Grafi pesati $w: E \to \mathbb{R}^+_{\neq 0}$

Cammini minimi da sorgente singola s

Q: coda di priorità rispetto le etichette $d(v): \forall v \in V$

$d(u) = \delta(s, u)$ quando u è estratto dalla coda di priorità

Nuova operazione per ri-etichettare

$s \rightsquigarrow \overset{u}{\square} \longrightarrow \overset{v}{\square}$

$\underbrace{d(u) + w(u,v)}$

Relax $(u, v, w(u,v))$

if $\boxed{d(v)} > \boxed{d(u) + w(u,v)}$
then

$$\begin{cases} d(v) \leftarrow d(u) + w(u,v) \\ \pi(v) \leftarrow u \end{cases}$$

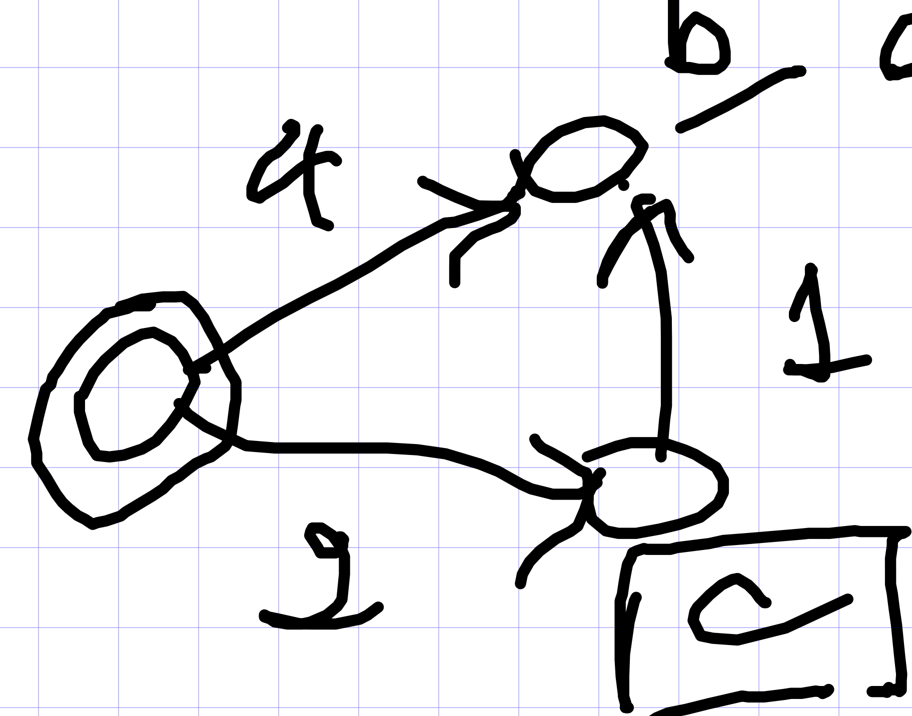
Soura : a

$w : E \to \mathbb{R}^+$

$d(a) : \underbrace{\delta(a, a)} = \emptyset$

a

$d(a) = \emptyset$



$\cancel{d(b) = +\infty}$

$d(b) = d(a) + w(a, b)$

$\cancel{= 4}$

$= 3$

$\pi(b) = c$

$d(c) = +\infty$

$d(c) = 2$ $\quad \pi(c) = a$

a b $\qquad$ a c b $\qquad$ =

$d(b) = 4 > d(c) + w(c, b)$

$\qquad\qquad\quad 2 + 1$

$$\begin{cases} d(b) = 3 \\ \pi(b) = c \end{cases}$$

Dijkstra $(G; w: E \to R^+; s)$

{ $\forall v \in V$: color(v) = white

$d(v) = +\infty$; $\pi(v) \leftarrow nil$

$d(s) \leftarrow 0$; $Sol \leftarrow \emptyset$

$Q \leftarrow$ MAKEQUEUE $(d(v) : \forall v \in V)$

while Q is not empty do

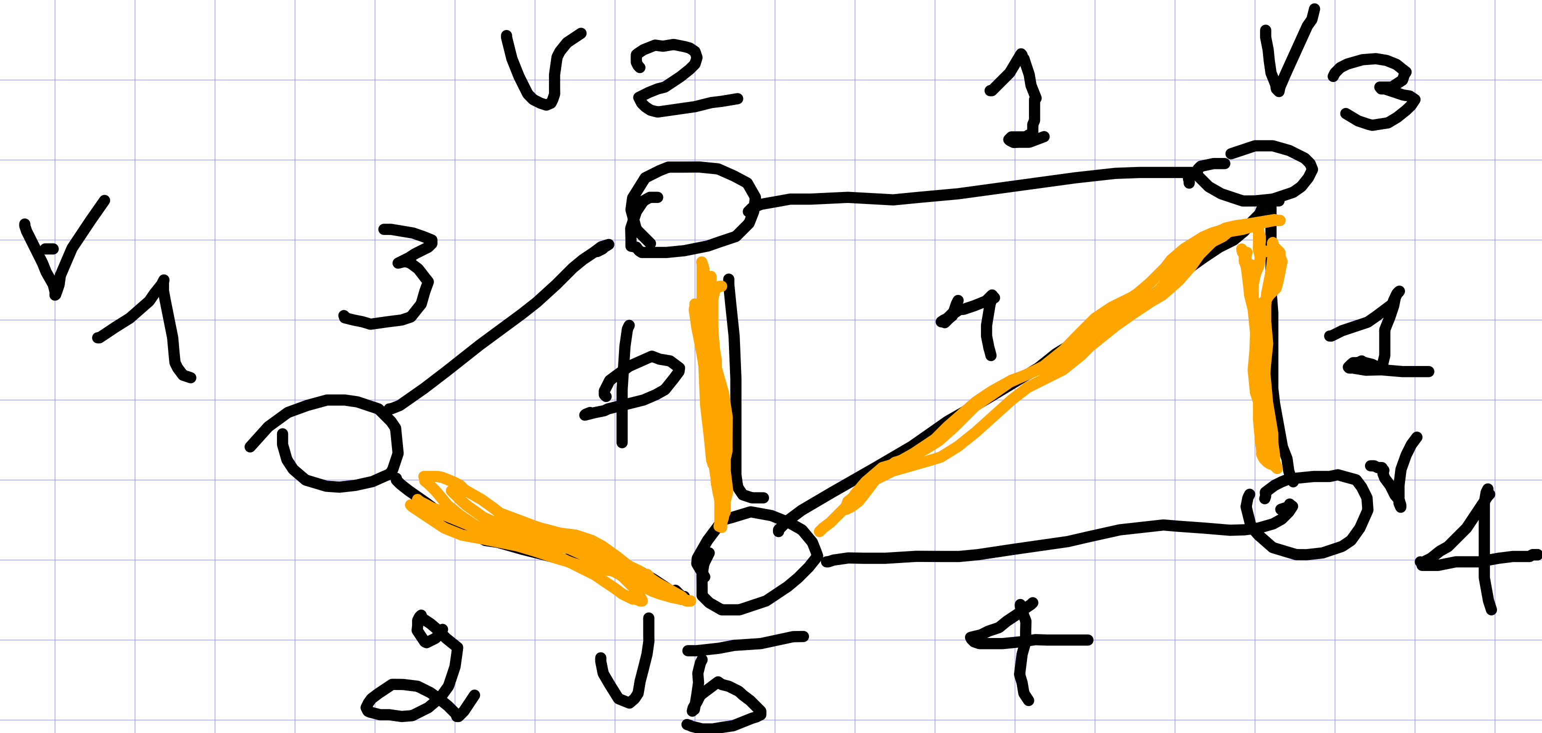
  $u \leftarrow$ extract.min (Q)

  $\forall v \in Adj(u)$ and $v \notin Sol$

    RELAX $(u, v, w(u, v))$

  $Sol \leftarrow Sol \cup \{u\}$

}



$S = v_1$

$Sol \leftarrow \{v_1, v_5, v_2, v_3, v_4\}$

π	$\varnothing$	v_5	v_5	v_3	v_1
d	$\varnothing$	2	3	4	2
	v_1	v_2	v_3	v_4	v_5

Q	$\cancel{\varnothing}$	$\cancel{2}$	$\cancel{3}$	$\cancel{4}$	$\cancel{2}$
	$\cancel{v_1}$	$\cancel{v_2}$	$\cancel{v_3}$	$\cancel{v_4}$	$\cancel{v_5}$

Relax$(u, v, w(u,v))$ $\quad \Theta(1)$

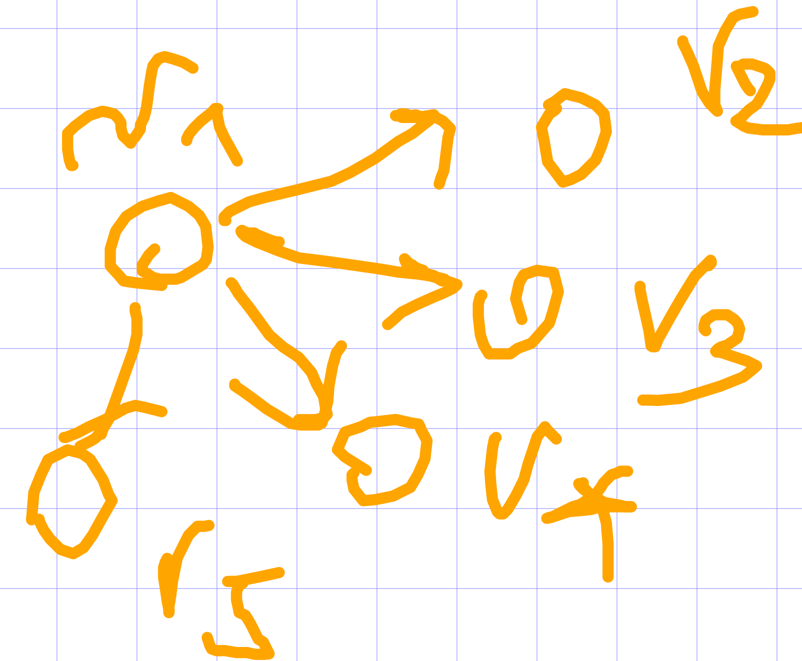
$d(v) > d(u) + w(u,v)$

$\quad d(v) \leftarrow d(u) + w(u,v)$

$\quad \pi(v) \leftarrow u$

Complessità

$|V| \cdot T_{extract-min}$
$+ |E| \cdot T_{RELAX}$



Coda Q realizzata da un vettore

extract-min $O(|Q|) = O(|V|)$

Relax : accesso diretto a Q usando l'indice del vertice $O(1)$

Complessità con coda di priorità realizzato con vettore

$$|V| \cdot T_{extract\text{-}min} + |E|\, T_{relax}$$

$$= |V| \cdot O(|V|) + |E|\, O(1)$$

$$= O(V^2) + O(|E|) = \boxed{O(V^2)}$$

M. adiacenza $E = V^2$

L. A $E =$ numero archi nel grafo

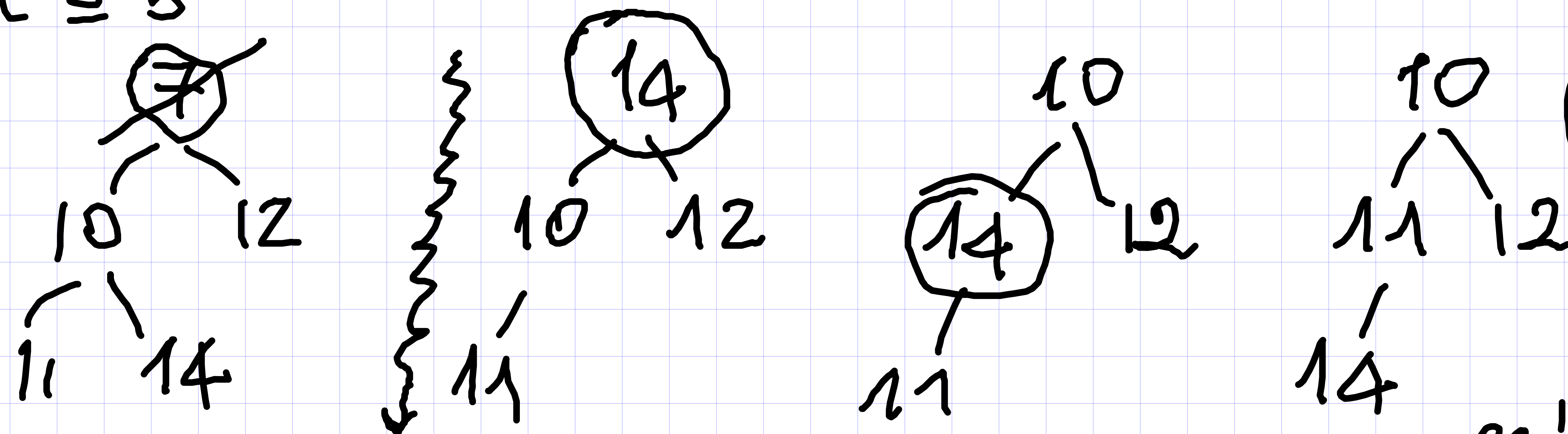
$\Rightarrow$ struttura non influisce

- extract-min = return H(1)
  Heapify(1, n-1)

- RELAX = decrease-key
  → le chiave si muove verso la radice

Esempio extract-min di heap binario

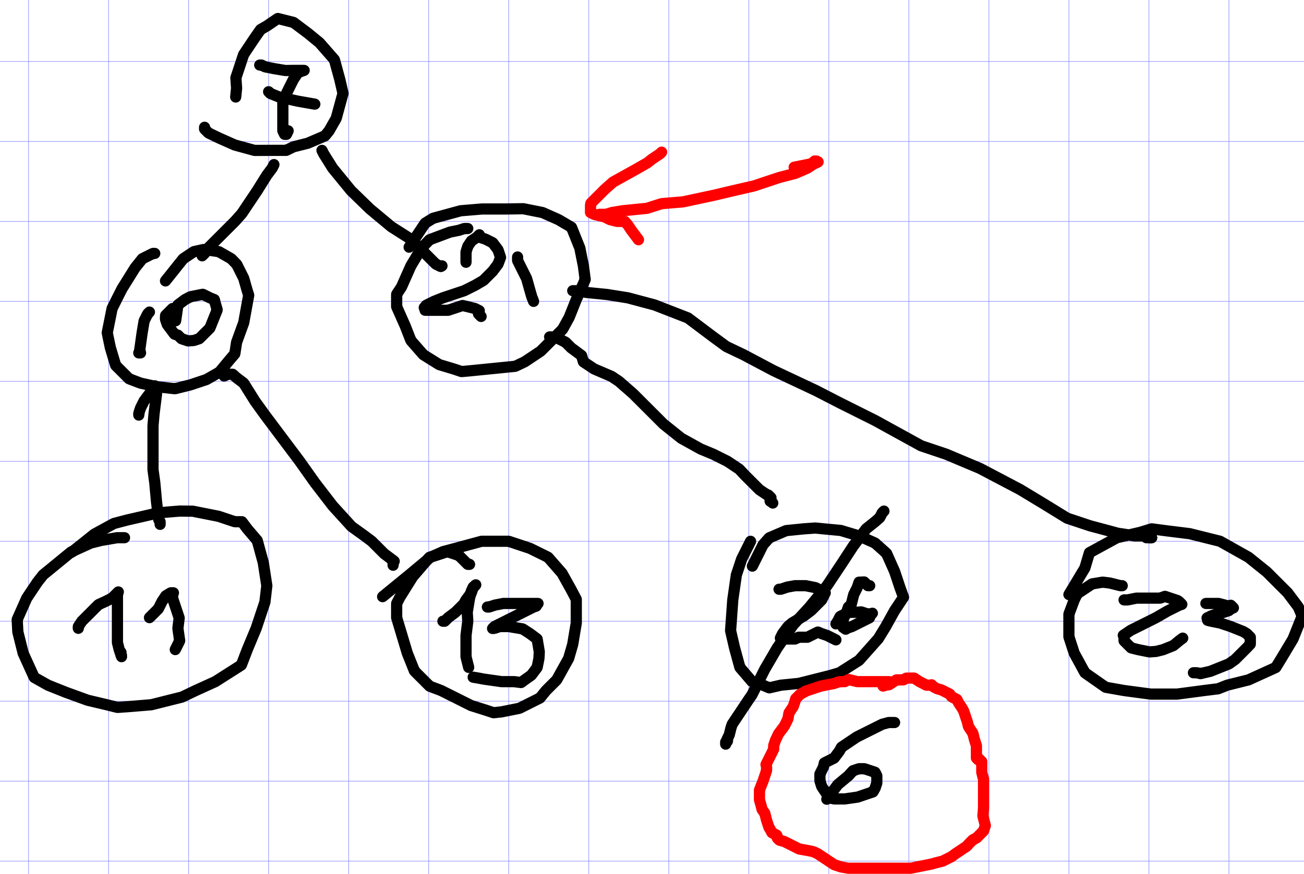
$n = 5$

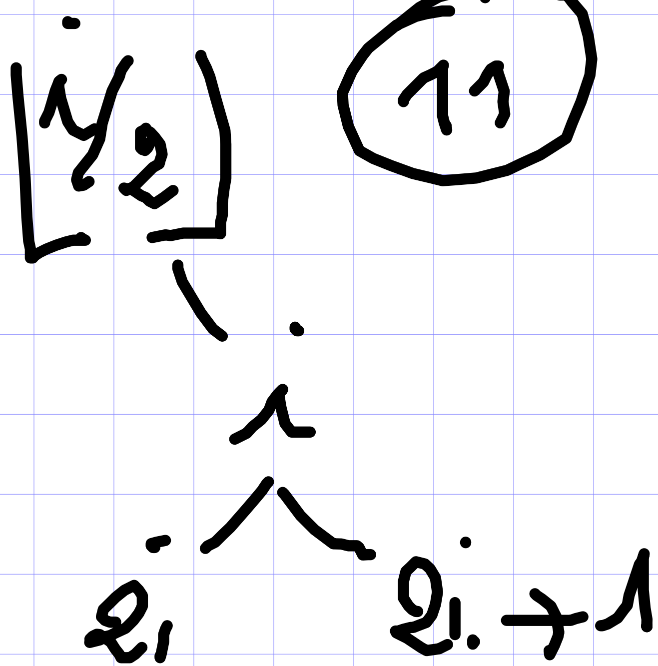


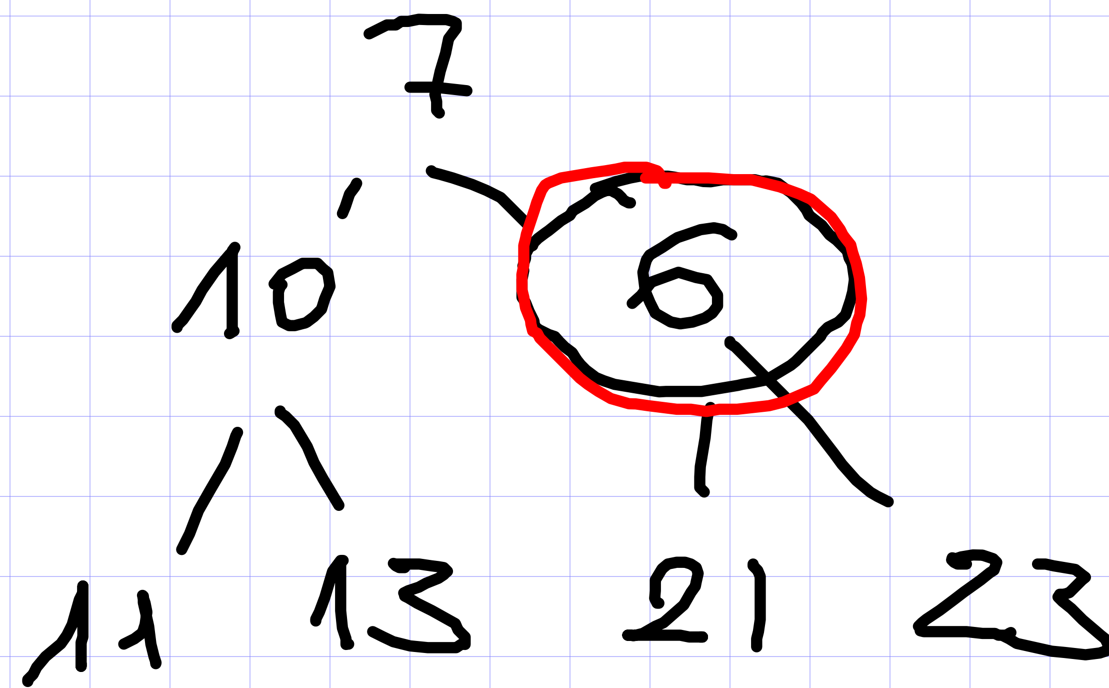
$\lceil \log_2 n \rceil$

$n' = 5 - 1 = 4$

Esempio decrease-key in min-heap







	1	2	3	4	5	6	7
H	6	10	7	11	13	21	23

// key ≤ H[i] //

Decrease-key (H, i, key)

{ H[i] ← key;

while (i > 1) and (H[i] < H[⌊i/2⌋])

do { scambia (H[i], H[⌊i/2⌋]) }

}

$T_{RELAX} = O(\log n) \quad n = |V|$

Dijkstra $(G; w: E \to R^+; s)$

$\forall v \in V$: color$(v) = W$; $d(v) = +\infty$; $\pi(v) \leftarrow$ nil

$d(s) \leftarrow 0$

$H \leftarrow$ MAKEHEAP$(d(v) : \forall v \in V)$

while H is not empty do

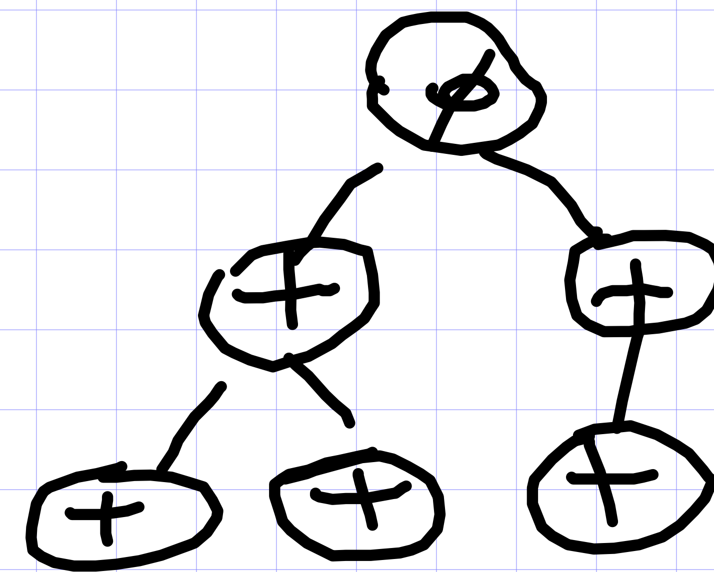
- $u \leftarrow$ extract.min(H)
- $\forall v \in Adj(u)$ and $v \notin Sol$
  - HEAP-RELAX$(u, v, w(u, v))$
- $Sol \leftarrow Sol \cup \{u\}$

MAKE HEAP $(d(v) : \forall v \in V)$

H

$\emptyset$	+	+	+	+	+

1



```
Decrease-key (H, i, key)
{ H[i] ← key;
  while (i > 1) and (H[i] < H[⌊i/2⌋])
    do { scambia (H[i], H[⌊i/2⌋]) }
}
```

(HEAP)-RELAX $(u, v, w(u,v))$

if $d(v) > d(u) + w(u,v)$

then

→ $d(v) \leftarrow d(u) + w(u,v)$

$\pi(v) \leftarrow u$

DECREASE-KEY$(\mathcal{H}, pos[v], d(v))$

//mantieni le etichette all'interno di un min-heap//

Complessità

$|V| \, T_{\text{extract-min}}$

$|E| \, T_{\text{RELAX}}$

$(|V| + |E|) \, (\log_2 |V|)$

code realizzata con heap binario

$$d(v) \leftarrow \underbrace{d(u) + w(u,v)}$$

$$\text{DECREASE-key}\,(H, \underbrace{pos(v)}_{\substack{\text{nodo}\\ = \\ i}}, \underbrace{d(v)}_{\text{key}})$$

Serve mantenere una corrispondenza biunivoca tra i vertici (del grafo) e la loro posizione in H

Quanto segue è l'implementazione di Dijkstra con heap.
E' necessario stabilire una corrispondenza biunivoca fra i vertici del grafo e i nodi dello heap.

$\forall v \in V$

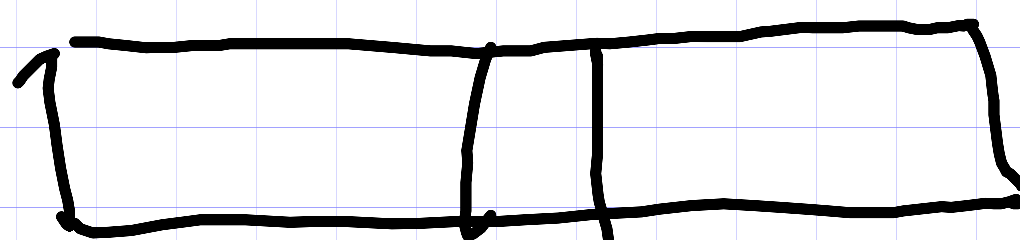
H memorizza in posizione i

nodo dello heap

v	$d(v)$	$\pi(v)$

chiave di priorità

Pos: vettore per trovare la posizione di v in H in $O(1)$

Pos 

$Pos(v)$ = posizione di $d(v)$ relativa al nodo v in H

vertice del grafo

Esempio

$Q \equiv H$

$d(v_1)$

v_1	7	v_0

v_3	10	v_1

$d(v_3)$

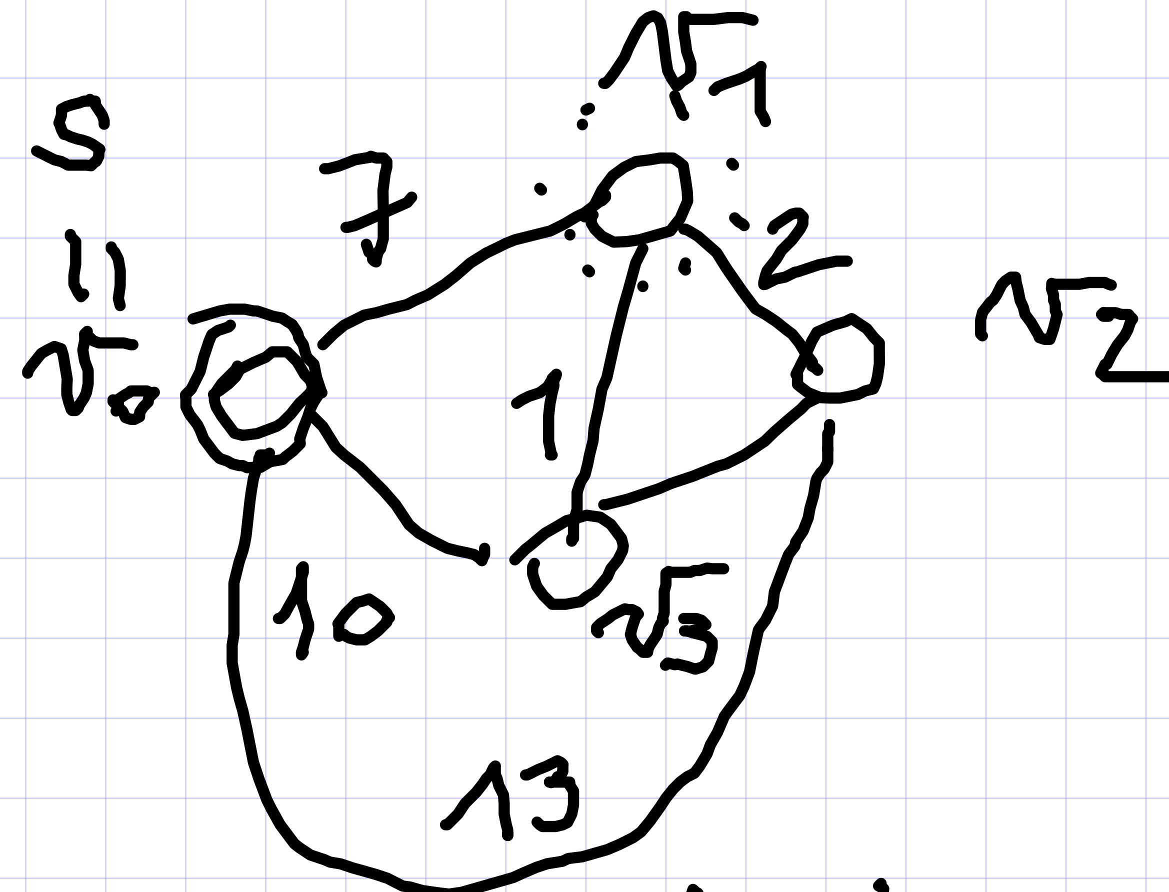
v_2	13	v_0

$d(v_2)$

H

v_1	7	v_0	v_3	10	v_0	v_2	13	v_0
1			2			3		

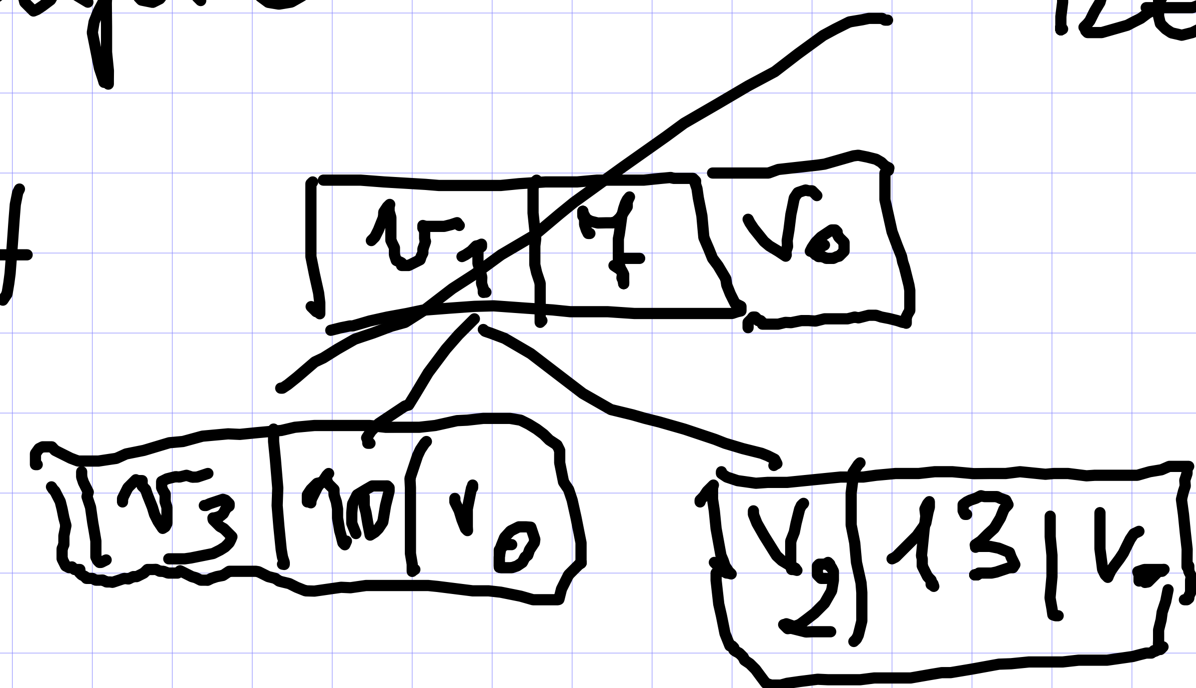
Sol $= \{ v_0$



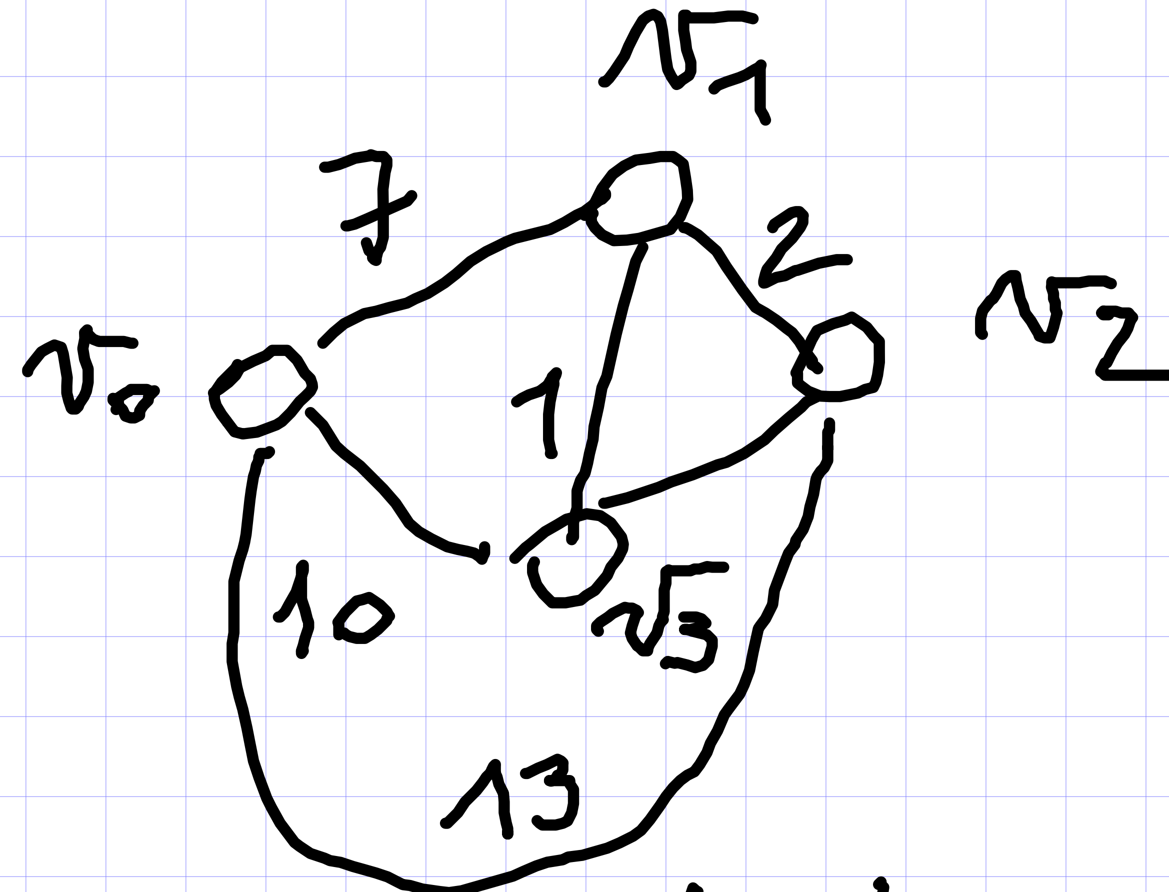
	v_0	v_1	v_2	v_3
d	0	7	13	10
π	nie	v_0	v_0	v_0
pos	–	1	3	2

Esempio

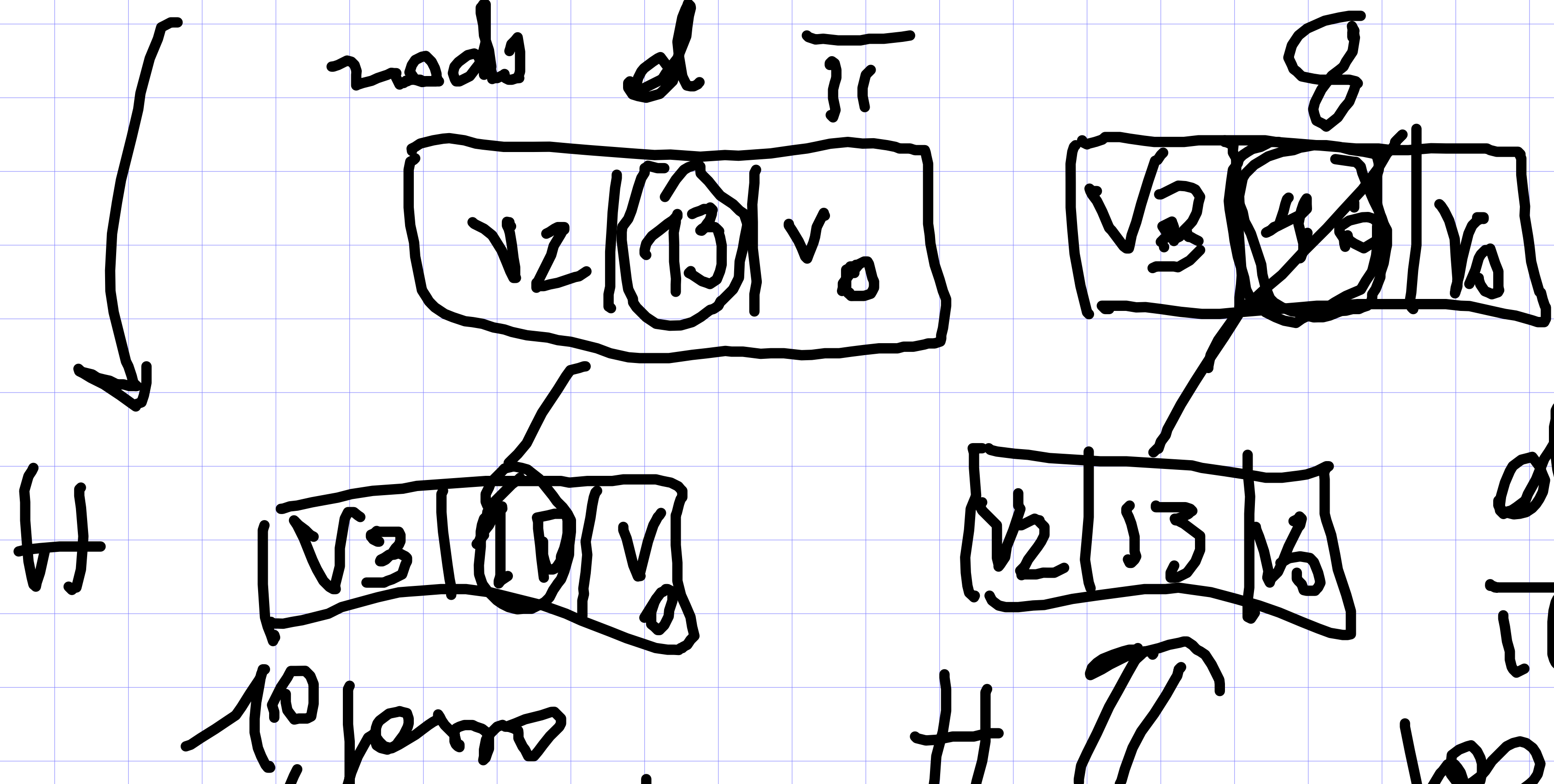
$Q \equiv H$



$u \leftarrow v_1$ (extract-min)
RELAX $(v_1, v_3, 1)$



nodo d π



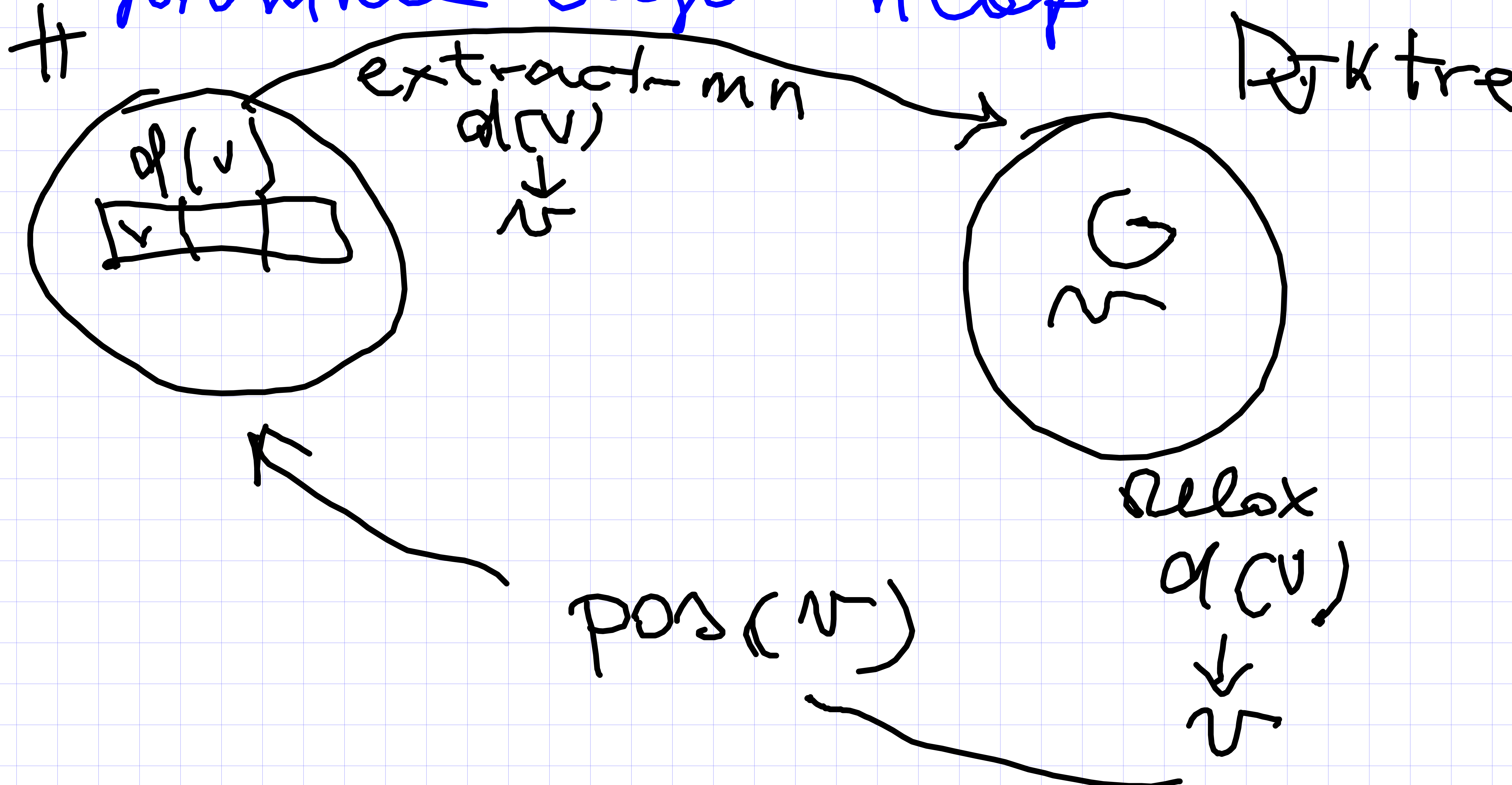
1° passo extract-min

	v_0	v_1	v_2	v_3	
d	0	7	13	~~10~~ 8	8
π	nie	v_0	v_0	v_0	v_1
pos	-	-	2	1	x

Implementazione con mapping grafo – heap

```
Decrease-key(H, t, key)
    H[t].d ← key;
{ while (t > 1) and
        (H[t].d < H[⌊t/2⌋].d) do
    { temp ← H[t]
      H[t] ← H[⌊t/2⌋];  pos(H[t].nodo) ← t;
      H[⌊t/2⌋] ← temp;  pos(H[⌊t/2⌋].nodo) ← ⌊t/2⌋
      t ← ⌊t/2⌋
    }
```

Ilustrare corespondența dintre Graf - Heap



Dijkstra $(G; w: E \to R^+; s)$

$\forall v \in V$: color $(v) = W$; $d(v) = +\infty$; $\pi(v) \leftarrow nil$

$d(s) \leftarrow 0$

$H \leftarrow$ MAKEHEAP $(d(v) : \forall v \in V)$

while H is not empty do

$u \leftarrow$ extract.min (H)

$\forall v \in Adj(u)$ and $v \notin Sol$

RELAX $(u, v, w(u, v))$

$Sol \leftarrow Sol \cup \{u\}$

Complessità con heap binario

$$V \cdot T_{\text{extract-min}}$$

$$+ \; E \cdot T_{\text{decrease-key}}$$

$$= O\left[(V+E)\log|V|\right]$$

Operazioni coinvolte:

Esplorazione del grafo + $|V|$ extract-min + $|E|$ relax archi esistenti

Operazioni coinvolte:

Esplorazione del grafo +
$|V|$ extract-min + $|E|$ relax
archi esistenti

Rappresentazione con L.A.

Tempo alg di Dijkstra ->

$$O(V+E) + O\big((|V|+|E|)(\log_2|V|)\big)$$

$$= O\big((V+E)(\log_2 V)\big)$$

Rappresentazione con M. A. $\longrightarrow$

Tempo dell'alg Dijkstra

$$O\Big[\underbrace{(V^2)}_{\text{esplorazione}} + \underbrace{(V + E_n)(\log_2 V)}_{T_{\text{extract-min}} + T_{\text{relax}}}\Big]$$

$$= O\left[V^2 + E \log_2 V\right]$$

L.A.f Dijkstra con heap →

$$(V+E)(\log V) = E \log V \quad (1)$$

grafo connesso $|E| \geq |V|$

Dijkstra con vettore

$$(V^2 + E) = V^2 \quad (2)$$

Se $|E| \in O\left(\frac{V^2}{\log V}\right)$ → (1) batte (2)

Se $|E| \in \Theta(V^2)$ → (2) batte (1)

M.A + Dijkstra con heap

$$O\left[V^2 + E \log_2 V\right] \qquad (1)$$

Dijkstra con vettore

$$O\left(V^2\right) \qquad (2)$$

(2) batte (1) sempre

Per l'algoritmo di Dijkstra per il calcolo dei cammini minimi da sorgente singola è conveniente usare lo heap binario se il grafo è rappresentato con la lista delle adiacenze e $|E| \in O\left(\frac{V^2}{\log |V|}\right)$

CURIOSITÀ:

Alg Dijkstra può migliorare la sua complessità rappresentando il grafo con liste delle Adiacenze e heap di fibonacci (struttura ammortizzata).

Textract. min $O(\log V)$

Trelox $O(1)$

Complessità

$O(V \log V + E)$

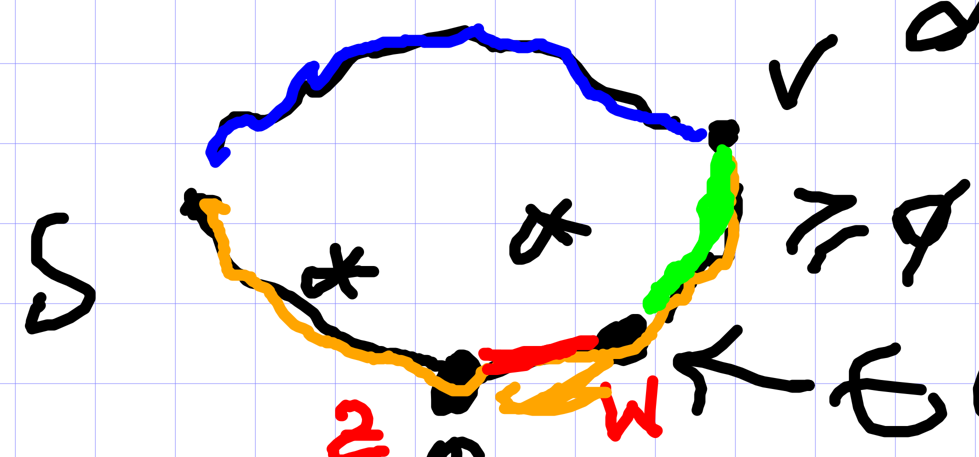
Correttezza dell'alg. di Dijkstra

$\forall v \in V$: $d(v) = \delta(s,v)$ quando $v \in Sol$.

$d(s) = \delta(s,s) = \phi$ base induzione

$\exists \bar{v}$: primo nodo per cui $d(\bar{v}) \neq \delta(s,\bar{v})$

sol Dijkstra $d(\bar{v})$



$d(w) \geq d(\bar{v})$ ← minimo delle code

perché estratto $\bar{v}$

→ cammino minimo costo più $d(\bar{v})$

← $\in Q$ quando estraggo $\bar{v}$

$\Rightarrow \nexists \bar{v}$

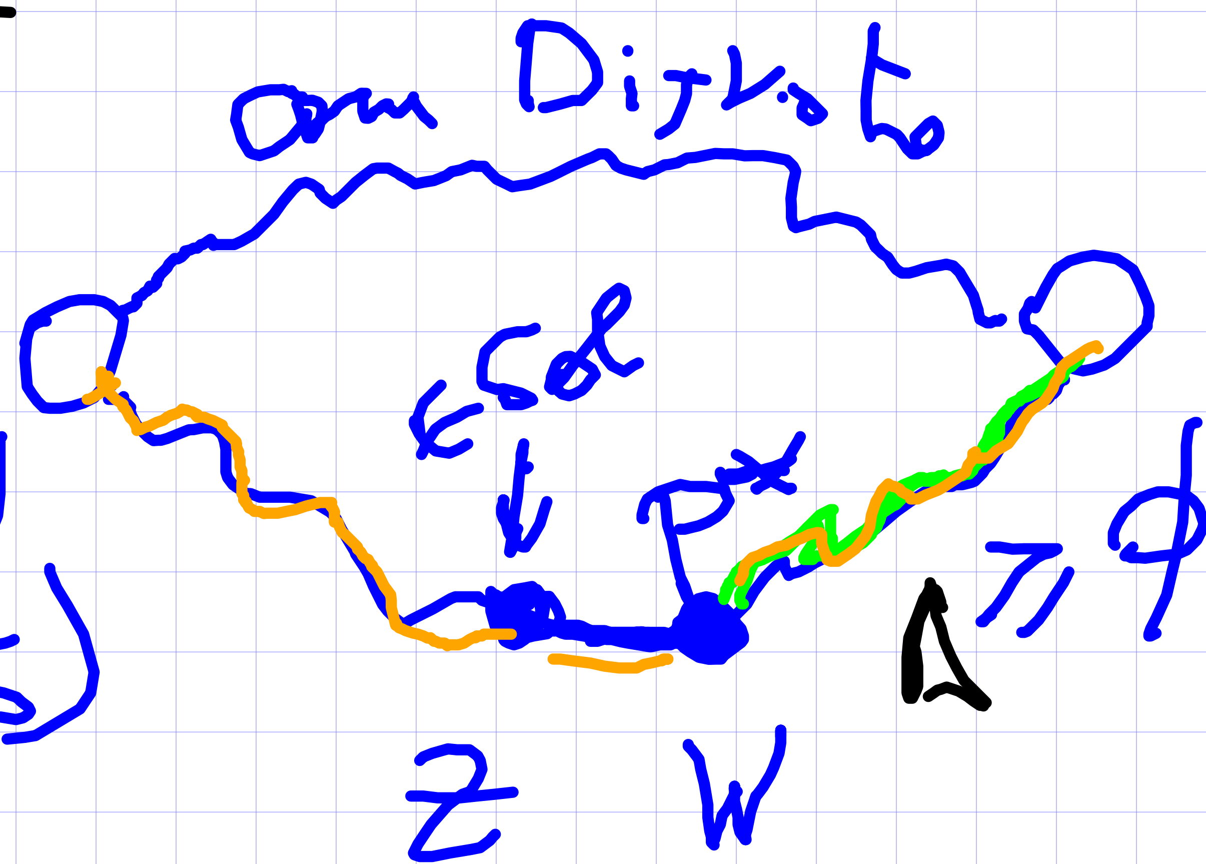
↑ ultimo nodo $\in Sol$ ($\exists$ sempre perché $s \in Sol$)

Illustrazione prova di correttezza

$\bar{v} \leftarrow$ extract-min

$d(\bar{v}) \leq d(w)$

$\bar{v} \quad d(\bar{v}) \neq \delta(s, \bar{v})$



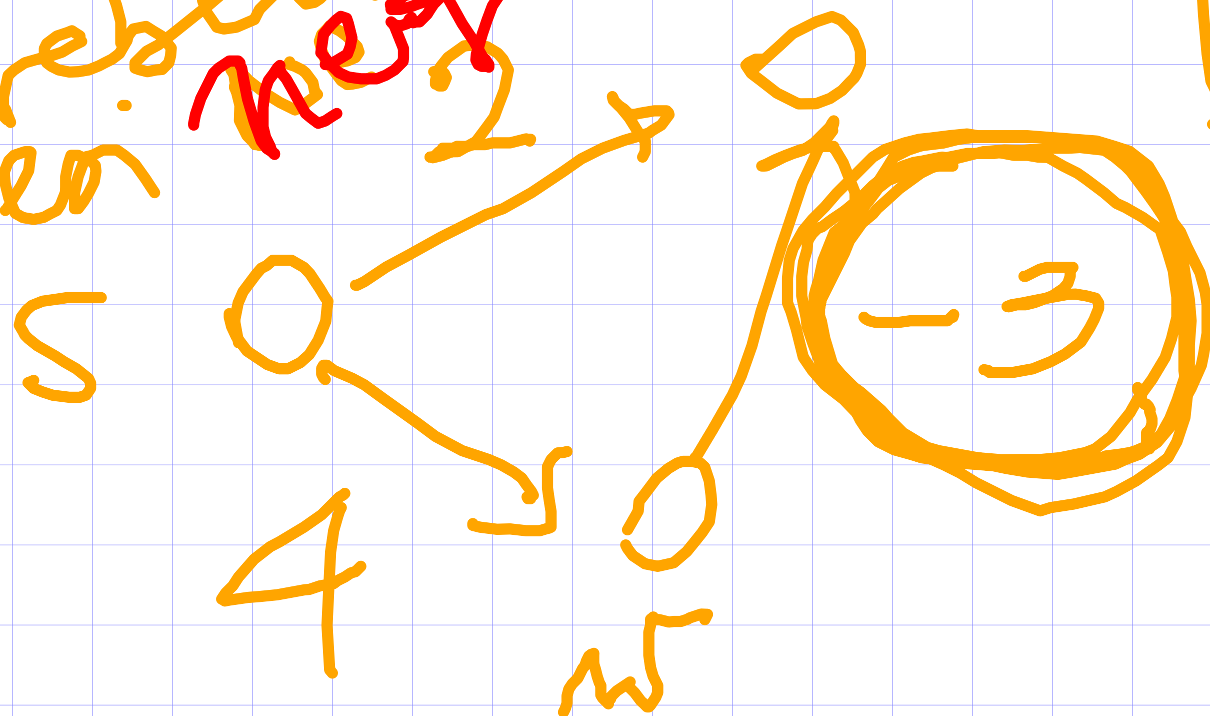
$d(s) = \delta(s,s)$

$d(w) = d(z) + \omega(w,z)$

$\text{costo}(P^*) = d(w) + \Delta \geq d(\bar{v})$

$\Delta \geq 0$

DiJkstra non funziona in presenza di pesi negativi



$\boxed{d(\underline{v}) = 2}$

sbaglia!

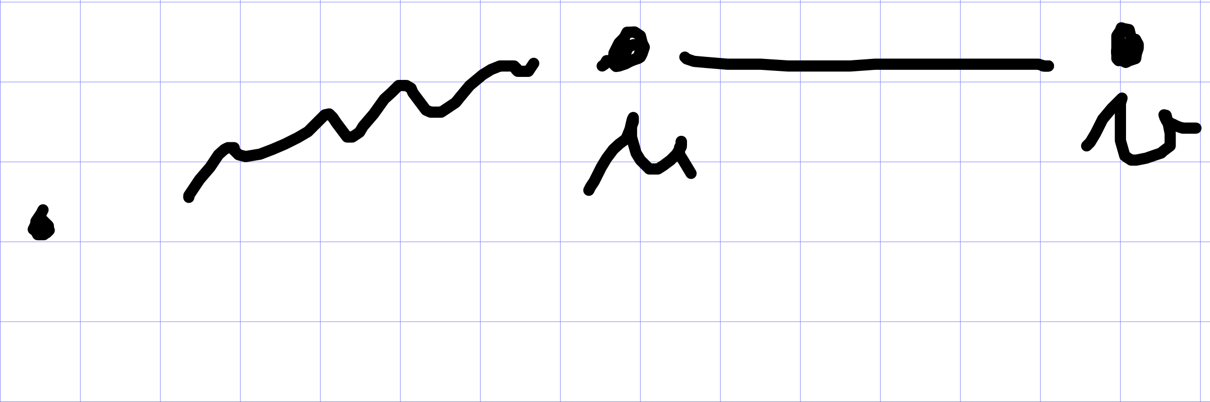
perché il costo è negativo

$d(w) = 4$

$d(w) + \underset{<0}{\underline{\Delta}} \geq d(v)$

OBS $s \overset{\pi}{\rightsquigarrow} u \to v$

$$\text{Se } \boxed{(u,v)} \in \delta(s,v)$$

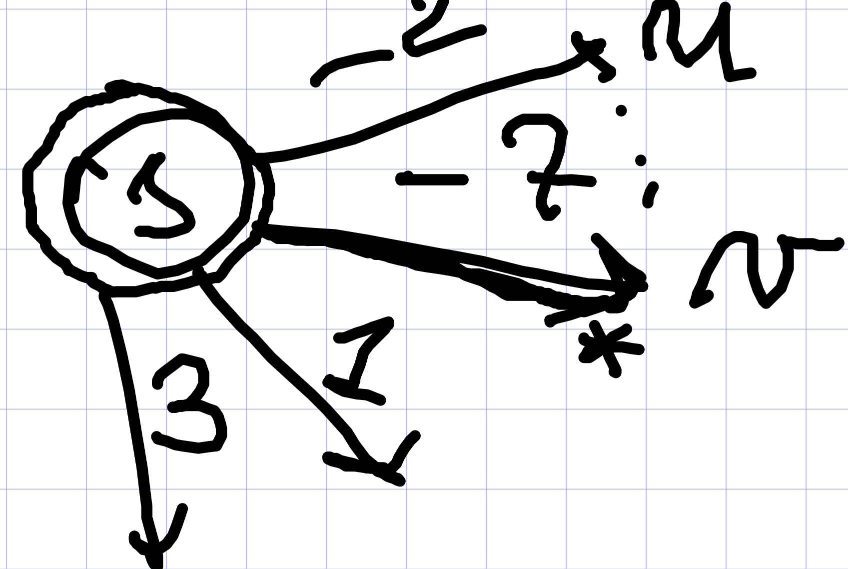


quando estraggo u e relax (u,v) $d(v)$ nella coda ha il valore ottimo $\delta(s,v)$

Dijkstra può tollerare archi di costo negativo **uscenti** dalla **sorgente**

$w: E \to \mathbb{R}^+$

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



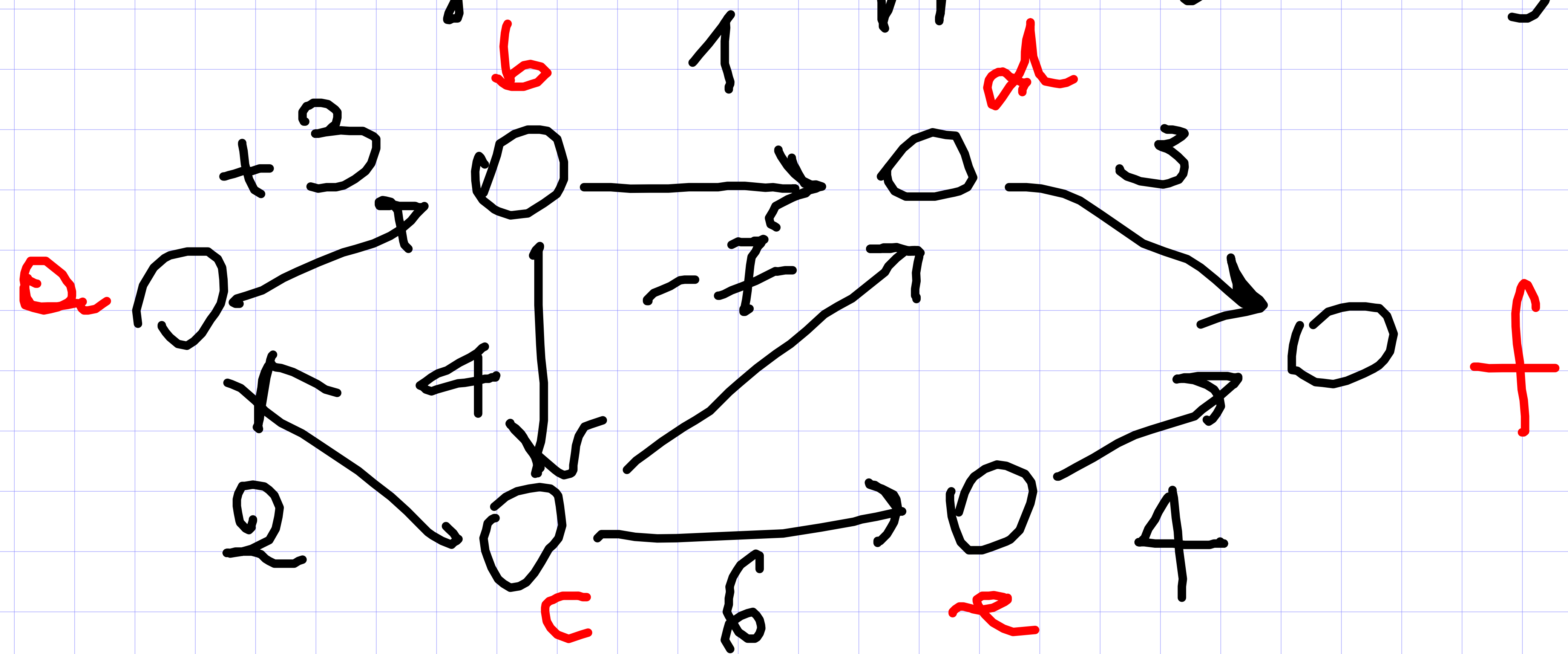
$d(v) = -7 = \delta(s,v)$

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

$\exists$ sicuramente un cammino minimo che usa arco uscente da s, sia (s,v)

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

Esempio in cui posso applicare Dijkstra



$G = (V, E)$ $\quad w : E \to \mathbb{R}$

Dijkstra	
S	No
a	
e	Sì

Se avessi $w(e,f) = -4$ non posso mai applicare Dijkstra

Cammini minimi $w : E \to \mathbb{R}$

($\exists$no archi con costo negativo)

$\Rightarrow$ rischio di avere cicli di costo negativo

Bellman Ford $(G, w: E \to \mathbb{R})$: boolean

TRUE se $\exists$ ciclo di costo negativo

FALSE se $\nexists$ ciclo di costo negativo

$\delta(s, v)$ $\forall v \in V$

OBS lunghezza (# archi) cammino minimo $\geq n$

$\Longleftrightarrow$ $\exists$ ciclo costo negativo

In un path

$a - b - c - d$

$\#\text{vertici} = \#\text{archi} + 1$

$$\downarrow$$

$\#\text{vertici} = n - 1 + 1$

$\#\text{vertici} = n + 1 \longrightarrow$ ripetizione tra i vertici

$\exists$ ciclo $\longleftarrow$

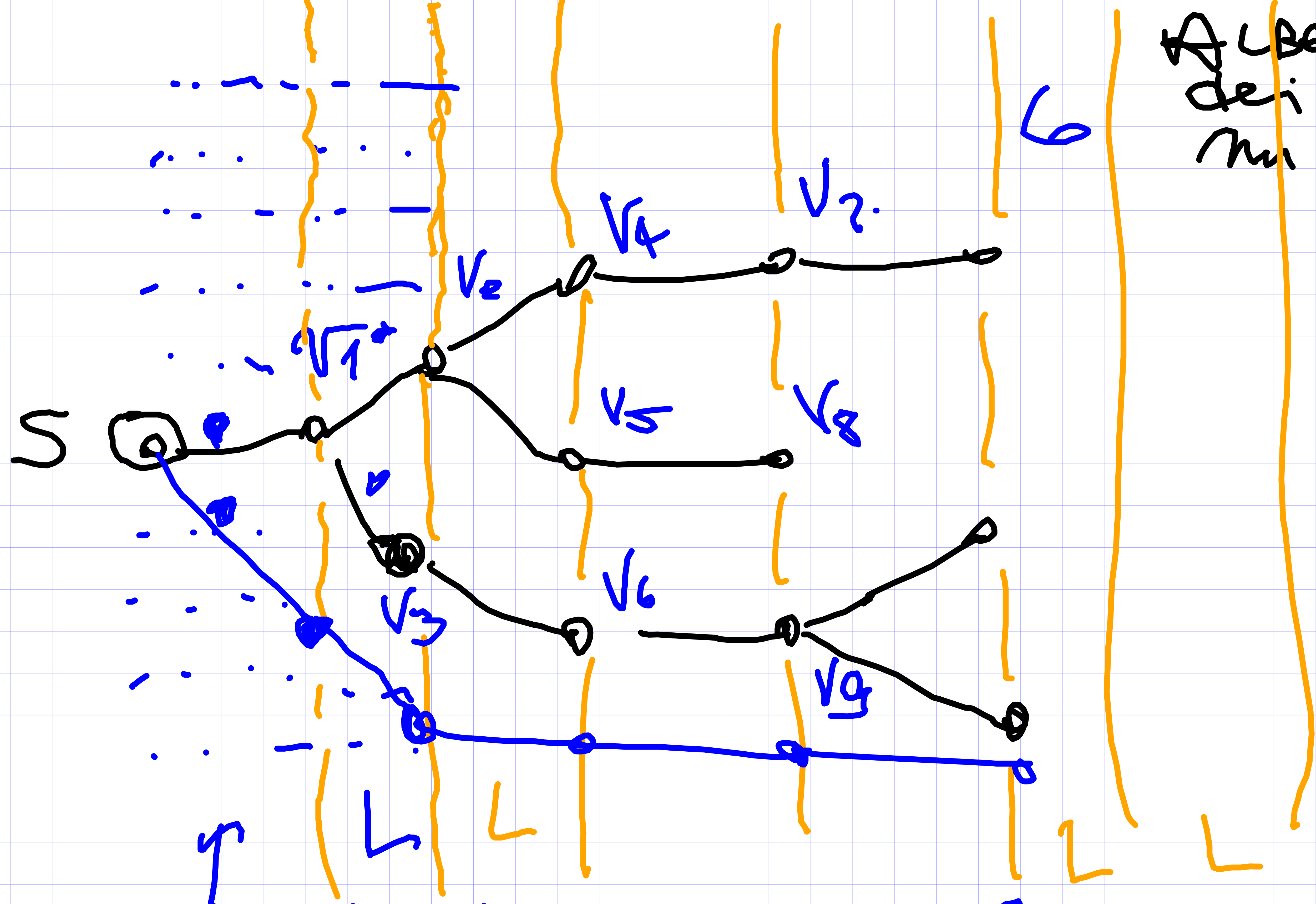
```
BellmanFord(G=(V,E); w: V -> E): boolean
{ INIZIALIZZA
  Crea una lista L di tutti gli archi

I°  for i <- 1 to |V|-1
        rilassa tutti gli archi in L
        nell'ordine prestabilito

    for e = (u,v) ∈ L do
        if d(v) > d(u) + w(u,v) then
            return TRUE
    return FALSO
}
```

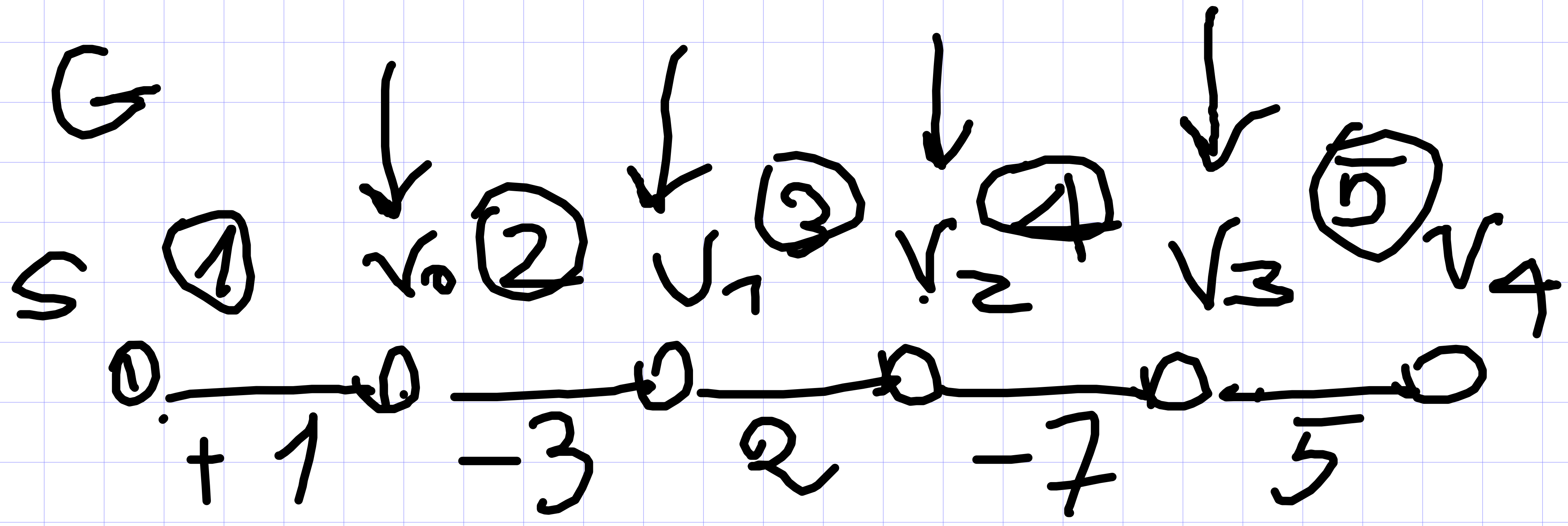
Albero dei cammini minimi.

Illustrazione per prove di correttezza



L = tutti gli archi in G

Esempio

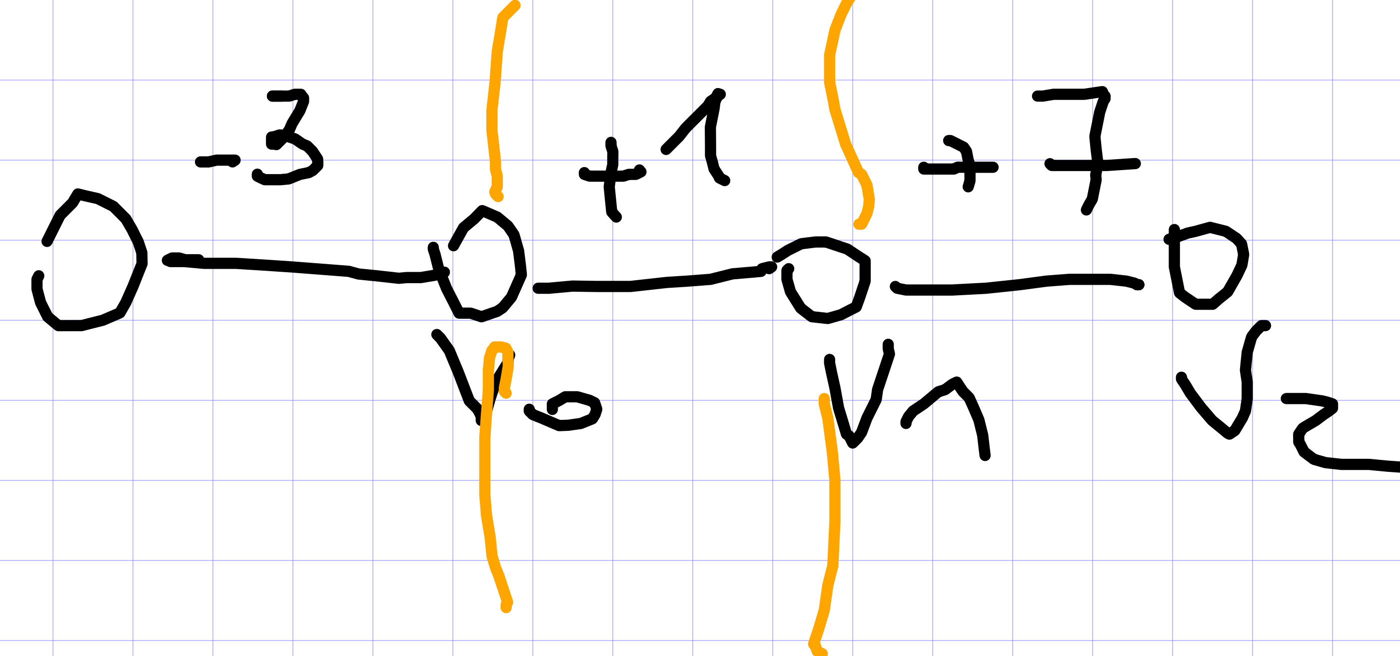


$|V| = 6$

$L = \{(s, v_0); (v_0, v_1); (v_3, v_4); (v_2, v_3); (v_1, v_2)\}$

	s	v_0	v_1	v_2	v_3	v_4
d	0	1	-2	0	-7	-2
π	$\emptyset$	s	v_0	v_1	v_2	v_3

1°

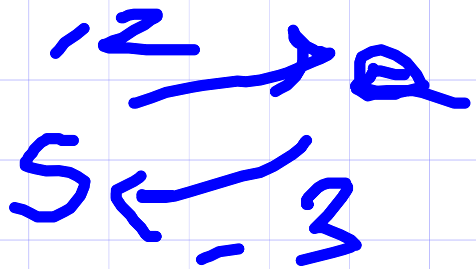


$L = \{(v_1, v_2); (v_0, v_1), (s, v_0)\}$

	s	v_0	v_1	v_2
d	ϕ	-3	-2	5
π	ϕ	s	v_0	v_1

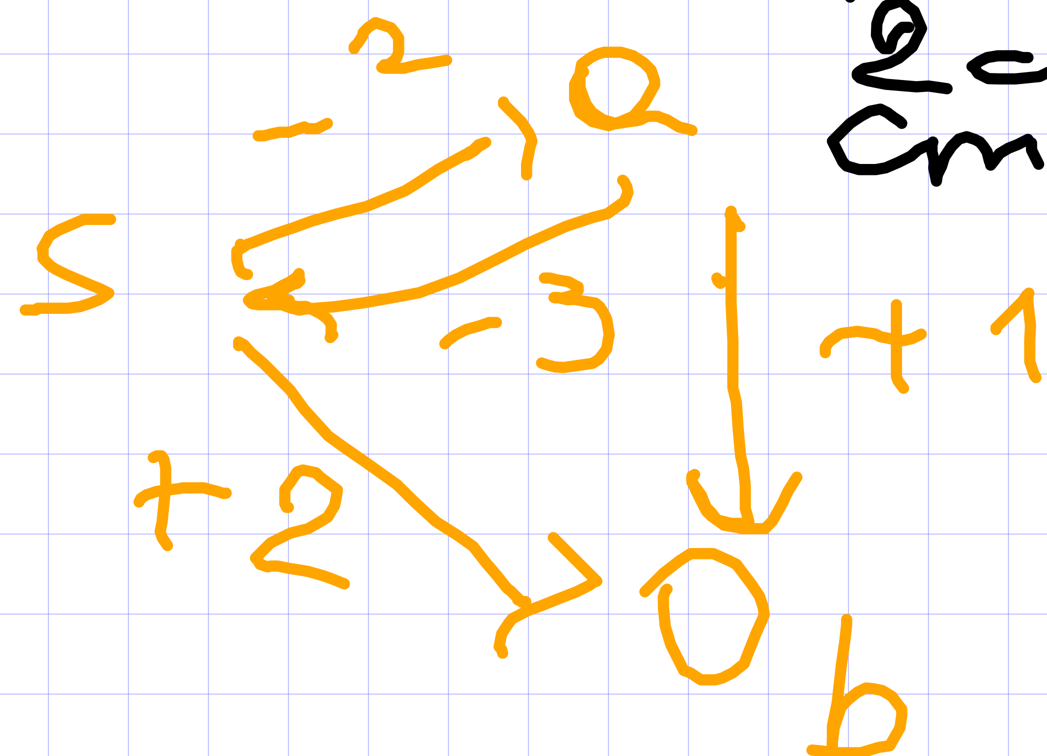
ITERATA
1
2
3
4 —
~~5~~

Esempio G = path
L è tale per cui servono 4 iterate per stabilire che $\nexists$ cicli negativi e fa $\nexists$ trovare le ch[illegible]



Esempio con ciclo di lunghezza 2 che comunque stabilisce l'ultima del ciclo alla 4th iterata

$L = \{(a,b),\ (a,s),\ (s,a),\ (s,b)\}$



	s	a	b
d	$\emptyset$	-2	2
π	$\emptyset$	s	s

1

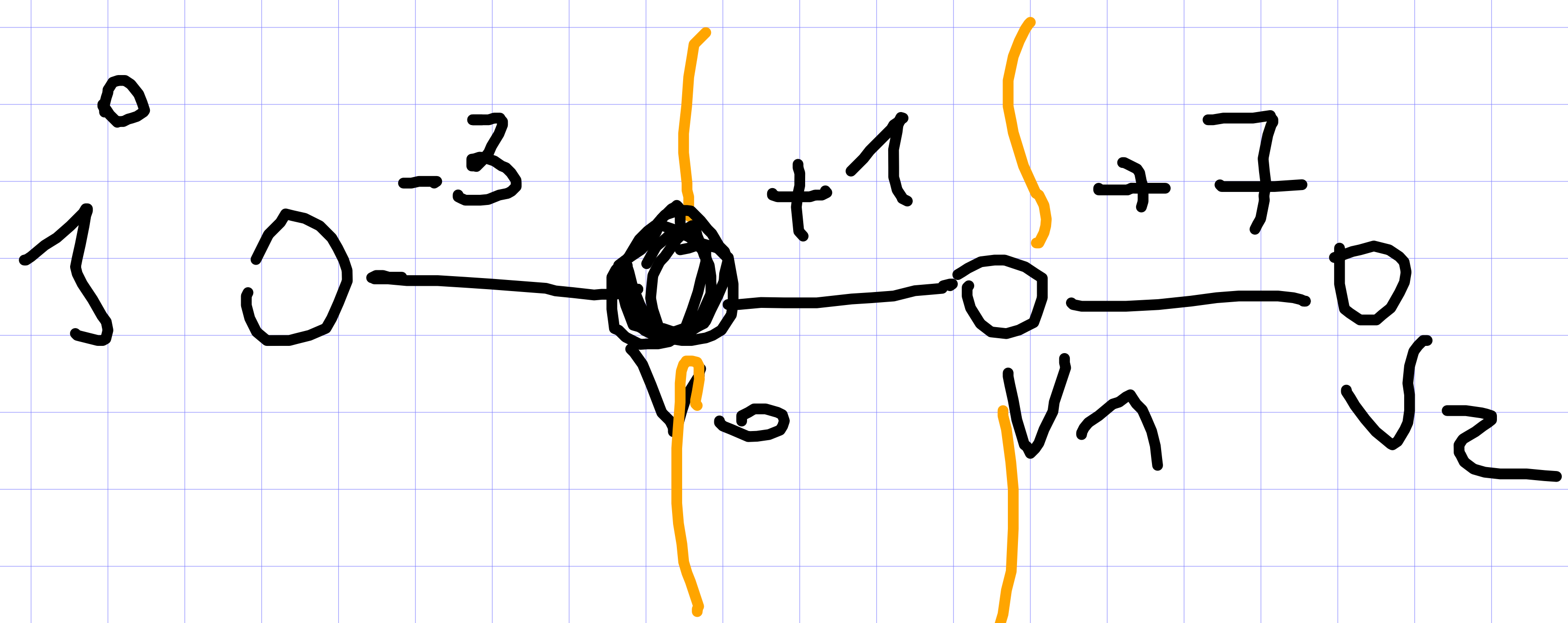
	s	a	b
d	-5	-7	-3
π	a	s	s

2

	s	a	b
d	-10	-12	-8
π	a	s	s

3

ciclo!



$$L = \{ \underline{(s, v_0)};\ (v_0, v_1);\ \underline{(v_1, v_2)} \}$$

	s	v_0	v_1	v_2
d	0	-3	-2	+5
π	$\emptyset$	s	v_0	v_1

ITERATA

1

2 —

Grafo è un parte

Esempio in cui con una sola iterata ho trovato l'albero dei cammini minimi

Correttezza

Se conosco che cammino ottimo da $s \to u$ è costituito dagli archi $(s,v) = e_1, e_2, \dots, e_r = (z,u)$ e rilasso nell'ordine $e_1, e_2, \dots, e_r$ trovo il costo $d(u) = \delta(s,u)$

Non conosco l'ordine per cui per ogni posizione i provo tutti gli archi e sicuramente passerò per l'arco che è l'i-esimo del cammino ottimo.

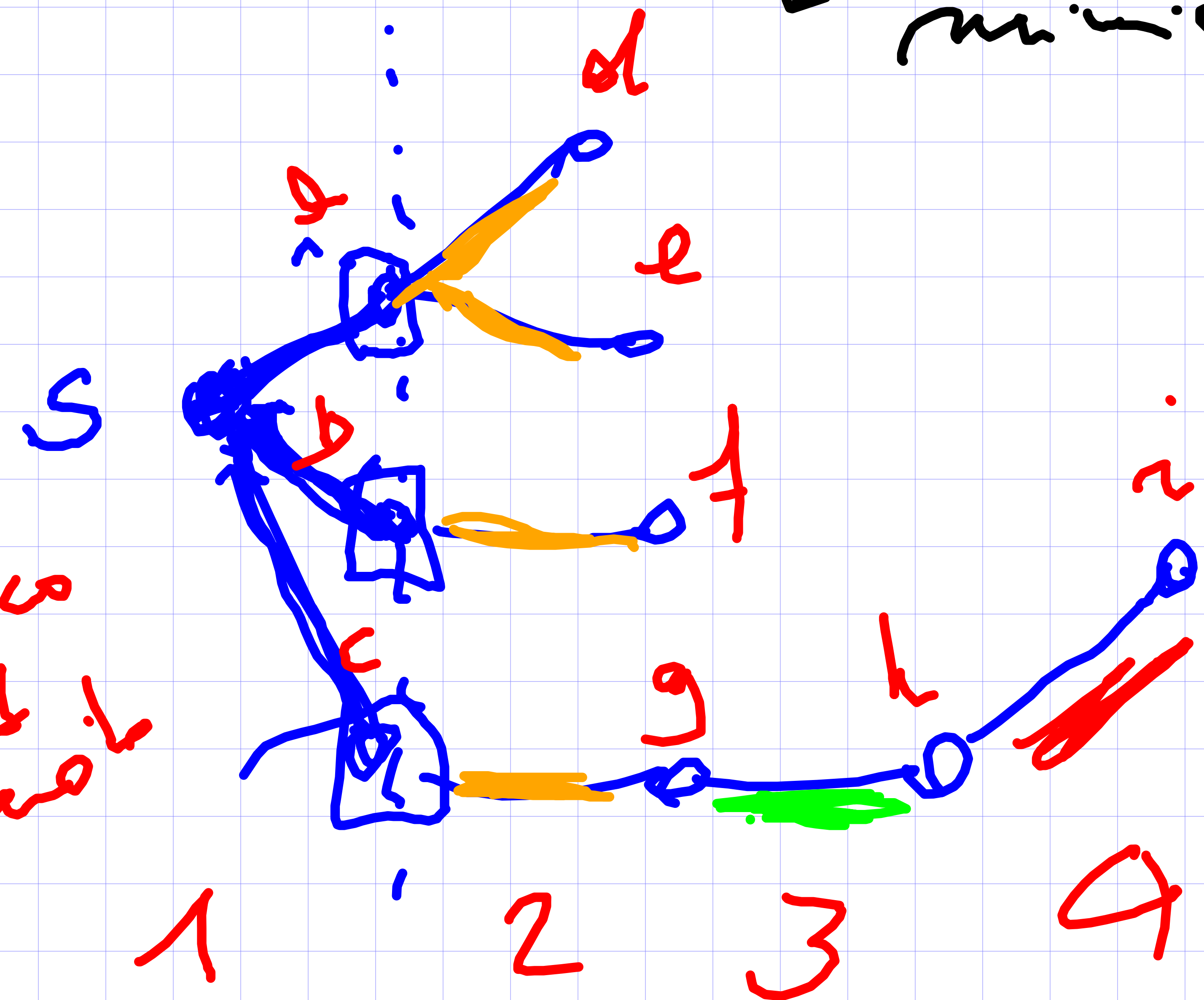
$n = |V|$

Cammini minimi sono semplici.

Hanno $\leq n-1$ archi.

Se trovo che rilasso un arco in posizione n (ossia trovo un cammino minimo di lunghezza n) allora $\exists$ (TRUE) un cammino con un ciclo di costo negativo.

$\Rightarrow \nexists$ soluzione al problema albero dei cammini minimi



giro di ritardo

L

dopo la i-esima iterata ho sistemato almeno tutti i vertici a distanza $\leq i$ da s

L = (s,a)(s,b)(s,c)(a d)(a e)(b f)(c g)(g h)(h i)

Complessità in tempo

Ciclo lista $|E|$
rilasso tutti gli archi $|V|-1$ volte
$O(VE)$
rilasso ancora gli archi
$O(E)$

Complessità $O(VE)$
Non dipende dalla rappresentazione.

cammini minimi da sorgente singola

---

cammini minimi
∀ coppia di vertici

$\forall v \in V$
shortest path
con sorgente v

Floyd Warshall
METODO TROPICALE

Algoritmo di Johnson