
2
¥
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u

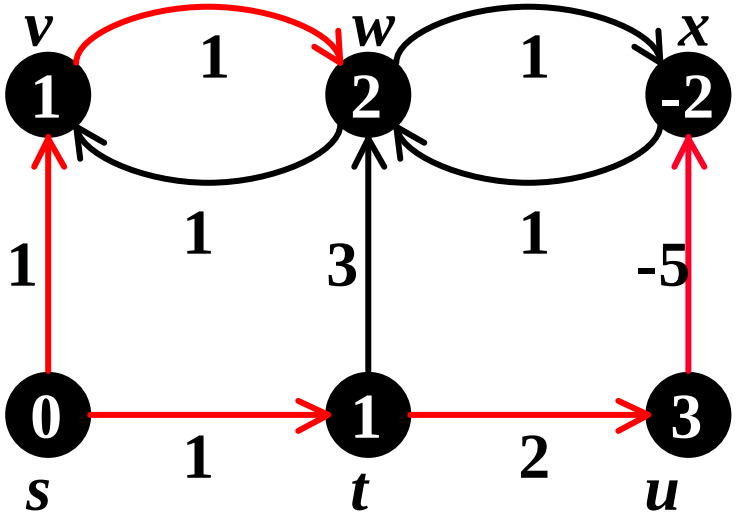

v
w
x
1
2
¥
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u


v
w
x
1
2
3
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u

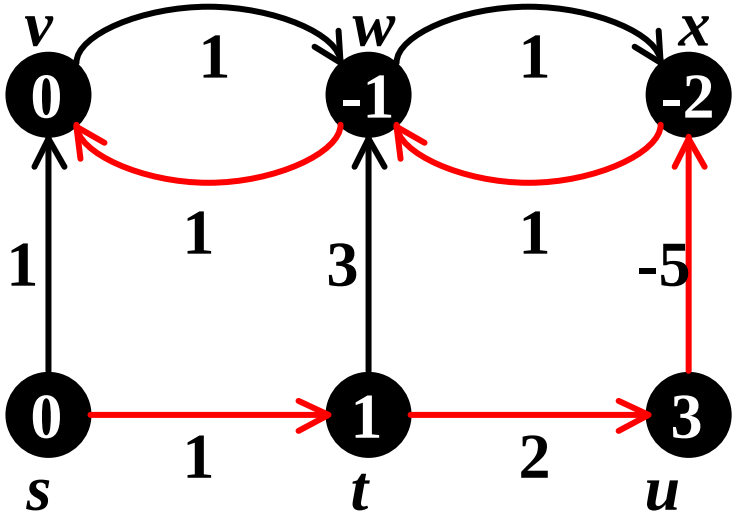

v
w
x
1
2
3
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u


v
w
x
1
2
-2
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u

Problemi con l'Algoritmo di Dijkstra



Soluzione di Dijkstra



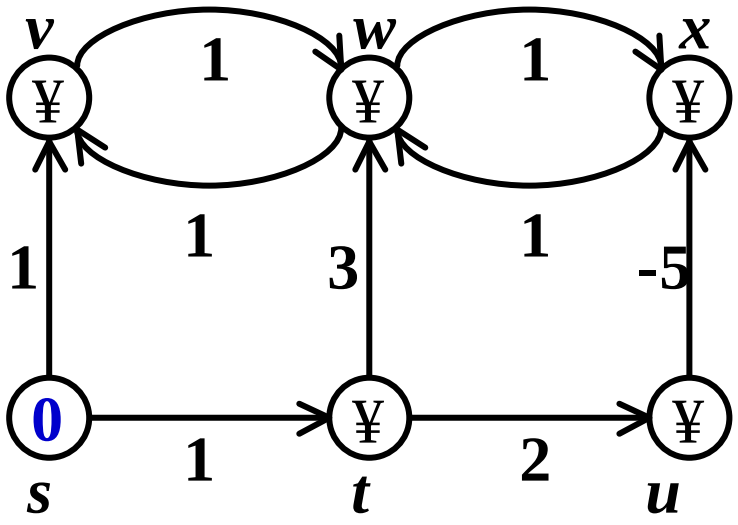
Soluzione Ottimale

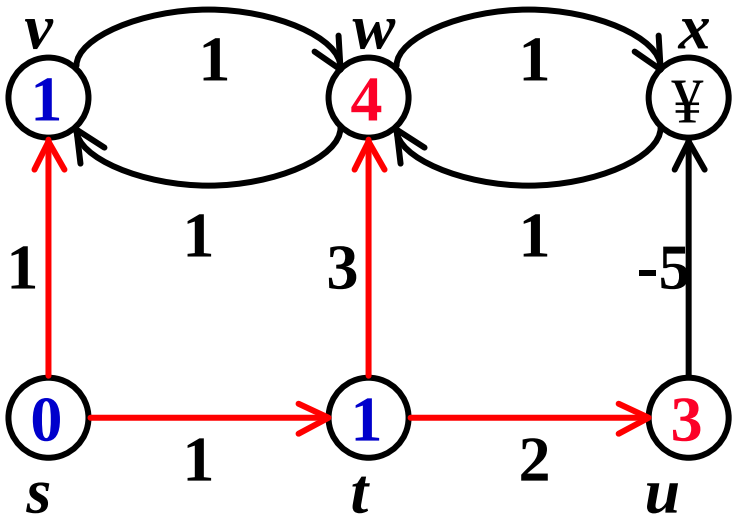
Algoritmo di Bellman-Ford

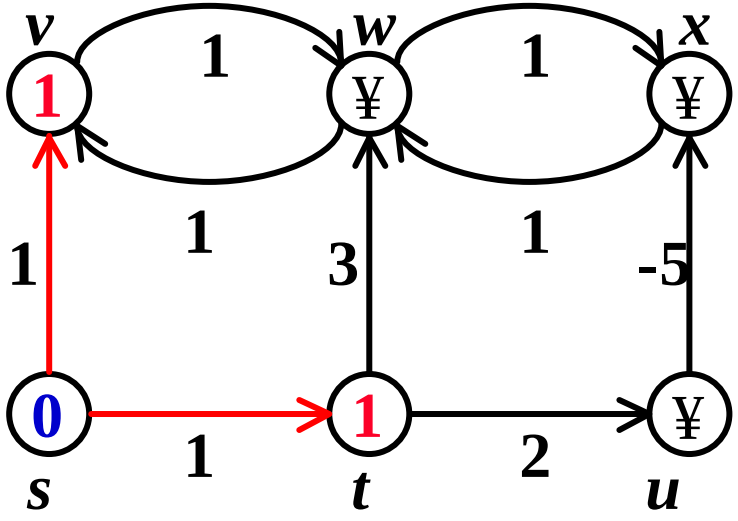
- **Inizalmente, $d[s] = 0$ e $d[u] = ¥$ per u ^{1} s**
- **Vengono fatte $|V| - 1$ passate**
- **Ad ogni passata, si applica il rilassamento *a ogni arco***

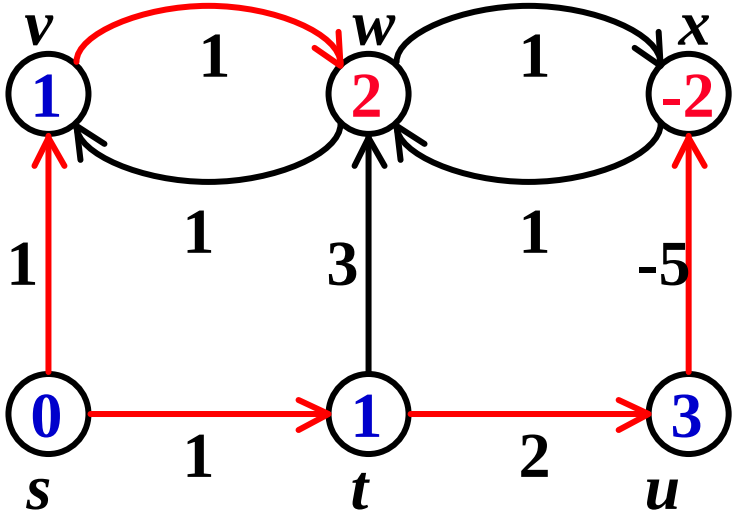
Tempo = $O(|V||E|)$

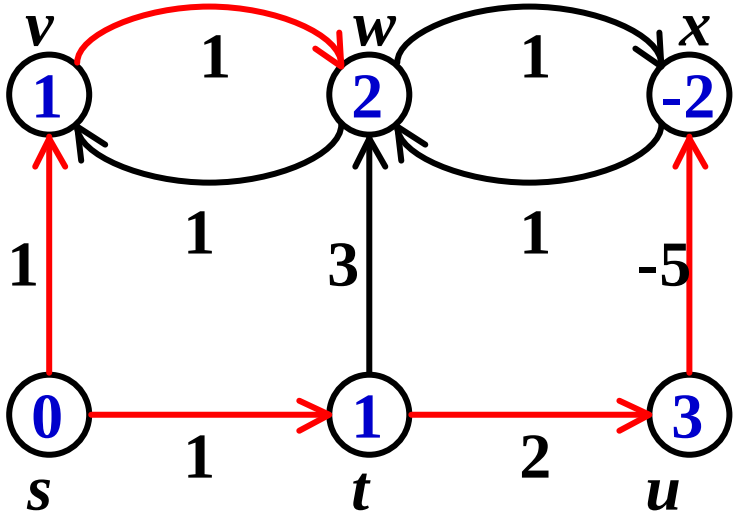
```
Bellman_Ford(G,s)
  Inizializza(G,s,d)
  for i = 1 to |V|-1 do
    for each arco (u,v) in E do
        relax(u,v,w)
  for each arco (u,v) in E do
      if d[v] > d[u] + w(u,v) then
          return FALSE
  return TRUE
```

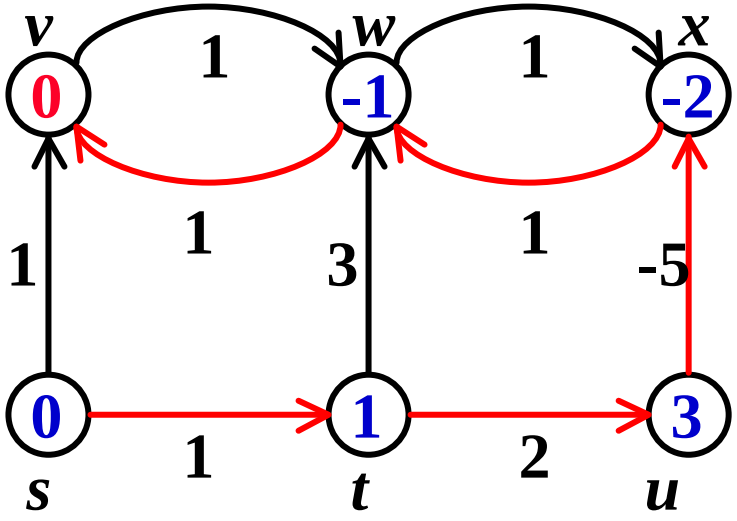

v
w
x
¥
¥
¥
1
1
1
1
3
1
-5
0
¥
¥
1
2
s
t
u

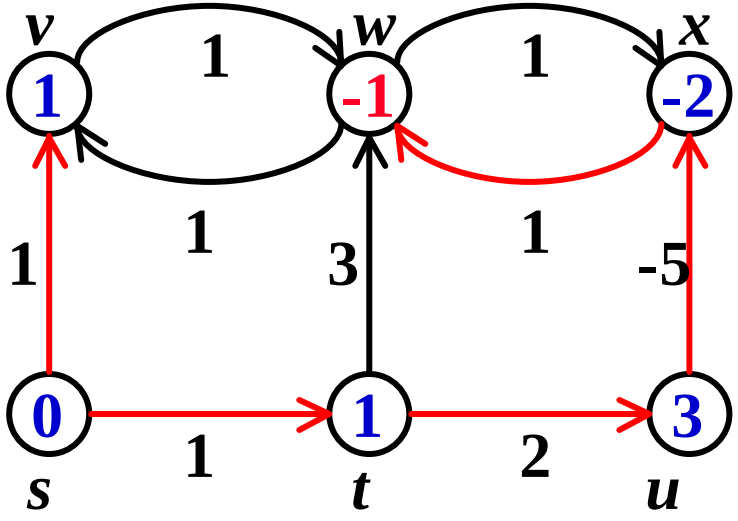

v
w
x
1
4
¥
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u


v
w
x
1
¥
¥
1
1
1
1
3
1
-5
0
1
¥
1
2
s
t
u


v
w
x
1
2
-2
1
1
1
1
3
1
-5
0
1
3
1
2
s
t
u


v
w
x
1
2
-2
1
1
1
1
3
-5
0
1
3
1
2
s
t
u


v
w
x
0
-1
-2
1
1
1
1
3
-5
0
1
3
1
2
s
t
u


v
w
x
1
-1
-2
1
1
1
1
3
-5
0
1
3
1
2
s
t
u

Bellman-Ford: correttezza

Teorema: **Si esegua l'*algoritmo di Bellman-Ford* su un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$, e un vertice sorgente *s*:**

- ***se non esistono cicli negativi raggiungibili da s*, *allora* l'algoritmo ritorna `TRUE`, $d[v] = \delta(s,v)$, $\forall v \in V$, inoltre il *grafo dei predecessori* G_p è l'*albero dei percorsi minimi*;**
- ***se esistono cicli negativi raggiungibili da s*, *allora* l'algoritmo ritorna `FALSE`.**

Bellman-Ford: correttezza

Lemma 8: **Si esegua l'*algoritmo di Bellman-Ford* su un grafo pesato orientato $G = (V, E)$, con funzione di peso $w: E \to \mathbb{R}$, e un vertice sorgente s e *non esistano cicli negativi raggiungibili da s*. *Allora*, alla terminazione, $d[v] = \delta(s,v)$ per tutti i vertici v in V.**

Bellman-Ford: correttezza

Dimostrazione: **Sia v un vertice raggiungibile da s e $p=\langle v_0,v_1,\ldots,v_k\rangle$ un percorso minimo tra s e v ($v_0=s$ e $v_k=v$).**

Poiché p deve essere *semplice*, non può avere più di $|V|-1$ archi (cioè $k \leq |V|-1$).

Per *induzione* sul numero di passate, dimostriamo che, dopo i passate sugli archi di G, si ha $d[v_i] = \delta(s,v_i)$ e che poi non cambia più.

Poiché vengono fatte esattamente $|V|-1$ passi dall'algoritmo, questo è *sufficiente a dimostrare* il *lemma*.

Bellman-Ford: correttezza

Dimostrazione: **Sia v un vertice raggiungibile da s e $p=\langle v_0,v_1,\ldots,v_k\rangle$ un percorso minimo tra s e v ($v_0=s$ e $v_k=v$).**

Per induzione su k dimostriamo che, dopo i passi sugli archi di G, $d[v_i] = \mathrm{d}(s,v_i)$ e che poi non cambia più.

Base: **$d[v_0] = d[s] = 0 = \mathrm{d}(s,s)$ e il *Lemma 4* garantisce che non cambia più.**

Induzione: **Assumiamo che valga per i-1 cioè che $d[v_{i-1}] = \mathrm{d}(s,v_{i-1})$ dopo il passo i-1.**

Si rilassa l'arco (v_{i-1},v_i) al passo i e per il *Lemma 5* concludiamo che $d[v_i] = \mathrm{d}(s,v_i)$ da allora in poi.

Bellman-Ford: correttezza

Corollario 3: **Sia $G = (V, E)$ un grafo pesato orientato con funzione di peso $w: E \to \mathbb{R}$, e s un vertice sorgente. Allora, per ogni nodo $v \in S$, *esiste un percorso* da s a v *se e solo* se alla terminazione dell'*algoritmo di Bellman-Ford* vale $d[v] < \infty$.**

Dimostrazione: **Simile al *Lemma 8* precedente.**

Bellman-Ford: correttezza

Dimostrazione teorema:

caso 1: Non ci sono cicli negativi **raggiungibili da** ***s***

Allora per ogni $v \in V$ **vale** $d[v] = \delta(s,v)$

infatti, se ***v*** **è** ***raggiungibile*** **da** ***s*** ***Lemma 8***

se ***v non*** **è** ***raggiungibile*** **da** ***s*** ***Corollario 3***

Da questo e dal ***Lemma 7*** **sappiamo inoltre che** G_p **è un albero dei** ***percorsi minimi*****.**

L'algoritmo restituisce `TRUE` **poiché, per ogni** $(u,v) \in E$**, vale che** $d[v] = \delta(s,v)$ **e per** ***Lemma 2*****,**

$d[v] = \delta(s,v) \leq \delta(s,u) + w(u,v) = d[u] + w(u,v)$**.**

Quindi ***nessuno dei controlli*** **dell'algoritmo (dopo il** ***while*****)** ***è verificato*****, perciò ritorna** `TRUE`**.**

Bellman-Ford: correttezza

Dimostrazione teorema:

caso 2: Ci sono cicli negativi **raggiungibili da s, e sia $p=\langle v_0,v_1,\ldots,v_k\rangle$ uno di tali cicli ($v_0 = v_k$).**

Allora $\sum_{i=1\ldots k} w(v_{i-1},v_i) < 0$

Contraddizione!

Supponiamo che l'algoritmo ritorni TRUE.

Allora $d[v_i] \leq d[v_{i-1}] + w(v_{i-1},v_i)$ **per** $i = 1,\ldots,k$.

Sommando le disuguaglianze lungo il ciclo si ha

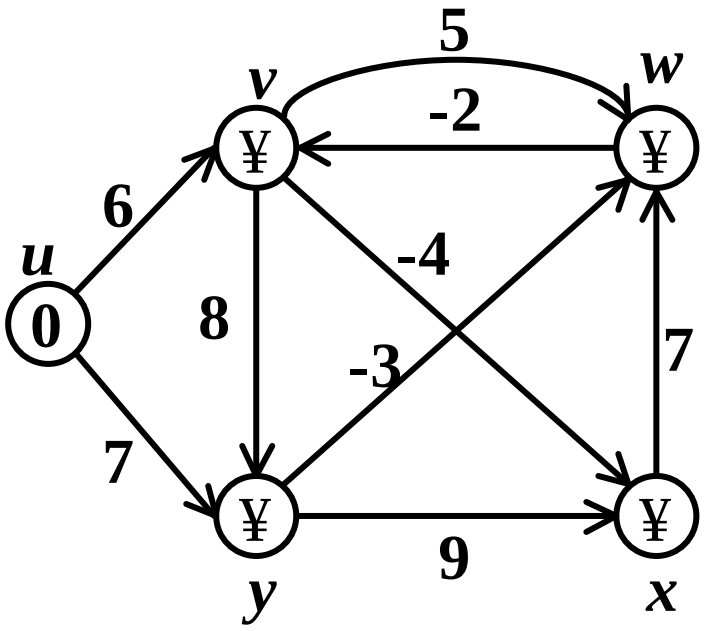
$$\sum_{i=1\ldots k} d[v_i] \leq \sum_{i=1\ldots k} d[v_{i-1}] + \sum_{i=1\ldots k} w(v_{i-1},v_i)$$

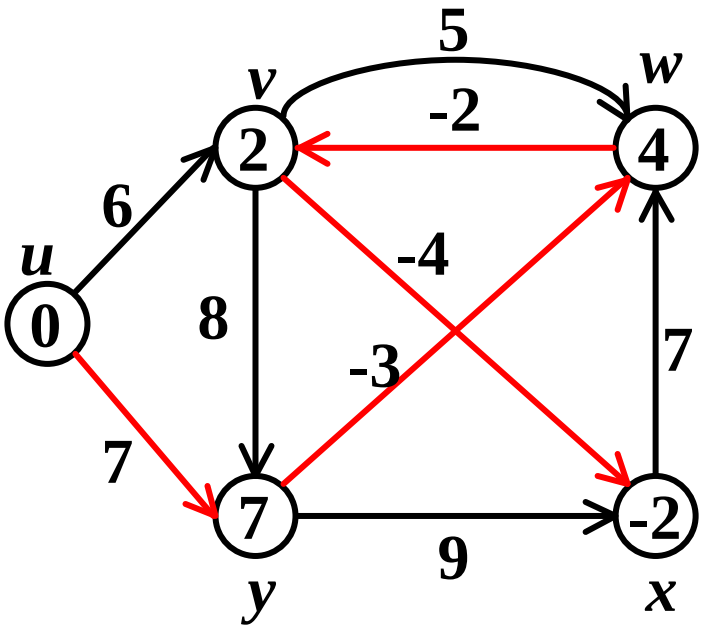
Ma come per il *Lemma 6* otterremo che

$$0 \leq \sum_{i=1\ldots k} w(v_{i-1},v_i)$$

Esercizio

- **Trovare il percorso minimo da *u* ad ogni altro vertice**





Grafi: Percorsi minimi

- **Percorsi minimi da una singola sorgente**
  - **Trovare il percorso minimo dalla sorgente *s* ad ogni altro vertice**
- **Percorsi minimi per tutte le coppie**
  - **Trovare il percorso minimo tra tutte le coppie di vertici**
    - **Input: $G = (V, E)$ un *grafo orientato* e**

      $w: E \to \mathbb{R}$ una *funzione di peso*.
    - **Output: per ogni coppia di vertici $i,j \in V$ il *percorso minimo* tra i e j.**

      Una tabella $D=(d_{ij})$ contenente la *distanza minima* tra i vertici i e j.

Grafi: Percorsi minimi

- **Percorsi minimi da una singola sorgente**
  - **Trovare il percorso minimo dalla sorgente *s* ad ogni altro vertice**
- **Percorsi minimi per tutte le coppie**
  - **Trovare il percorso minimo tra tutte le coppie di vertici**
    - **Eseguire l'*algoritmo di Dijkstra* $|V|$ *volte* (ma solo se i pesi sono *non-negativi*)**
    - **Eseguire l'*algoritmo di Bellman-Ford* $|V|$ *volte***

Grafi: Percorsi minimi

- **Percorsi minimi da una singola sorgente**
  - **Trovare il percorso minimo dalla sorgente *s* ad ogni altro vertice**
- **Percorsi minimi per tutte le coppie**
  - **Trovare il percorso minimo tra tutte le coppie di vertici**
    - **Eseguire l'*algoritmo di Dijkstra* $|V|$ *volte* (ma solo se i pesi sono *non-negativi*)**

$O(|V|^3)$ **o** $O(|V||E| \log |V|)$

- **Eseguire l'*algoritmo di Bellman-Ford* $|V|$ *volte***

$O(|V|^2|E|)$ → **grafo denso** → $O(|V|^4)$

Grafi: Percorsi minimi

- **Percorsi minimi per tutte le coppie**
  - **Trovare il percorso minimo tra tutte le coppie di vertici**
    - **Input**: $G = (V, E)$ **un** ***grafo orientato***

      $w: E \to \mathbb{R}$ **una** ***funzione di peso***

      Il grafo può allora essere rappresentato come una ***matrice di adiacenza*** $W = (w_{ij})$ **di dimensione** $n \times n$**, così definita:**

$$w_{ij} = \begin{cases} 0 & \text{se } i = j \\ \text{peso di } (i, j) & \text{se } i \neq j \text{ e } (i,j) \in E \\ \infty & \text{se } i \neq j \text{ e } (i,j) \notin E \end{cases}$$

Grafi: Percorsi minimi

- **Percorsi minimi per tutte le coppie**
  - **Trovare il percorso minimo tra tutte le coppie di vertici**
    - **Output: per ogni coppia di vertici** $i,j \in V$ **il** ***percorso minimo*** **tra** i **e** j**.**

      Una tabella $D=(d_{ij})$ **contenente la** ***distanza minima*** **tra i vertici** i **e** j**.**

$$d_{ij} = \begin{cases} 0 & \text{se } i = j \\ d(i,j) & \text{se } i \neq j \text{ e } j \text{ è raggiungibile da } i \\ \infty & \text{se } i \neq j \text{ e } j \text{ non è raggiungibile da } i \end{cases}$$

Grafi: Percorsi minimi

- **Percorsi minimi tra tutte le coppie**
  - **Trovare il percorso minimo tra tutte le coppie di vertici**
    - **Output: per ogni coppia di vertici** $i,j \hat{I} V$ **il** ***percorso minimo*** **tra** i **e** j**.**

      Una tabella $D=(d_{ij})$ **contenente la** ***distanza minima*** **tra i vertici** i **e** j**.**

      Una tabella $P=(p_{ij})$ **contenente il** ***predecessore*** **di** j **lungo il percorso minimo da** i **a** j**.**

$$p_{ij} = \begin{cases} Nil & se\ i = j \\ k & se\ i \neq j\ e\ j \text{ è raggiungibile da } i\ e\ k \text{ è il predecessore di } j \text{ lungo il percorso minimo} \\ Nil & se\ i \neq j\ e\ j \text{ non è raggiungibile da } i \end{cases}$$

Percorsi minimi: con matrici

Programmazione Dinamica

1. ***Caratterizzare*** **la** ***struttura*** **di una** ***soluzione ottima***
2. ***Definire*** ***ricorsivamente*** **il valore di una** ***soluzione ottima***
3. **Calcolare il** ***valore di una soluzione*** **ottima "*bottom-up*"**
4. **Cotruzione di una soluzione ottima.**

❶ *Carattedizzazione della soluzione ottima*

***Struttura della soluzione ottima*.**

Data la matrice $W=(w_{ij})$ di adiacenza del grafo pesato, consideriamo il *percorso minimo p* tra i vertici *i* e *j*.

Supponiamo che *p* contenga *al più k* archi.

Assumendo che *non* esistano *cicli negativi*, *k* sarà allora *finito*.

- **Se *i = j*, allora *p* ha *peso* 0 e *nessun arco*.**

❶ *Carattedizzazione della soluzione ottima*

***Struttura della soluzione ottima*.**

Data la matrice $W=(w_{ij})$ di adiacenza del grafo pesato, consideriamo il *percorso minimo p* tra i vertici *i* e *j*.

Supponiamo che *p* contenga *al più k* archi.

Assumendo che *non* esistano *cicli negativi*, *k* sarà allora *finito*.

- **Se $i \overset{1}{\;} j$, allora *p* sarà della forma $i \overset{p'}{\Rightarrow} m \to j$ dove *p'* contiene *al più k*-1 archi ed è un *percorso minimo* tra *i* e *m* *(Lemma 1)*.**

Ma allora (*Corollario 2*) $d(i,j) = d(i,m) + w_{mj}$

❷ *Definizione ricorsiva della soluzione ottima*

- $d_{ij}^{(k)}$ = **il peso del *percorso minimo* tra i vertici v_i e v_j usando al più *k archi*.**
- **Vogliamo calcolare $d_{ij}^{(|V|-1)}$ per tutte le coppie *i,j*.**
- **Iniziamo a calcolare il *caso base*, quando non abbiamo *alcun vertice* a disposizione per *formare percorsi*, cioè quando $k=0$.**

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{se } i = j \\ \infty & \text{se } i \neq j \end{cases}$$

❷ *Definizione ricorsiva della soluzione ottima*

- $d_{ij}^{(k)}$ = **il peso del *percorso minimo* tra i vertici v_i e v_j usando al più *k archi*.**
- **Vogliamo calcolare $d_{ij}^{(|V|-1)}$ per tutte le coppie *i,j*.**
- **Supponiamo di avere a disposizione la matrice $d_{ij}^{(k-1)}$ dei *percorsi minimi contenenti al più k*-1 archi.**
- **Il *percorso minimo* di tra v_j e v_j di al più *k* archi, o contiene al più *k*-1 archi e ha costo $d_{ij}^{(k-1)}$,**
- **o contiene esattamente *k* archi e sarà composto da un *percorso minimo* di *k*-1 archi più un nodo v_m con un arco a v_i e dal passo 1 sappiamo che in tal caso il costo sarà $d_{ij}^{(k)} = d_{im}^{(k-1)} + w_{mj}$**

❷ *Definizione ricorsiva della soluzione ottima*

- $d_{ij}^{(k)}$ = **il peso del *percorso minimo* tra i vertici v_i e v_j usando al più *k archi*.**
- **Vogliamo calcolare $d_{ij}^{(|V|-1)}$ per tutte le coppie *i,j*.**

$$d_{ij}^{(k)} = \min \left\{ d_{ij}^{(k-1)}, \min_{1 \le m \le n} \left\{ d_{im}^{(k-1)} + w_{mj} \right\} \right\}$$

$$= \min_{1 \le m \le n} \left\{ d_{im}^{(k-1)} + w_{mj} \right\}$$

Poiché $w_{jj}=0$, quindi se $m=j$, $d_{ij}^{(k-1)} = d_{ij}^{(k-1)} + w_{jj}$

❷ *Definizione ricorsiva della soluzione ottima*

- **Come si *calcolano* i *costi effettivi* dei *percorsi minimi*?**
- **Quante matrici $D^{(k)} = (d_{ij}^{(k)})$ dobbiamo calcolare?**
- ***Per quale valore* di k *otteniamo* in $D^{(k)}$ i *pesi* (costi) dei *percorsi minimi*?**

$$d_{ij}^{(k)} = \min_{1 \le m \le n} \left\{ d_{im}^{(k-1)} + w_{mj} \right\}$$

❷ *Definizione ricorsiva della soluzione ottima*

- **Come si *calcolano* i *costi effettivi* dei *percorsi minimi*?**
- **Quante matrici $D^{(k)} = (d_{ij}^{(k)})$ dobbiamo calcolare?**
- ***Per quale valore* di *k* *otteniamo* in $D^{(k)}$ i *pesi* (costi) dei *percorsi minimi*?**

- ***Se*, come da ipotesi, il grafo *non contiene cicli di peso negativo*,**
- ***allora* ogni *percorso minimo* sarà un *percorso semplice* con *al più* $|V|$-1 *archi*.**

E quindi $\mathrm{d}(i,j) = d_{ij}^{(|V|-1)} = d_{ij}^{(|V|)} = d_{ij}^{(|V|+1)} = \ldots$

❸ *Calcolo del valore della soluzione ottima*

- **L'algoritmo riceve in ingresso la matrice $W=(w_{ij})$ di adiacenza del grafo pesato.**
- **Viene quindi calcolata una sequenza di matrici $D^{(0)} = D^{(1)} = D^{(2)} = \ldots = D^{(|V|-1)}$, dove per $k=1,2,\ldots,|V|-1$ abbiamo $D^{(k)} = (d_{ij}^{(k)})$.**
- **La matrice $D^{(|V|-1)}$ conterrà i *pesi* dei *percorsi minimi*.**
- **La matrice $D^{(|1)} = W$**

❸ *Calcolo del valore della soluzione ottima*

```
Slow_All_Pair_Shst_Paths(W:matrice)
  n = righe[W]
  D(1) = W
  for k = 2 to n-1
    do D(k) = Extend_Shst_Paths(D(k-1),W)
  return D(n-1)
```

$\Theta(n\,T(Extend\_Shst\_Path))$

- **La procedura `Extend_Shst_Paths` è la procedura che, date le matrici $D^{(k-1)}$ e W calcola la matrice $D^{(k)}$ secondo la formula**

$$d_{ij}^{(k)} = \min_{1 \le m \le n} \left\{ d_{im}^{(k-1)} + w_{mj} \right\}$$

❸ *Calcolo del valore della soluzione ottima*

```
Extend_Shst_Paths(D,W:matrice)
  n = righe[D]
  "Sia D' una nuova matrice n'n"
  for i = 1 to n
    do for j = 1 to n
          do d'ij = ¥
             for m = 1 to n
               do d'ij=min(d'ij,dim+Wmj)
  return D'
```

$\Theta(n^3)$

- La procedura `Extend_Shst_Paths` è la procedura che, date le matrici $D^{(k-1)}$ e W calcola la matrice $D^{(k)}$ secondo la fomula

$$d_{ij}^{(k)} = \min_{1 \le m \le n} \left\{ d_{im}^{(k-1)} + w_{mj} \right\}$$

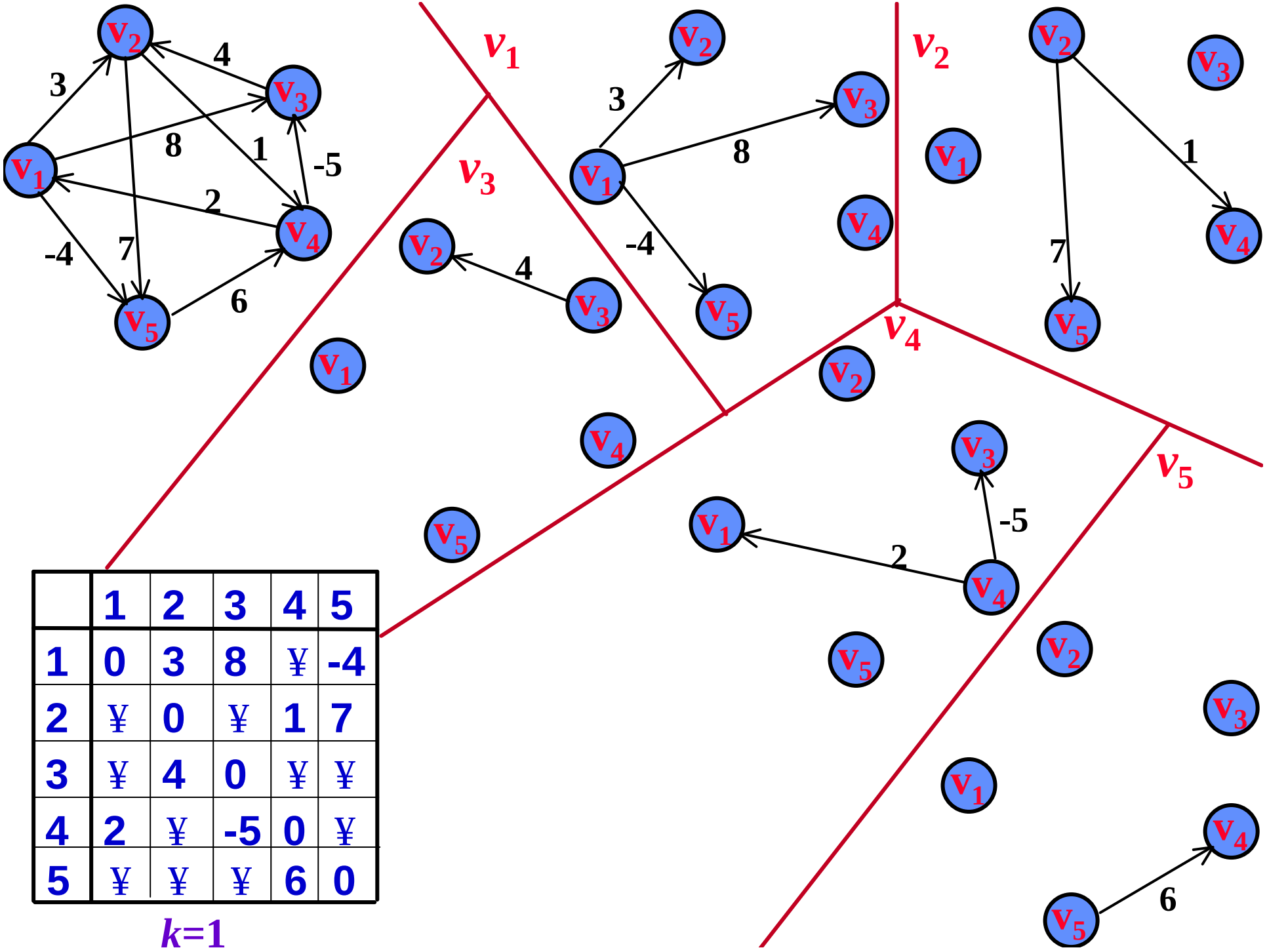
❸ *Calcolo del valore della soluzione ottima*

```
Slow_All_Pair_Shst_Paths(W:matrice)
  n = righe[W]
  D(1) = W
  for k = 2 to n-1
    do D(k) = Extend_Shst_Paths(D(k-1),W)
  return D(n-1)
```

$\Theta(n^4)$

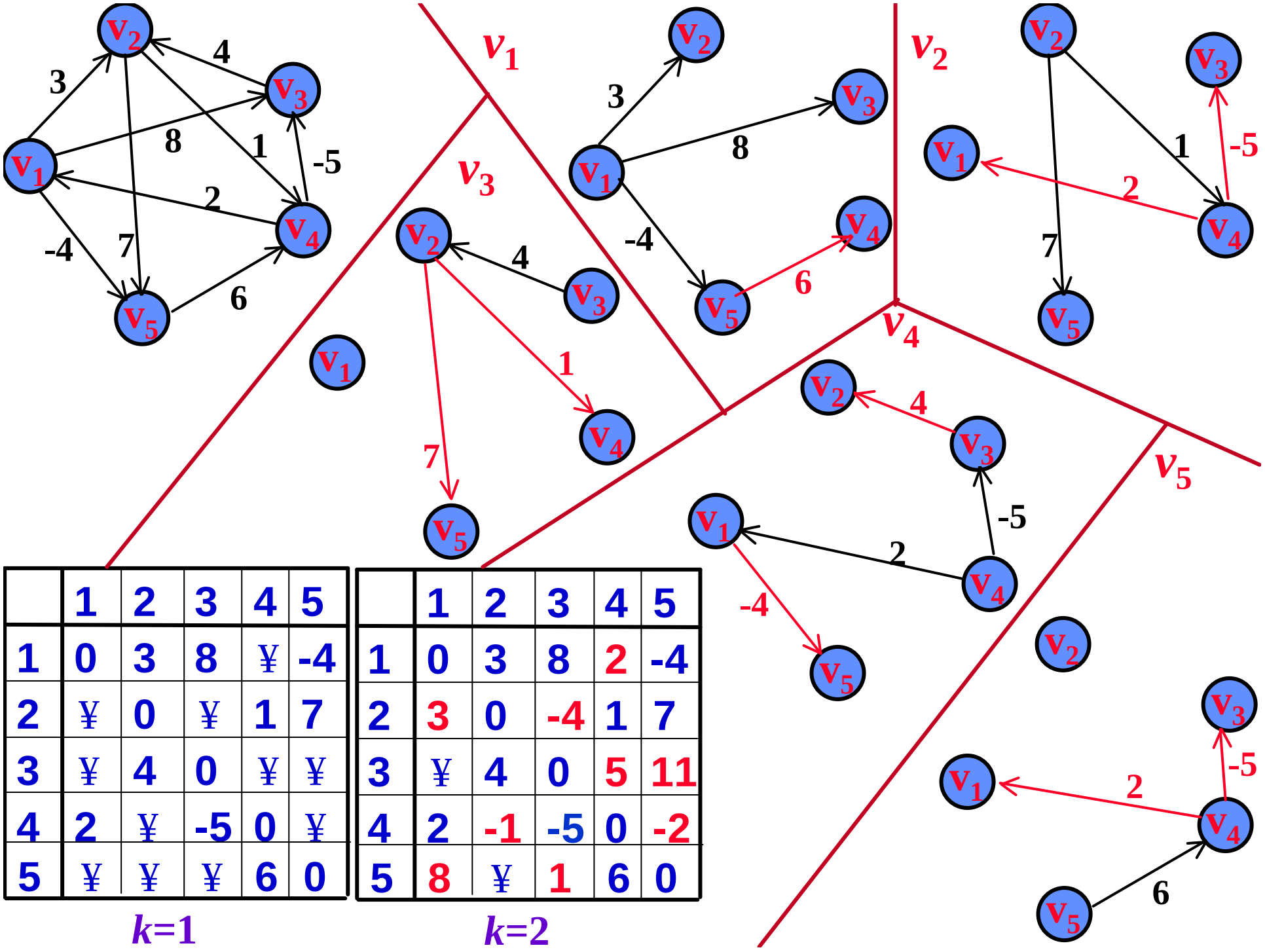
- La procedura `Extend_Shst_Paths` è la procedura che, date le matrici $D^{(k-1)}$ e W calcola la matrice $D^{(k)}$ secondo la fomula

$$d_{ij}^{(k)} = \min_{1 \le m \le n} \left\{ d_{im}^{(k-1)} + w_{mj} \right\}$$



	1	2	3	4	5
1	0	3	8	¥	-4
2	¥	0	¥	1	7
3	¥	4	0	¥	¥
4	2	¥	-5	0	¥
5	¥	¥	¥	6	0

k=1

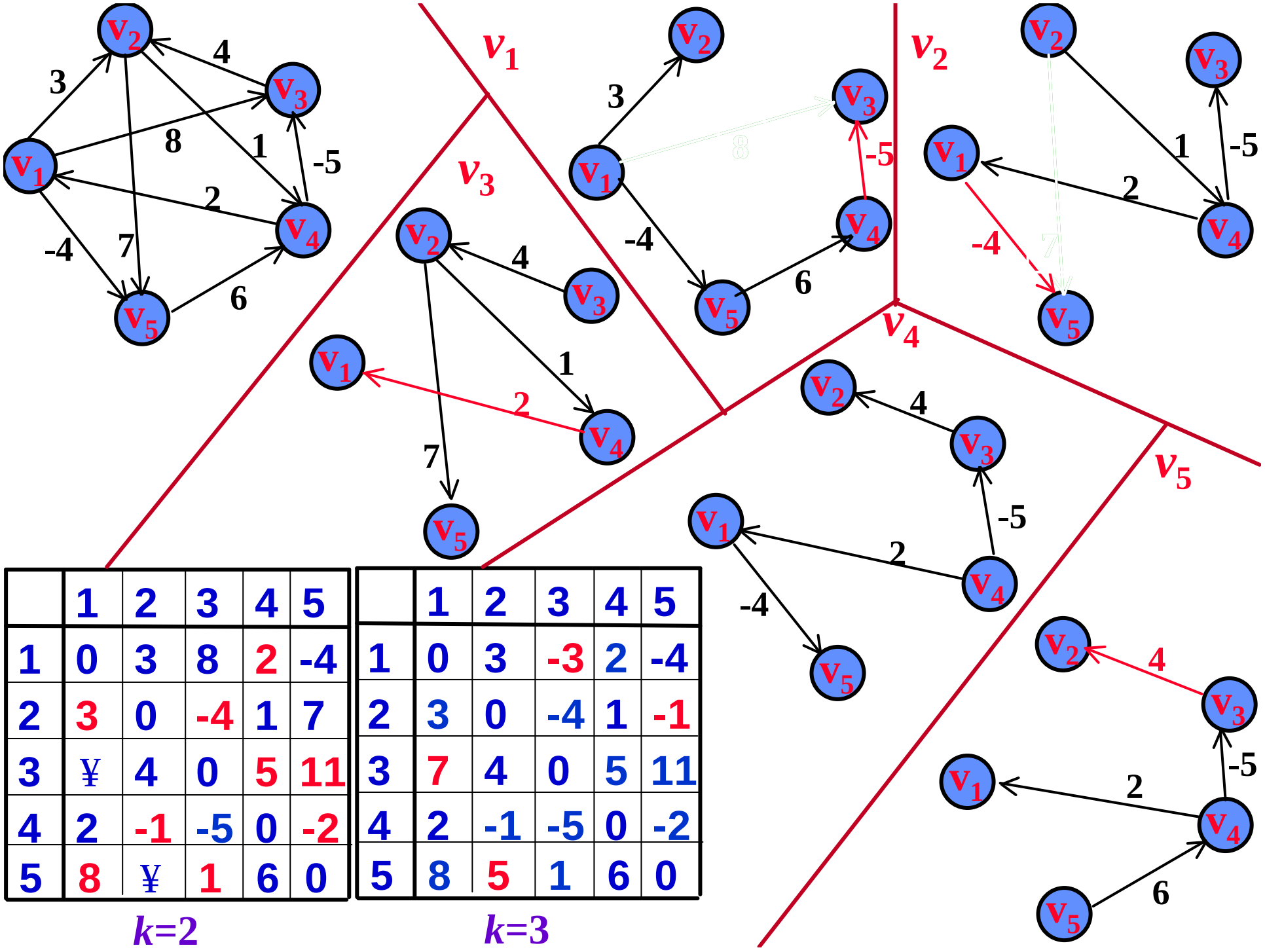


	1	2	3	4	5
1	0	3	8	¥	-4
2	¥	0	¥	1	7
3	¥	4	0	¥	¥
4	2	¥	-5	0	¥
5	¥	¥	¥	6	0

k=1

	1	2	3	4	5
1	0	3	8	2	-4
2	3	0	-4	1	7
3	¥	4	0	5	11
4	2	-1	-5	0	-2
5	8	¥	1	6	0

k=2

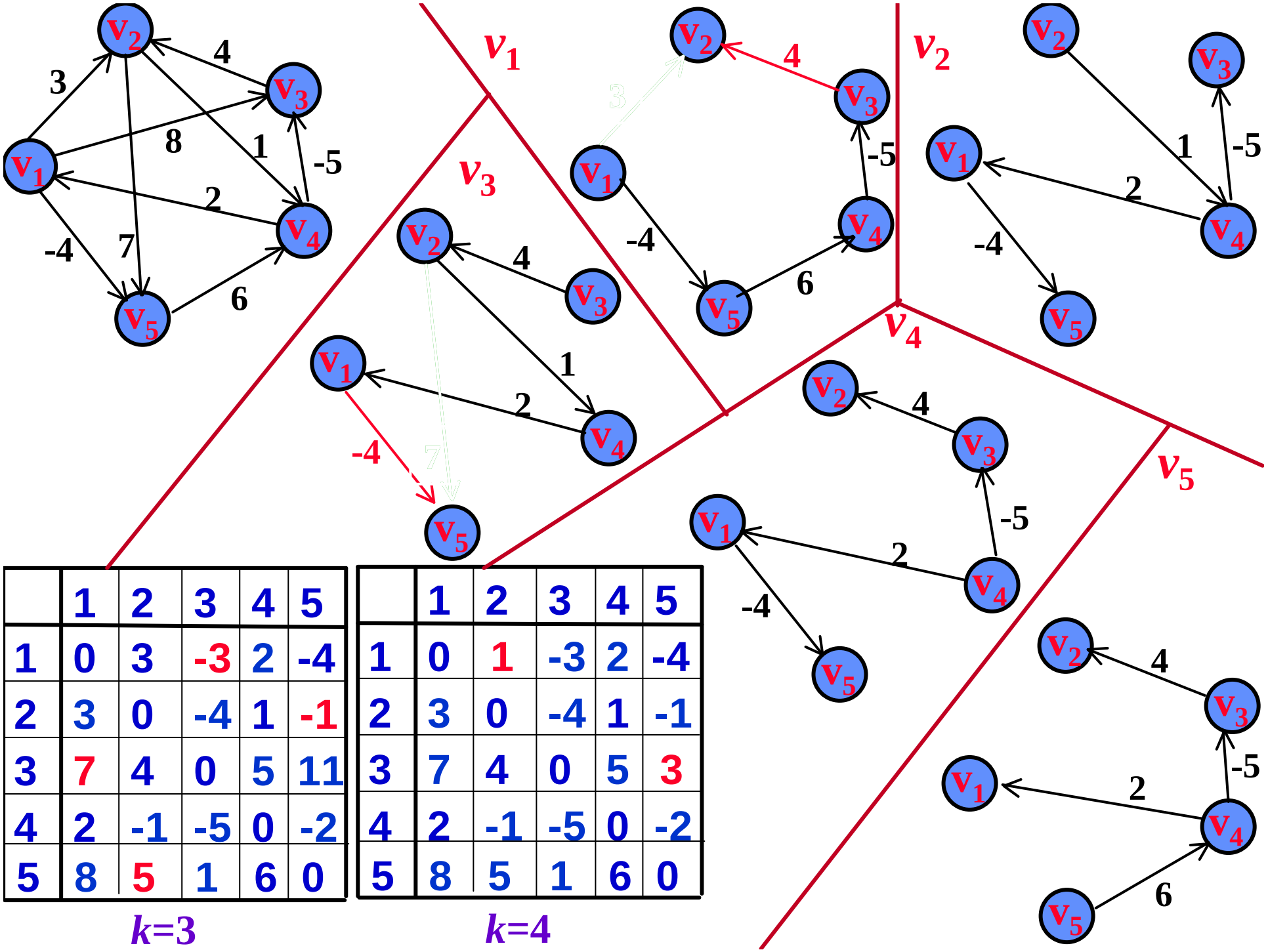


	1	2	3	4	5
1	0	3	8	2	-4
2	3	0	-4	1	7
3	¥	4	0	5	11
4	2	-1	-5	0	-2
5	8	¥	1	6	0

k=2

	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

k=3



	1	2	3	4	5
1	0	3	-3	2	-4
2	3	0	-4	1	-1
3	7	4	0	5	11
4	2	-1	-5	0	-2
5	8	5	1	6	0

k=3

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

k=4

❸ *Calcolo del valore della soluzione ottima*

- **Possiamo *migliorare il tempo di esecuzione* dell'algoritmo?**
- **In effetti *vengono calcolate* più matrici di quelle necessarie, o meglio *matrici inutili*.**

$$D^{(1)} = D^{(0)} \text{ Å } W = W$$
$$D^{(2)} = D^{(1)} \text{ Å } W = W^2$$
$$D^{(3)} = D^{(2)} \text{ Å } W = W^3$$
$$D^{(4)} = D^{(3)} \text{ Å } W = W^4$$
$$\ldots$$
$$D^{(n-1)} = D^{(n-2)} \text{ Å } W = W^{n-1}$$

❸ *Calcolo del valore della soluzione ottima*

- **Possiamo *migliorare il tempo di esecuzione* dell'algoritmo?**
- **In effetti *vengono calcolate* più matrici di quelle necessarie, o meglio *matrici inutili*.**

$$D^{(1)} = D^{(0)} \text{Å} W = W$$

$$D^{(2)} = D^{(1)} \text{Å} W = D^{(1)} \text{Å} D^{(1)} = W^2$$

$$D^{(3)} = D^{(2)} \text{Å} W = W^3$$

$$D^{(4)} = D^{(3)} \text{Å} W = D^{(2)} \text{Å} W \text{Å} W = D^{(2)} \text{Å} D^{(2)} = W^4$$

…

$$D^{(2k)} = D^{(k)} \text{Å} D^{(k)} = W^{2k}$$

Come il prodotto di matrici, anche l'*estensione* è una *operazione associativa*

❸ *Calcolo del valore della soluzione ottima*

- **Possiamo *migliorare il tempo di esecuzione* dell'algoritmo?**
- **In effetti *vengono calcolate* più matrici di quelle necessarie, o meglio *matrici inutili*.**
- **Cosa possiamo fare?**

$$D^{(1)} = D^{(0)} \text{Å} W = W$$
$$D^{(2)} = D^{(1)} \text{Å} D^{(1)} = W^2$$
$$D^{(4)} = D^{(2)} \text{Å} D^{(2)} = W^4$$
$$D^{(8)} = D^{(4)} \text{Å} D^{(4)} = W^8$$
$$\dots$$
$$D^{(2^{\log(n-1)})} = D^{(2^{\log(n-1)-1})} \text{Å} D^{(2^{\log(n-1)-1})} = W^{2^{\log(n-1)}}$$

❸ *Calcolo del valore della soluzione ottima*

- **Cosa possiamo fare?**

$$D^{(1)} = D^{(0)} \text{Å} W = W$$
$$D^{(2)} = D^{(1)} \text{Å} D^{(1)} = W^2$$
$$D^{(4)} = D^{(2)} \text{Å} D^{(2)} = W^4$$
$$D^{(8)} = D^{(4)} \text{Å} D^{(4)} = W^8$$
$$\dots$$
$$D^{(2^{\log(n-1)})} = D^{(2^{\log(n-1)-1})} \text{Å} D^{(2^{\log(n-1)-1})} = W^{2^{\log(n-1)}}$$

- **Il fatto che $d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$ assicura che se $k > \log(n-1)$ allora $2^k > n-1$, quindi $D^{(2^k)} = D^{(n-1)}$.**
- **Possiamo allora fermarci non appena il numero di iterazioni $k = 1,2,3,\dots$ supera il valore $\log(n-1)$.**

❸ *Calcolo del valore della soluzione ottima*

```
Fast_All_Pair_Shst_Paths(W:matrice)
   n = righe[W]
   D(1) = W
   for k = 1 to élog2(n-1)ù do
        D(k) = Extend_Shst_Paths(D(k-1),D(k-1))
   return D(k)
```

Il tempo di esecuzione sarà quindi $\Theta(n^3 \log n)$

❸ *Calcolo del valore della soluzione ottima*

- **Una volta che abbiamo calcolato la matrice delle *distanze minime* $D = D^{(n-1)}$, come possiamo *ricostruire i percorsi minimi* ?**
- **In altre parole come possiamo calcolare la *matrice dei predecessori* $P = P^{(n-1)}$?**
- **Si possono usare essenzialmente due metodi:**
  - **Calcolare una sequenza di *matrici dei predecessori* $P^{(k)}$ per k=1,…,n-1 mentre calcoliamo $D^{(k)}$**
  - **Calcolare P direttamente da D (*Esercizio 26-1.5*)**

Algoritmo di Floyd-Warshall

- **Altro algoritmo di *Programmazione Dinamica***
- **I *pesi* possono essere anche *negativi*, ma si assume sempre che *non* ci siano *cicli negativi***
- **Si *differenzia* dal precedente per il *tipo di sottostruttura* della soluzione che utilizza e quindi per la *nozione di sottoproblema*.**
- ***Miglioramento* del *tempo di esecuzione***

❶ *Caratterizzazione della soluzione ottima*

Consideriamo i *vertici intermedi* di un *percorso semplice* $p=<v_1, v_2, \ldots, v_l>$.

Cioè l'insieme dei vertici $\{v_2, v_3, \ldots, v_{l-1}\}$.

Se $\{1, 2, \ldots, n\}$ sono i vertici del grafo, prendiamone un sottoinsieme $\{1, 2, \ldots, k\}$.

Per ogni coppia di vertici *i,j* consideriamo i *percorsi* che hanno i *vertici solo* in $\{1, 2, \ldots, k\}$.

Sia inoltre *p* il *percorso di peso minimo* (e quindi anche *semplice*) tra di *essi*.

Che *relazione* c'è tra *p* e il *percorso minimo* tra *i* e *j* che ha i *vertici solo* in $\{1, 2, \ldots, k-1\}$?

❶ *Caratterizzazione della soluzione ottima*

Che *relazione* c'è tra *p* e il *percorso minimo* tra *i* e *j* che ha i *vertici solo* in {1, 2 ,…, *k*-1}?

La *relazione* dipenderà dal fatto che *k* sia o meno un *vertice intermedio* del nuovo *percorso minimo p*!

Abbiamo cioè due casi:

- ***k non è* un *vertice intermedio* di *p***
- ***k è* un *vertice intermedio* di *p***

❶ *Caratterizzazione della soluzione ottima*

Che *relazione* c'è tra *p* e il *percorso minimo* tra *i* e *j* che ha i *vertici solo* in {1, 2 ,…, *k*-1}?

Abbiamo cioè due casi:

- ***k non è* un *vertice intermedio* di *p***

  Allora tutti i vertici di *p* percorsi sono contenuti in {1, 2 ,…, *k*-1}.

  Quindi, un *percorso minimo* tra *i* e *j* contenente solo vertici in {1, 2 ,…, *k*-1} è anche un *percorso minimo* tra *i* e *j* contenente solo vertici in {1, 2 ,…, *k*}.

- ***k è* un *vertice intermedio* di *p***

❶ *Caratterizzazione della soluzione ottima*

Che *relazione* c'è tra *p* e il *percorso minimo* tra *i* e *j* che ha i *vertici solo* in {1, 2 ,…, *k*-1}?

Abbiamo cioè due casi:

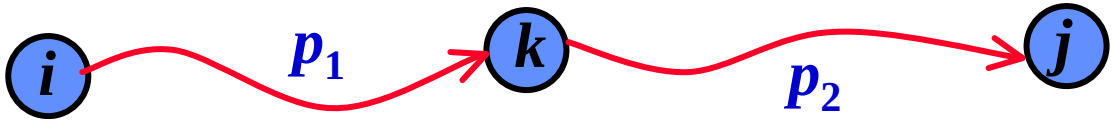
- ***k* non è un vertice intermedio di *p***
- ***k* *è* un *vertice intermedio* di *p***

  Allora *p* può essere scomposto in due percorsi p_1 e p_2 che si congiungono in *k* ($i \overset{p_1}{\Longrightarrow} k \overset{p_2}{\Longrightarrow} j$)

  Per la sottostruttura, p_1 e p_2 sono percorsi minimi da *i* a *k* e da *k* a *j* con vertici in {1, 2 ,…, *k*-1}.

  Infatti *k* non è un vertice intermedio né di p_1 né di p_2,

  e $w(p) = w(p_1) + w(p_2) = d(i,k) + d(k,j)$



❷ *Definizione ricorsiva della soluzione ottima*

- $d_{ij}^{(k)}$ = **il peso del *percorso minimo* tra i vertici v_i e v_j con *vertici intermedi* solo tra v_1, v_2, …, v_k.**
- **Vogliamo calcolare $d_{ij}^{(n)}$ per tutte le coppie *i,j*.**
- **Iniziamo a calcolare il *caso base*, quando *non* abbiamo *vertici intermedi* a disposizione per *formare percorsi*, cioè quando $k=0$.**

$$d_{ij}^{(0)} = w_{ij}$$

❷ *Definizione ricorsiva della soluzione ottima*

- $d_{ij}^{(k)}$ = **il peso del *percorso minimo* tra i vertici *i* e *j* usando solo {1,2,…,*k*} come *possibili vertici intermedi*.**
- **Vogliamo calcolare $d_{ij}^{(n)}$ per tutte le coppie *i,j*.**
- **Supponiamo di avere a disposizione la matrice $d_{ij}^{(k-1)}$ dei *percorsi minimi che contengono solo vertici* in {1,…,*k*-1}. Allora il *percorso minimo* di tra *i* e *j*,**
- **o non contiene il vertice intermedio *k*, e ha quindi peso $d_{ij}^{(k-1)}$,**
- **o contiene *k* e sarà perciò composto dalla *concatenazione* del *percorso minimo* tra *i* e *k* e quello tra *k* e *j*.**

Il *peso* sarà quindi $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$.

❷ *Definizione ricorsiva della soluzione ottima*

- $d_{ij}^{(k)}$ = **il peso del *percorso minimo* tra i vertici v_i e v_j usando solo $v_1, v_2, \ldots, v_k$ come *possibili vertici intermedi*.**
- **Vogliamo calcolare $d_{ij}^{(n)}$ per tutte le coppie i,j.**

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{se } k = 0 \\ \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) & \text{se } k \geq 1 \end{cases}$$

Notate che ora l'indice *k* delle matrici è anche l'indice del *vertice intermedio*

❷ *Definizione ricorsiva della soluzione ottima*

- **Come si *calcolano* i *costi effettivi* dei *percorsi minimi*?**
- **Quante matrici $D^{(k)} = (d_{ij}^{(k)})$ dobbiamo calcolare?**
- ***Per quale valore* di k *otteniamo* in $D^{(k)}$ i *pesi* (costi) dei *percorsi minimi*?**

- ***Se*, come da ipotesi, il grafo *non contiene cicli di peso negativo*,**
- ***allora* ogni *percorso minimo* sarà un *percorso semplice* con *al più vertici intermedi* in $\{1,2,\ldots,n\}$**

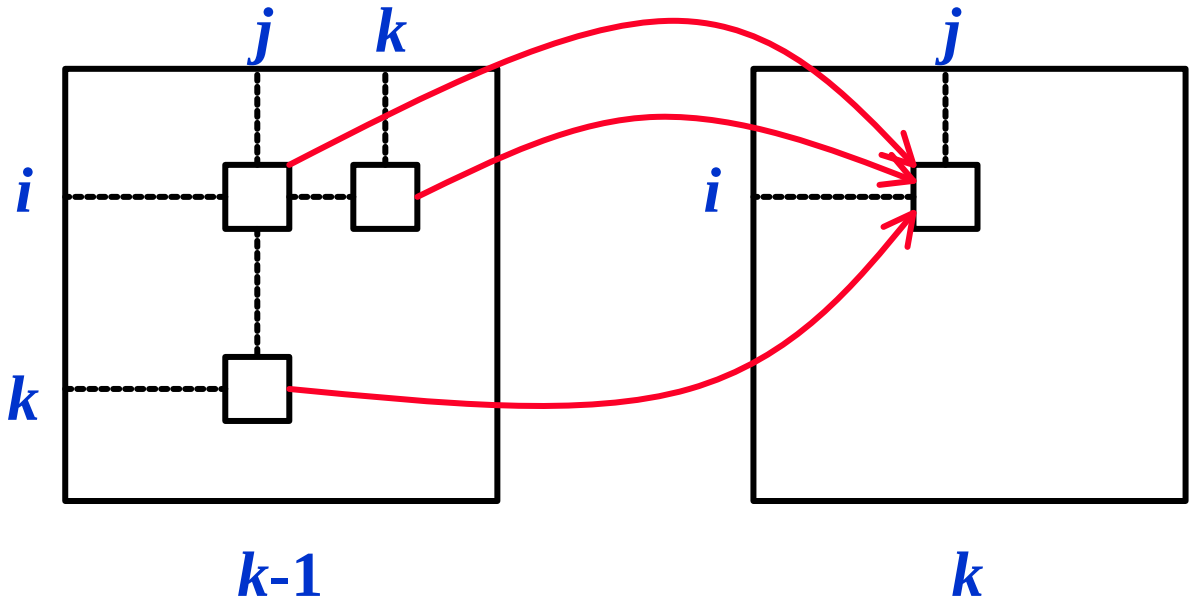
E quindi $d(i,j) = d_{ij}^{(n)} = d_{ij}^{(n+1)} = d_{ij}^{(n+2)} = \ldots$

❸ *Calcolo del valore della soluzione ottima*

- **L'algoritmo riceve in ingresso la matrice $W=(w_{ij})$ di adiacenza del grafo pesato.**
- **Viene quindi calcolata una sequenza di matrici $D^{(0)} = D^{(1)} = D^{(2)} = \ldots = D^{(n)}$, dove per $k=0,1,2,\ldots,n$ abbiamo $D^{(k)} = (d_{ij}^{(k)})$.**
- **La matrice $D^{(n)}$ conterrà i *pesi* dei *percorsi minimi*.**
- **La matrice $D^{(0)} = W$**

❸ Calcolo del valore della soluzione ottima

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right) \qquad \text{per } k \geq 1$$



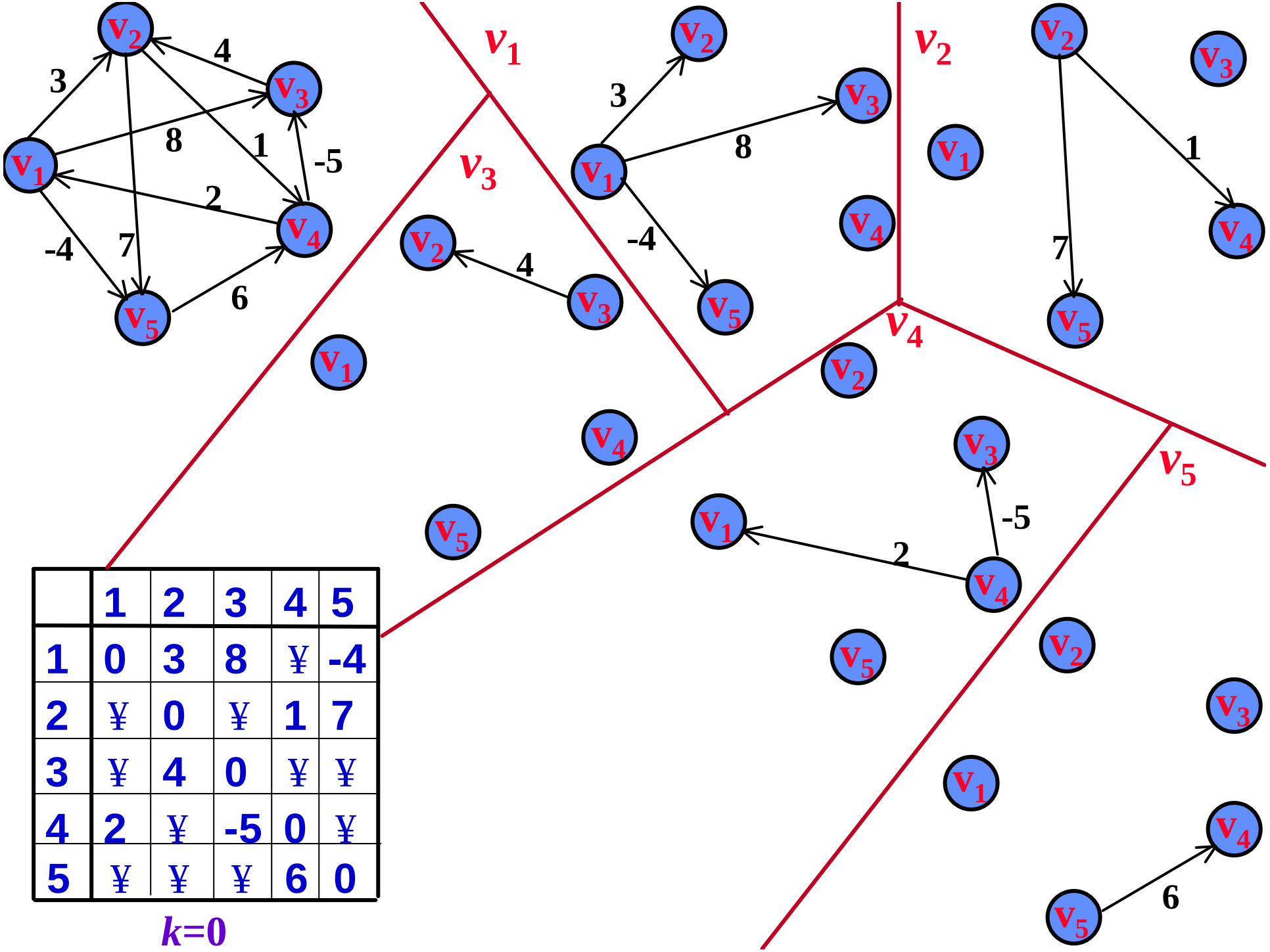
❸ *Calcolo del valore della soluzione ottima*

```
Floyd_Warshall(D,W:matrice)
  n = righe[D]
  D(0) = W
  for k = 1 to n
   do for i = 1 to n
      do for j = 1 to n
         do d(k)ij = min(d(k-1)ij, d(k-1)ik + d(k-1)kj)
  return D(n)
```

$\Theta(n^3)$

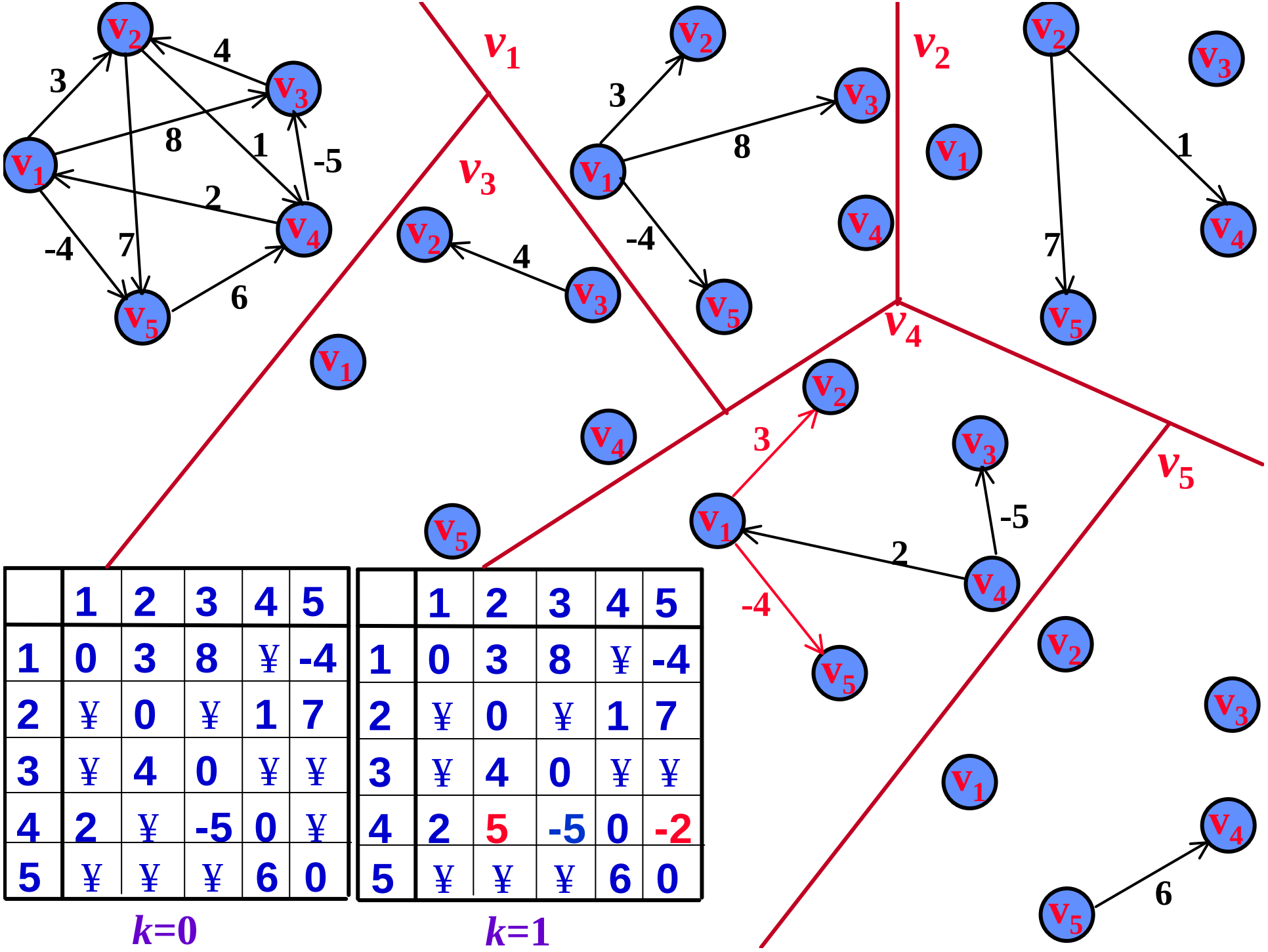
- **Calcola la sequenza di n matrici $D^{(1)}$, $D^{(2)}$,…, $D^{(n)}$ secondo la formula**

$$d_{ij}^{(k)} = \min\left(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\right)$$



	1	2	3	4	5
1	0	3	8	¥	-4
2	¥	0	¥	1	7
3	¥	4	0	¥	¥
4	2	¥	-5	0	¥
5	¥	¥	¥	6	0

k=0

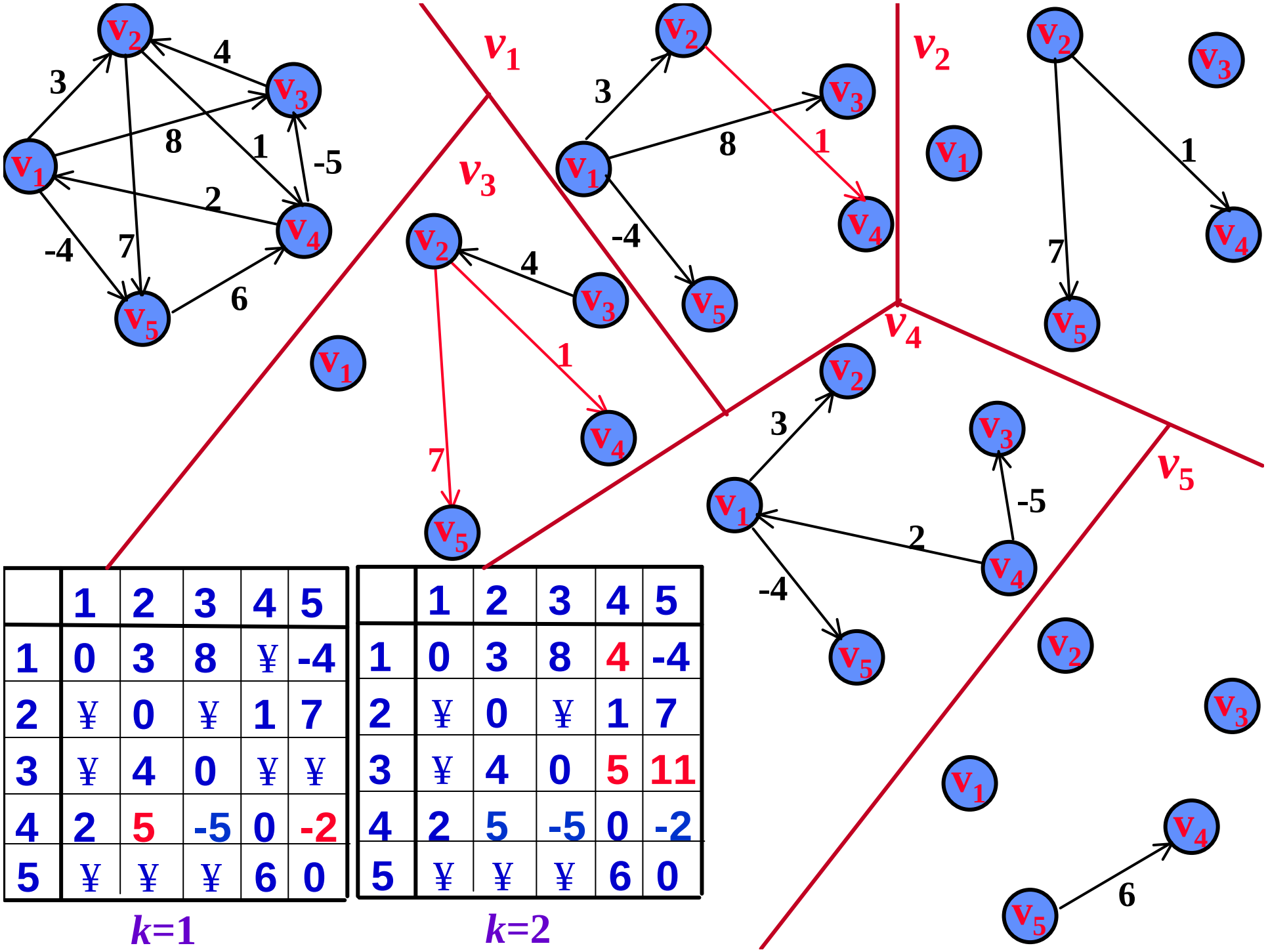


	1	2	3	4	5
1	0	3	8	¥	-4
2	¥	0	¥	1	7
3	¥	4	0	¥	¥
4	2	¥	-5	0	¥
5	¥	¥	¥	6	0

k=0

	1	2	3	4	5
1	0	3	8	¥	-4
2	¥	0	¥	1	7
3	¥	4	0	¥	¥
4	2	5	-5	0	-2
5	¥	¥	¥	6	0

k=1

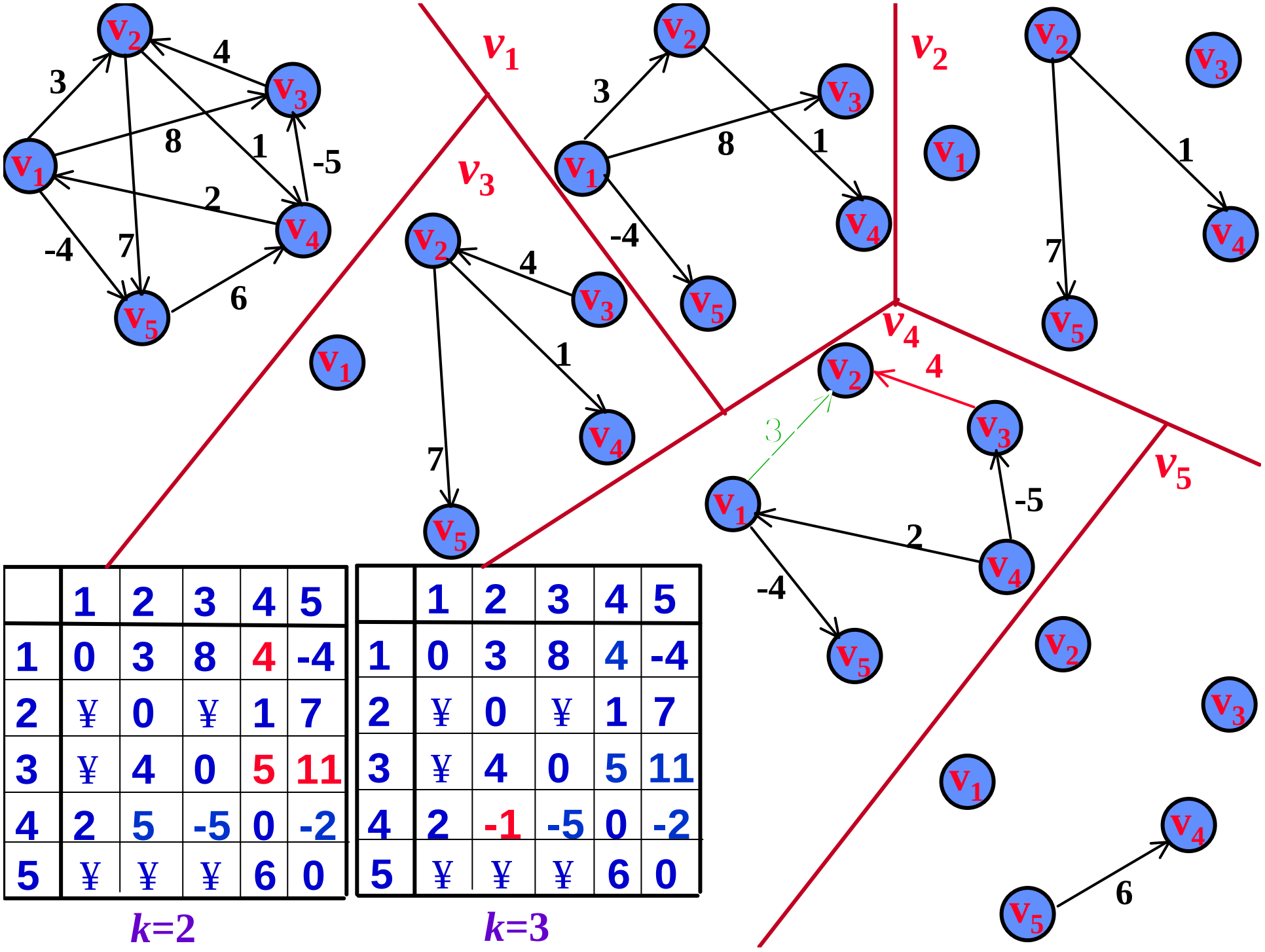


	1	2	3	4	5
1	0	3	8	¥	-4
2	¥	0	¥	1	7
3	¥	4	0	¥	¥
4	2	5	-5	0	-2
5	¥	¥	¥	6	0

k=1

	1	2	3	4	5
1	0	3	8	4	-4
2	¥	0	¥	1	7
3	¥	4	0	5	11
4	2	5	-5	0	-2
5	¥	¥	¥	6	0

k=2

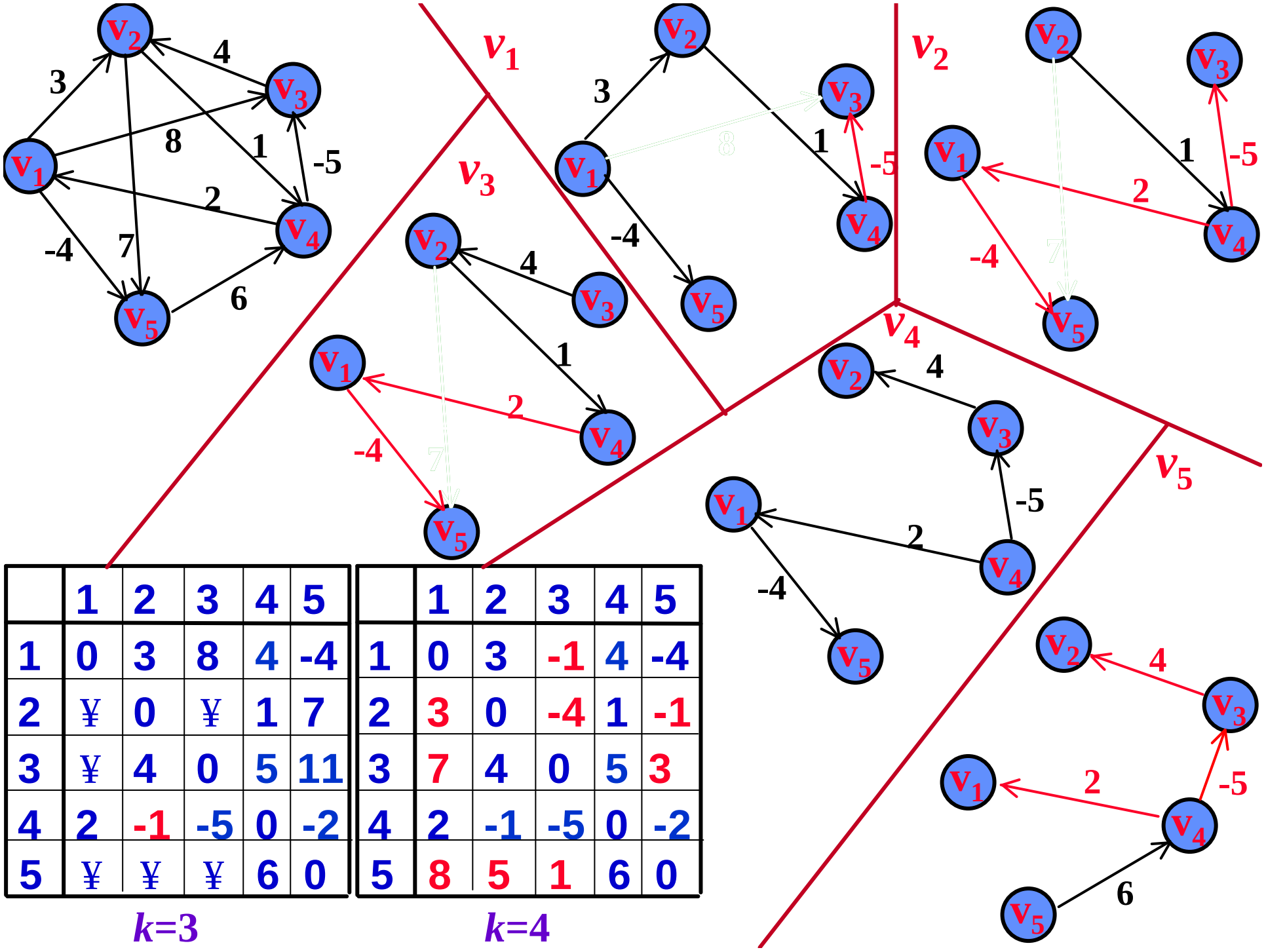


	1	2	3	4	5
1	0	3	8	4	-4
2	¥	0	¥	1	7
3	¥	4	0	5	11
4	2	5	-5	0	-2
5	¥	¥	¥	6	0

k=2

	1	2	3	4	5
1	0	3	8	4	-4
2	¥	0	¥	1	7
3	¥	4	0	5	11
4	2	-1	-5	0	-2
5	¥	¥	¥	6	0

k=3

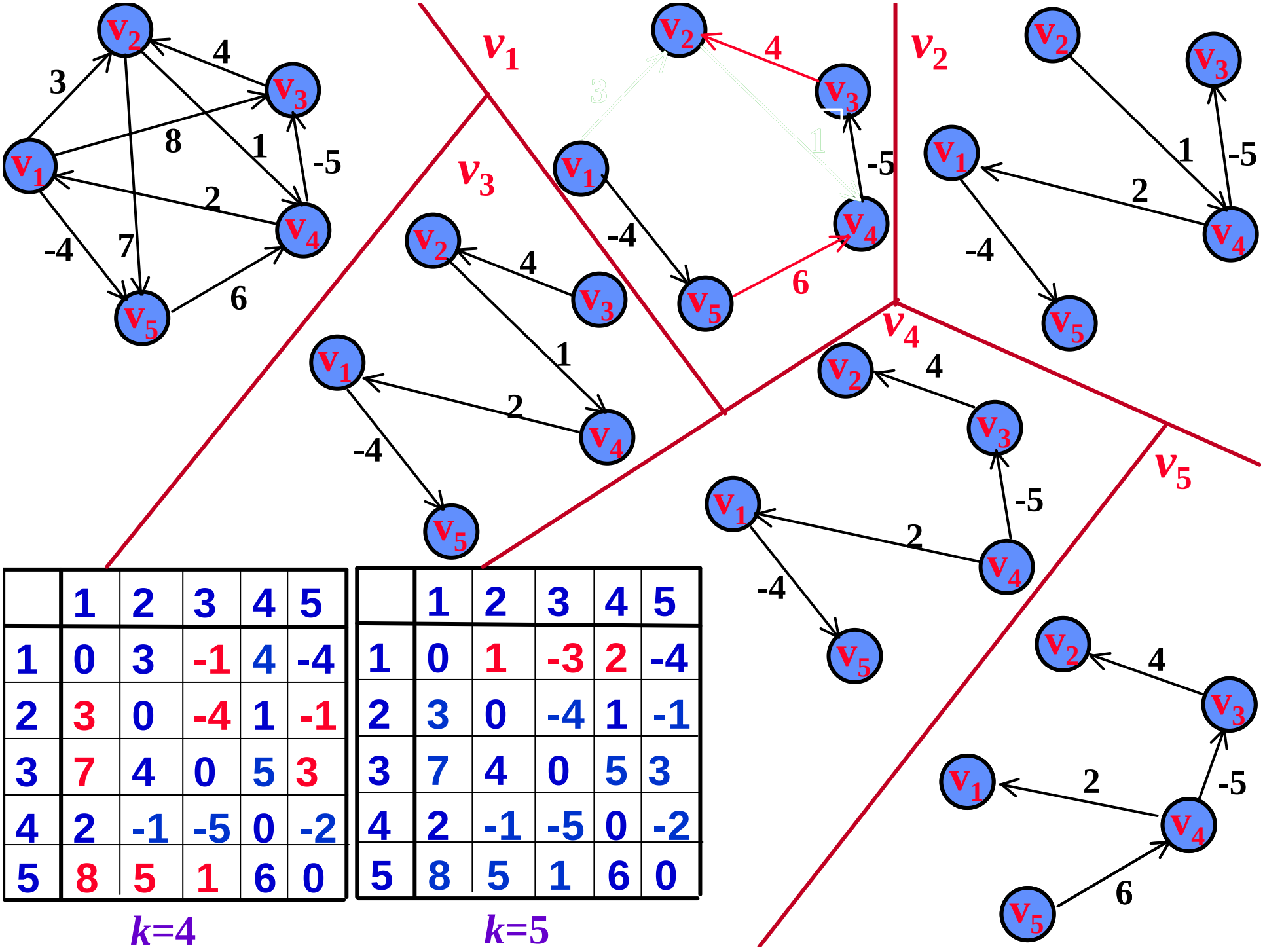


	1	2	3	4	5
1	0	3	8	4	-4
2	¥	0	¥	1	7
3	¥	4	0	5	11
4	2	-1	-5	0	-2
5	¥	¥	¥	6	0

k=3

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

k=4



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

k=4

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

k=5

❸ *Calcolo del valore della soluzione ottima*

- **Una volta che abbiamo calcolato la matrice delle *distanze minime* $D = D^{(n)}$, come possiamo *ricostruire i percorsi minimi* ?**
- **In altre parole come possiamo calcolare la *matrice dei predecessori* $P = P^{(n)}$?**
- **Si possono usare essenzialmente due metodi:**
  - **Calcolare una sequenza di *matrici dei predecessori* $P^{(k)}$ per $k=1,\ldots,n$ mentre calcoliamo $D^{(k)}$**
  - **Calcolare P direttamente da D (*Esercizio 26-1.5*)**