Drumuri minime de la un vârf s dat la celelalte vârfuri

(de sursă unică)

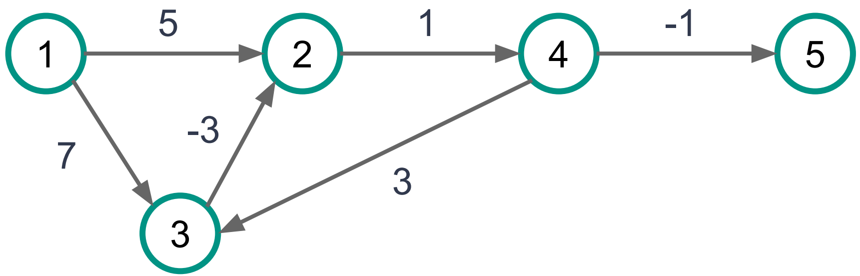
Algoritmul Bellman-Ford

Algoritmul Bellman-Ford

Ipoteză:

- Arcele pot avea și cost **negativ**
- Graful **nu** conține circuite de cost negativ
  - (dacă există ⇒ algoritmul le va detecta ⇒ nu are soluție)

Algoritmul Bellman-Ford

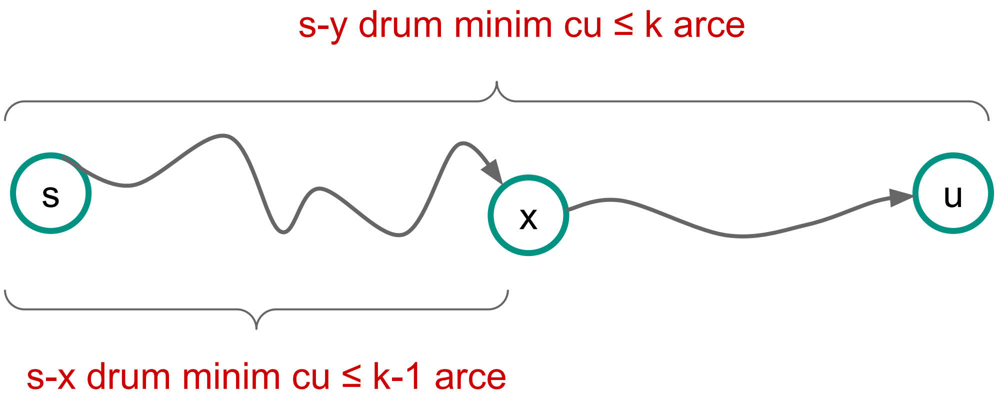


Algoritmul lui Dijkstra - doar pentru ponderi nenegative

Algoritmul Bellman-Ford

Idee: La un pas, relaxăm **toate arcele**

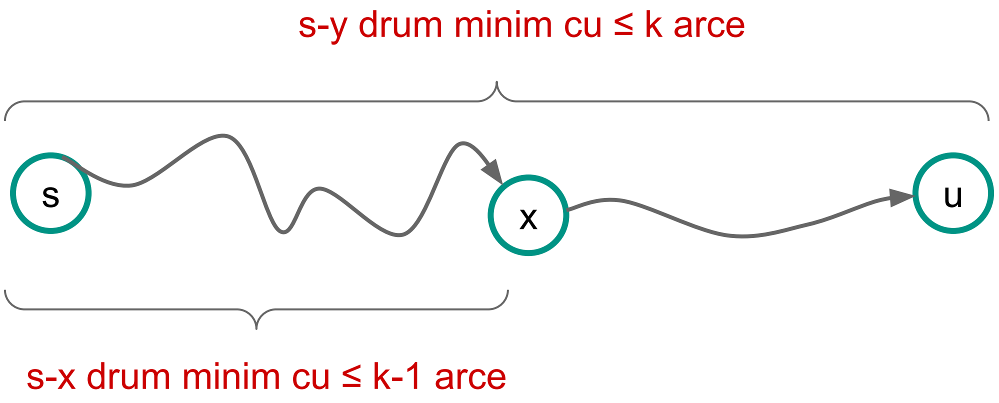
De câte ori?



Algoritmul Bellman-Ford

Idee: La un pas, relaxăm **toate arcele**

De câte ori?



Dacă P este s-y drum cu ≤ k arce $\Rightarrow$ $[s \xrightarrow{P} x]$ este s-x drum minim cu ≤ k-1 arce

Un drum minim are cel mult n-1 arce ⇒ n-1 etape

Algoritmul Bellman-Ford

Idee: La un pas, relaxăm **toate arcele**

Nu relaxăm arcele dintr-un vârf selectat u, ci **din toate vârfurile**.

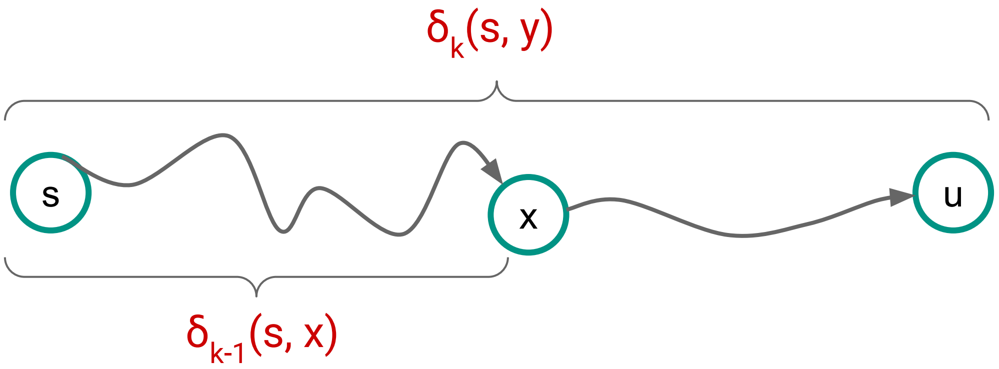
- **n-1 etape** - după k etape, d[u] ≤ costul minim al unui drum de la s la u cu cel mult k arce (programare dinamică)

⇒ după k etape, sunt corect calculate etichetele d[u] pentru acele vârfuri u pentru care **există un s-u drum minim cu cel mult k arce**

Algoritmul Bellman-Ford

Notăm

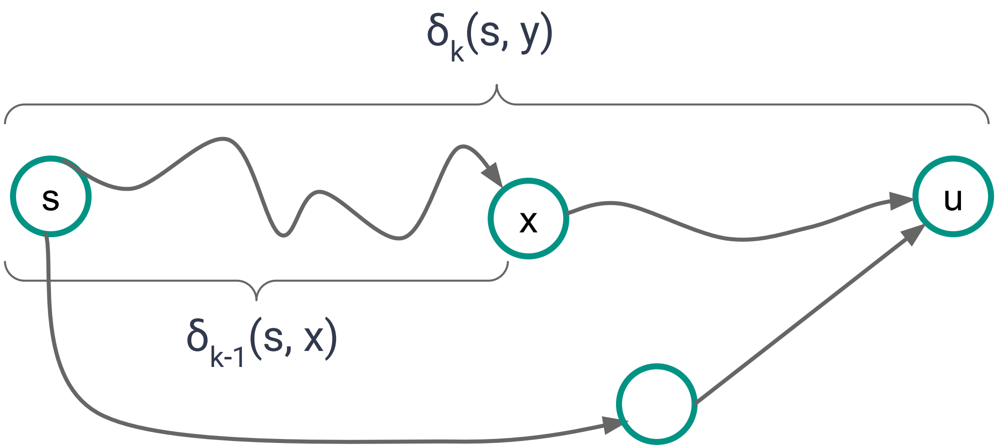
$\delta_k(s, x)$ = costul minim al unui s-x drum cu cel mult k arce



Algoritmul Bellman-Ford

Notăm

$\delta_k(s, x)$ = costul minim al unui s-x drum cu cel mult k arce



Avem:

$$\delta_k(s, y) = \min\{\ \delta_{k-1}(s, y),\quad \min\{\ \delta_{k-1}(s, x) + w(xy) \mid xy \in E\ \}\ \}$$

Algoritmul Bellman-Ford

```
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0

d[s] = 0

pentru i = 1,n-1 execută
    pentru fiecare uv∈E execută
        dacă d[u]+w(u, v) < d[v] atunci
            d[v] = d[u]+w(u, v)
            tata[v] = u
```

Algoritmul Bellman-Ford

```
pentru fiecare u∈V execută
    d[u] = ∞; tata[u] = 0

d[s] = 0

pentru i = 1,n-1 execută
    pentru fiecare uv∈E execută
        dacă d[u]+w(u, v) < d[v] atunci
            d[v] = d[u]+w(u, v)
            tata[v] = u
```

Complexitate: O(nm)

Algoritmul Bellman–Ford

Optimizări

La un pas, este suficient să relaxăm arcele din vârfuri ale căror etichetă s-a modificat anterior

Algoritmul Bellman-Ford

Optimizări

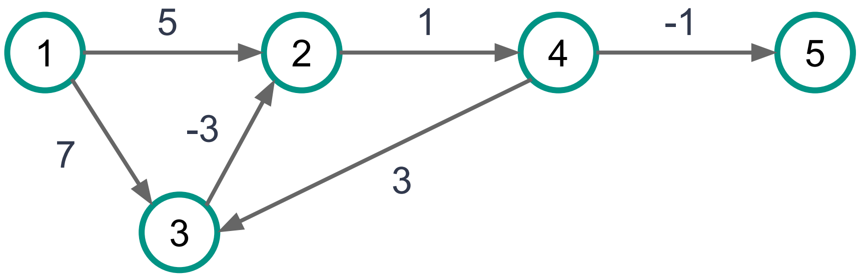
La un pas, este suficient să relaxăm arcele din vârfuri ale căror etichetă s-a modificat anterior

⇒ **coadă** cu vârfurile ale căror etichetă s-a actualizat

(Detalii - laborator)

Algoritmul Bellman-Ford

Etapa 1



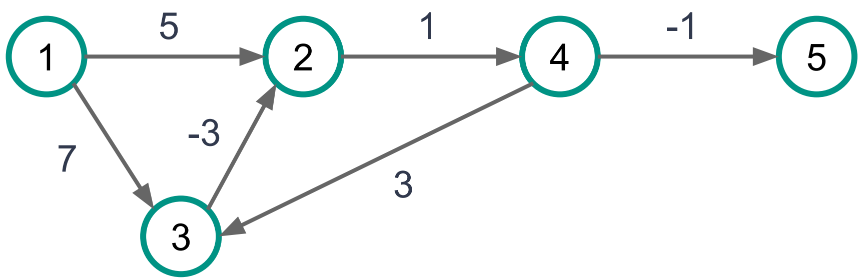
Relaxăm

1 2
1 3

	1	2	3	4	5
d	0	5	7	∞	∞

Algoritmul Bellman-Ford

Etapa 1



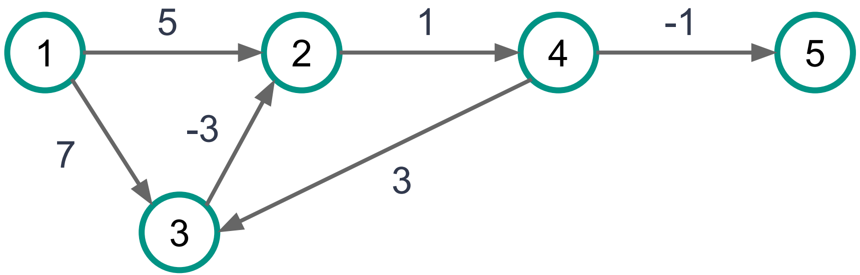
Relaxăm

1 2
1 3
2 4

	1	**2**	**3**	**4**	**5**
d	0	5	7	6	∞

Algoritmul Bellman-Ford

Etapa 1



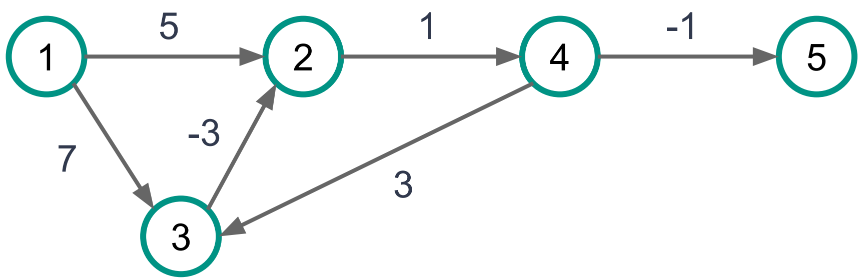
Relaxăm

1 2
1 3
2 4
4 3
4 5

	1	2	3	4	5
d	0	5	7	6	5

Algoritmul Bellman-Ford

Etapa 1



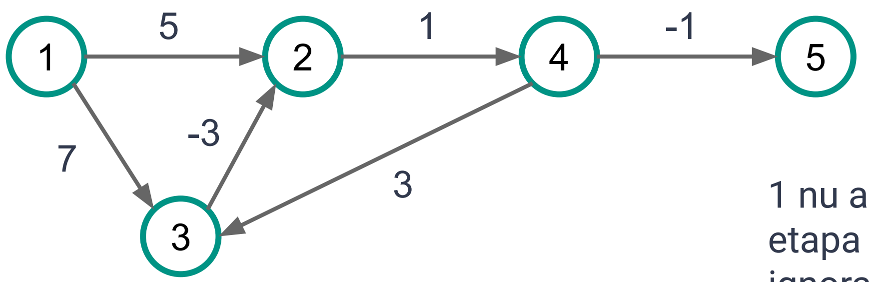
Relaxăm

1 2
1 3
2 4
4 3
4 5
3 2

	1	**2**	**3**	**4**	**5**
d	0	4	7	6	5

Algoritmul Bellman–Ford

Etapa 2



Relaxăm

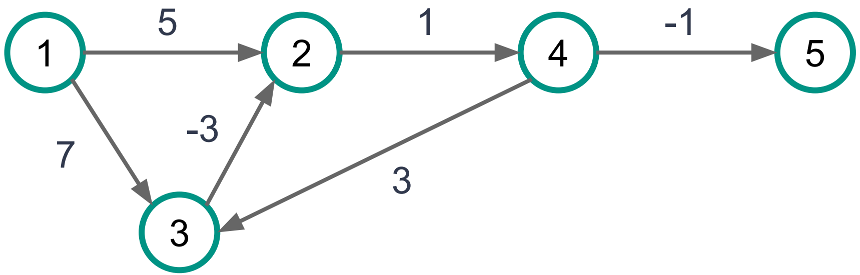
1 2
1 3

1 nu a fost actualizat la etapa anterioară, le putem ignora

	1	**2**	**3**	**4**	**5**
d	0	4	7	6	5

Algoritmul Bellman–Ford

Etapa 2



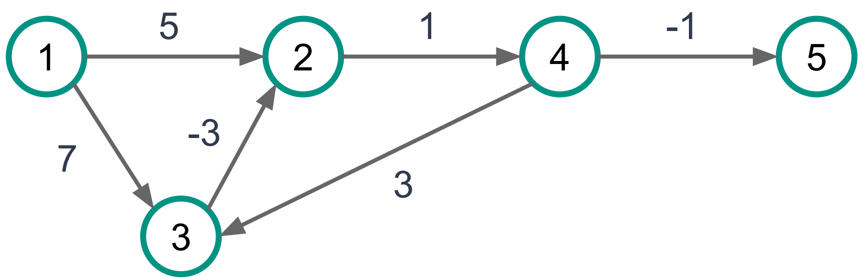
Relaxăm

1 2
1 3
2 4

	1	2	3	4	5
d	0	4	7	5	5

Algoritmul Bellman-Ford

Etapa 2



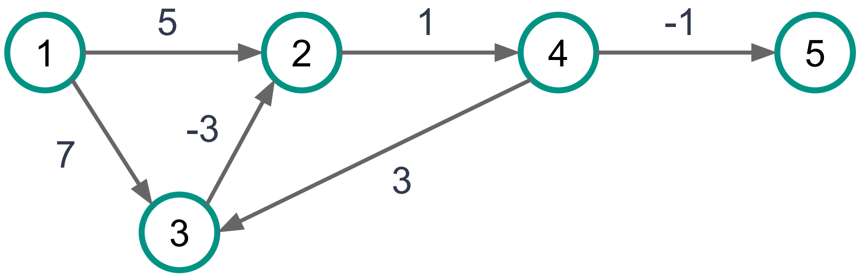
Relaxăm

1 2
1 3
2 4
4 3

	1	**2**	**3**	**4**	**5**
d	0	4	7	5	5

Algoritmul Bellman-Ford

Etapa 2



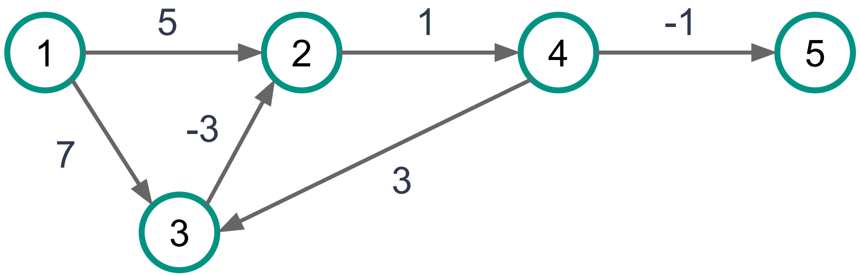
Relaxăm

1 2
1 3
2 4
4 3
4 5

	1	**2**	**3**	**4**	**5**
d	0	4	7	5	4

Algoritmul Bellman-Ford

Etapa 2



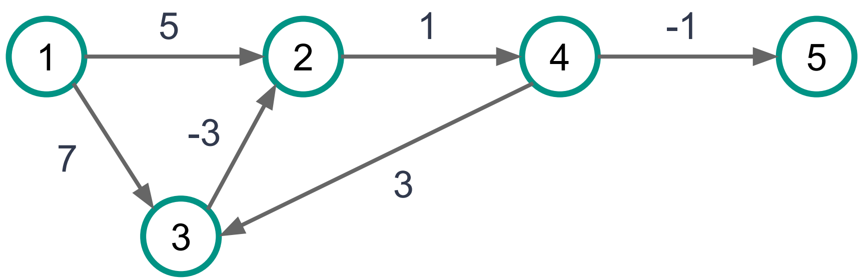
Relaxăm

1 2
1 3
2 4
4 3
4 5
3 2

	1	**2**	**3**	**4**	**5**
d	0	4	7	5	4

Algoritmul Bellman-Ford

Etapa 3



Relaxăm

1 2
1 3
2 4
4 3
4 5
3 2

Nu se mai actualizează nimic - STOP

	1	**2**	**3**	**4**	**5**
d	0	4	7	5	4

Corectitudinea algoritmului Bellman–Ford

Corectitudine

Lema 1 (comună).

Pentru orice u∈V, la orice pas al algoritmului, avem:

- **dacă d[u] < ∞**, există un drum de la s la u în G de cost d[u] și acesta se poate determina din vectorul tata
  - tata[u] = predecesorul lui u pe un drum de la s la u de cost d[u]

- **d[u] ≥ δ(s, u)**

Corectitudine

Lema 1 (comună) - Demonstrație

Inducție după numărul de etape (o etapă = relaxarea tuturor muchiilor)

După k iterații

$d[x] \le \delta_k(s, x)$

= costul minim al unui s-x drum cu cel mult k muchii

Corectitudine

$d[x] \leq \delta_k(s, x)$

$= \text{costul minim al unui } s\text{-}x \text{ drum cu cel mult } k \text{ muchii}$

$k = 0$: $d[s] = 0 = w([s])$

$k-1 \Rightarrow k$: Presupunem că, înainte de iterația k:

$$d[x] \leq \delta_{k-1}(s, x) \quad \text{pentru orice } x$$

Eticheta unui vârf y la iterația k se actualizează astfel:

Corectitudine

$d[x] \le \delta_k(s, x)$

= costul minim al unui s-x drum cu cel mult k muchii

$k = 0$: $d[s] = 0 = w([s])$

$k-1 \Rightarrow k$: Presupunem că, înainte de iterația k:

$d[x] \le \delta_{k-1}(s, x)$ pentru orice x

ipoteza de inducție

Eticheta unui vârf y la iterația k se actualizează astfel:

$d[y] \le \min\{\ d[y], \min\{\ d[x]+w(x,y) \mid xy \in E\ \}\ \}$

Corectitudine

$d[x] \le \delta_k(s, x)$

= costul minim al unui s-x drum cu cel mult k muchii

$k = 0$: $d[s] = 0 = w([s])$

$k-1 \Rightarrow k$: Presupunem că, înainte de iterația k:

$d[x] \le \delta_{k-1}(s, x)$ pentru orice x

ipoteza de inducție

Eticheta unui vârf y la iterația k se actualizează astfel:

$$d[y] \le \min\{ d[y], \min\{ d[x]+w(x,y) \mid xy \in E \} \}$$

$$\le \min\{ \delta_{k-1}(s, y), \min\{ \delta_{k-1}(s, x)+w(x,y) \mid xy \in E \} \}$$

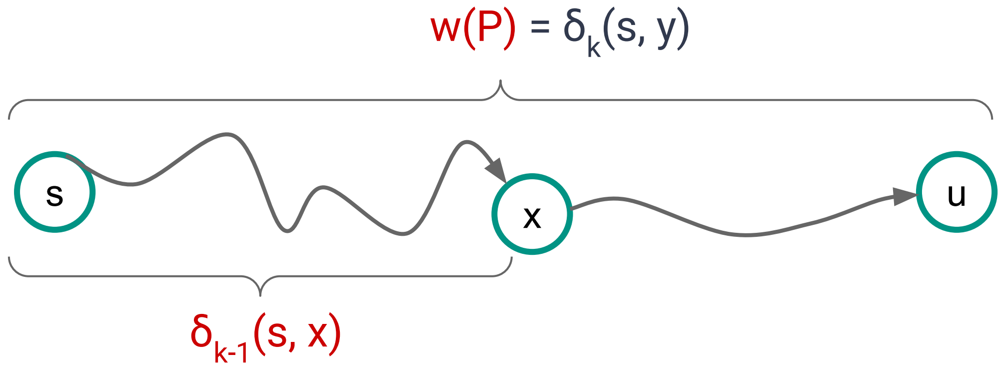
$$= \delta_k(s, y)$$

Corectitudine

k-1 ⇒ k: Presupunem că, înainte de iterația k:

$$d[x] \le \delta_{k-1}(s, x) \quad \text{pentru orice } x$$

Varianta 2: Fie un s-y drum cu cost minim printre cele cu cel mult k arce ($w(P) = \delta_k(s, y)$)



Corectitudine

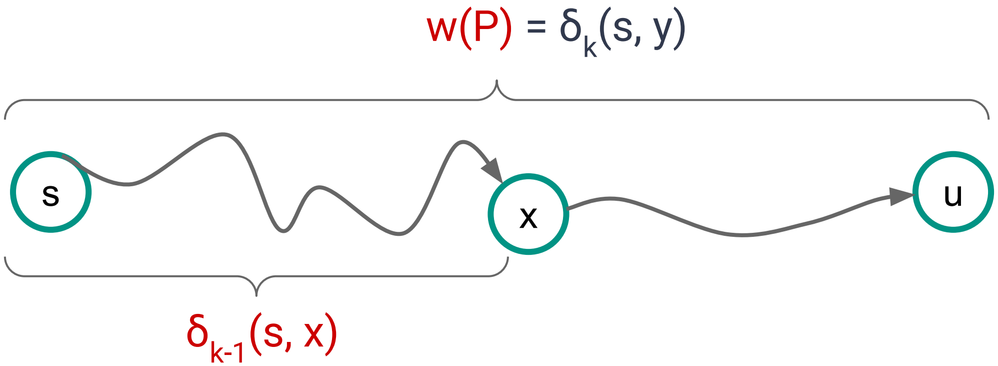
$k-1 \Rightarrow k$: Presupunem că, înainte de iterația k:

$$d[x] \le \delta_{k-1}(s, x) \quad \text{pentru orice } x$$

Varianta 2: Fie un s-y drum cu cost minim printre cele cu cel mult k arce ($w(P) = \delta_k(s, y)$)

$\Rightarrow [s \overset{P}{—} x]$ este un s-x drum cu cost minim printre cele cu cel mult k-1 arce, deci are cost $\delta_{k-1}(s, x)$

$\Rightarrow d[x] \le \delta_{k-1}(s, x)$



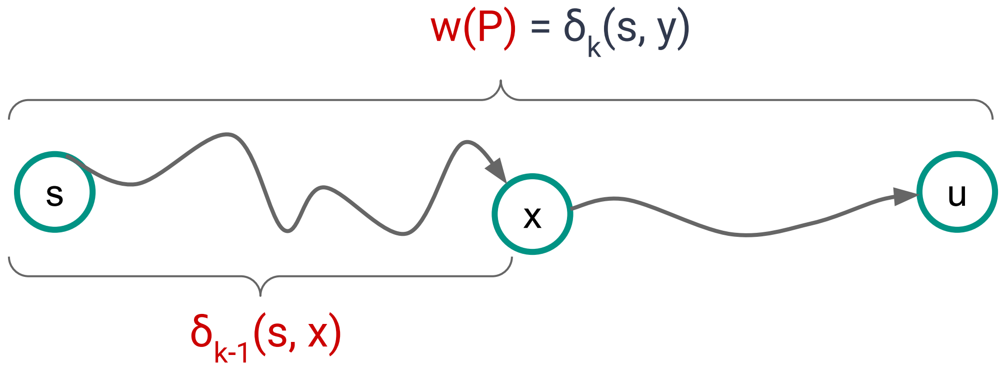
Corectitudine

După relaxarea arcului xy:

$$d[y] \le d[x] + w(xy)$$

$$\le \delta_{k-1}(s, x) + w(xy)$$

$$= w([s \overset{P}{\text{—}} x]) + w(xy) = w(P) = \delta_k(s, y)$$



Detectarea de circuite negative

Detectarea de circuite negative

Există circuit negativ în G

⇔ dacă algoritmul ar mai face o iterație, s-ar mai actualiza etichete de distanță

⇔ după n-1 iterații, există un arc uv cu

$d[v] > d[u] + w(uv)$

Detectarea de circuite negative

Demonstrație

Arătăm că:

nu există cicluri negative ⇔ nu se mai fac actualizări la pasul n

Detectarea de circuite negative

Demonstrație

Arătăm că:

nu există cicluri negative ⇔ nu se mai fac actualizări la pasul n

- Dacă nu există cicluri negative ⇒ nu se mai fac actualizări la pasul n (din corectitudine)

Detectarea de circuite negative

Demonstrație

Arătăm că:

nu există cicluri negative ⇔ nu se mai fac actualizări la pasul n

- Dacă nu există cicluri negative ⇒ nu se mai fac actualizări la pasul n (din corectitudine)
- Dacă nu se mai fac actualizări la pasul n, pentru orice ciclu C = $[v_0, \ldots, v_p, v_0]$ ⇒ $d[v_i] \leq w(v_i v_{i+1})$

Însumând pe ciclu:

Detectarea de circuite negative

Demonstrație

Arătăm că:

nu există cicluri negative ⇔ nu se mai fac actualizări la pasul n

- Dacă nu există cicluri negative ⇒ nu se mai fac actualizări la pasul n (din corectitudine)
- Dacă nu se mai fac actualizări la pasul n, pentru orice ciclu C = $[v_0, \dots, v_p, v_0]$ ⇒ $d[v_i] \le w(v_i v_{i+1})$

Însumând pe ciclu:

$d[v_0] + \dots + d[v_p] \le d[v_0] + \dots + d[v_p] + w(v_0 v_1) + \dots + w(v_p v_0)$

$\Rightarrow 0 \le w(v_0 v_1) + \dots + w(v_p v_0) = w(C)$

Detectarea de circuite negative

Afișarea ciclului negativ detectat - folosind tata:

Detectarea de circuite negative

Afișarea ciclului negativ detectat - folosind tata:

- Fie v un vârf al cărei etichetă s-a actualizat la pasul k

Detectarea de circuite negative

Afișarea ciclului negativ detectat - folosind tata:

- Fie v un vârf al cărei etichetă s-a actualizat la pasul k
- Facem n pași înapoi din v, folosind vectorul tata (către s);   fie x vârful în care am ajuns
- Afișăm ciclul care conține pe x, folosind tata (din x până ajungem iar în x)