Algoritmul Bellman Ford

Algoritmul lui BELLMAN – FORD

- **Ipoteză:**

  Arcele pot avea și **cost negativ**

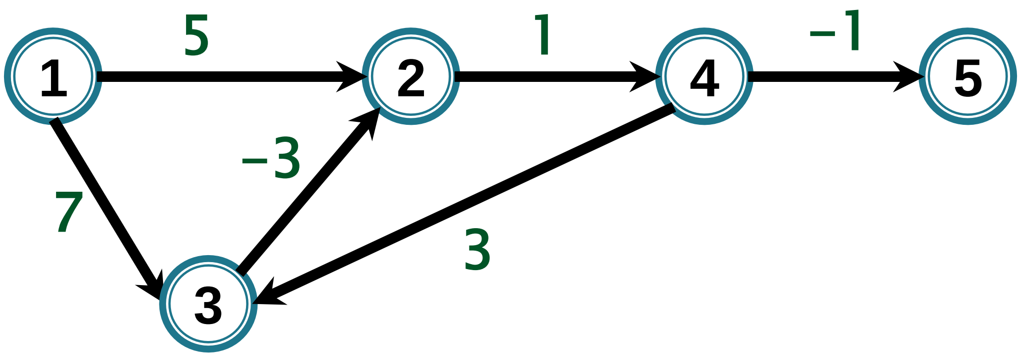
  Graful **nu** conține circuite de cost negativ

  (dacă există – algoritmul le va detecta => nu soluție)

  Toate vârfurile sunt accesibile din s

Algoritmul Bellman Ford

Algoritmul lui BELLMAN – FORD



Algoritmul lui Dijkstra – doar pentru ponderi nenegative

Algoritmul lui BELLMAN - FORD

- **Idee: La un pas relaxăm toate arcele (extindem toate drumurile deja construite)**
  - Scop: la pasul k să fie corect calculate distanțele de la s la u pentru acele vârfuri u cu proprietatea ca există s-u drum minim cu cel mult k arce
  - după k etape d[u] ≤ **costul minim al unui drum de la s la u cu cel mult k arce** (subproblemă programare dinamică)

Algoritmul lui BELLMAN – FORD

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

- De câte ori?

  Un drum minim elementar are cel mult n-1 arce => n-1 etape

Algoritmul lui 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)

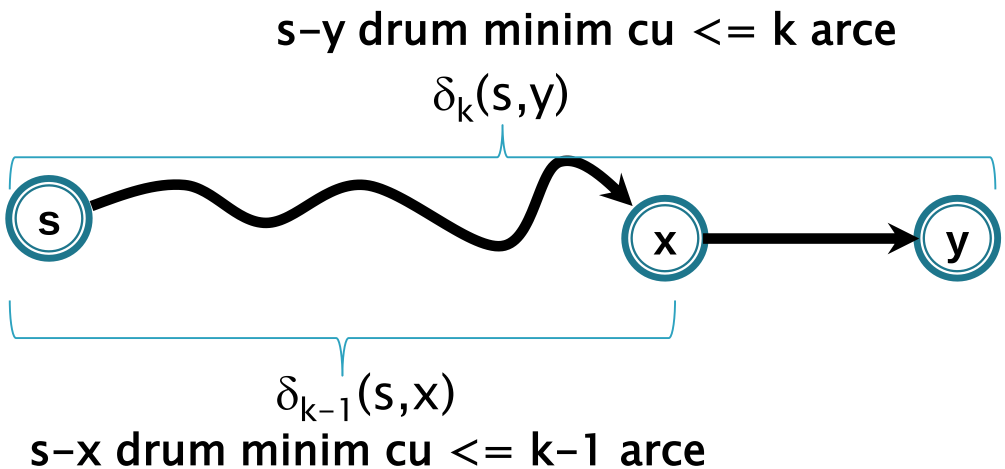
=> după k etape sunt corect calculate etichetele d[u] pentru acele vârfuri u pentru care **există un s-u drum minim cu cel mult k arce**

Algoritmul lui BELLMAN - FORD

Corectitudinea rezultă din recurența (!programare dinamică):

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

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



Dacă P este s-y drum minim cu ≤ k arce =>

[s $\underline{P}$ x] este s-x drum minim cu ≤ **k-1 arce**

Algoritmul lui BELLMAN – FORD

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

pentru k = 1,n-1 executa
   pentru fiecare uv∈E executa
         daca d[u]+w(u,v)<d[v] atunci
                d[v] = d[u]+w(u,v)
                tata[v] = u
```

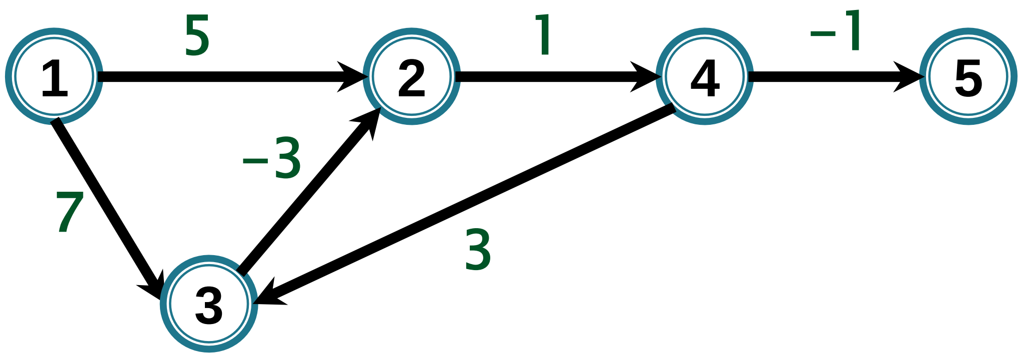
Algoritmul lui BELLMAN – FORD

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

pentru k = 1,n-1 executa
   pentru fiecare uv∈E executa
         daca d[u]+w(u,v)<d[v] atunci
                d[v] = d[u]+w(u,v)
                tata[v] = u
```

Complexitate: O(nm)

Etapa 1



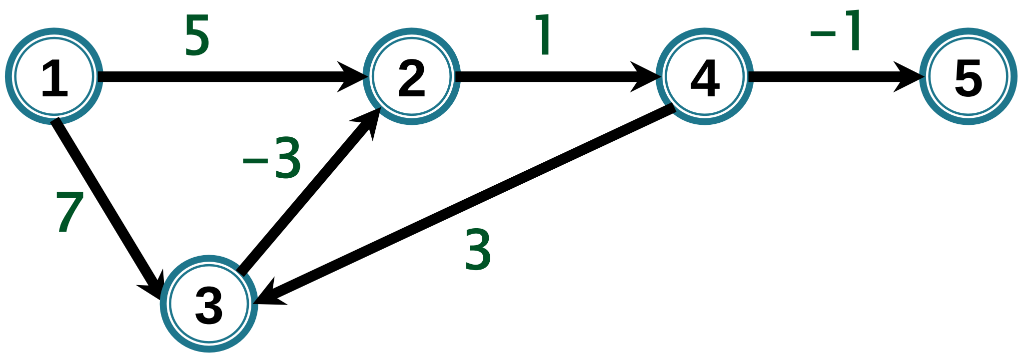
Relaxăm

1 2

1 3

	1	2	3	4	5
d	0	5	7	∞	∞

Etapa 1



Relaxăm

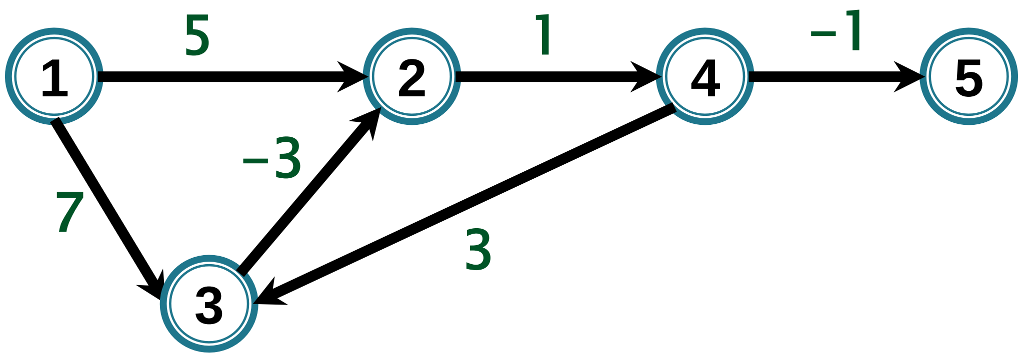
1 2

1 3

2 4

	1	2	3	4	5
d	0	5	7	6	∞

Etapa 1



Relaxăm

1 2

1 3

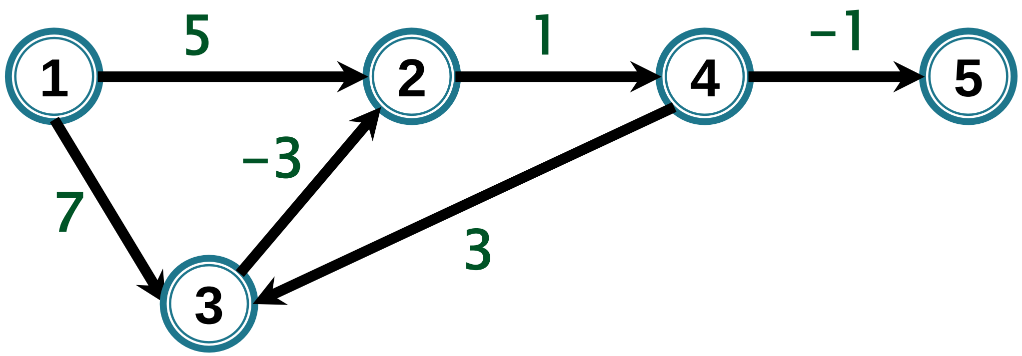
2 4

4 3

4 5

	1	2	3	4	5
d	0	5	7	6	5

Etapa 1



Relaxăm

1 2

1 3

2 4

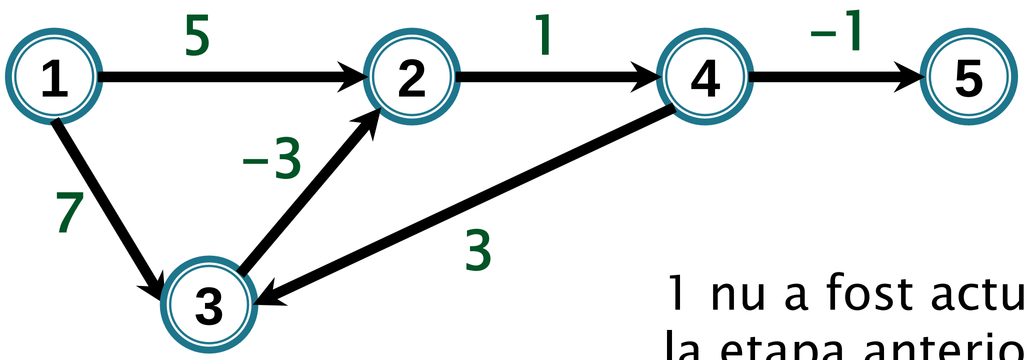
4 3

4 5

3 2

	1	2	3	4	5
d	0	4	7	6	5

Etapa 2



Relaxăm

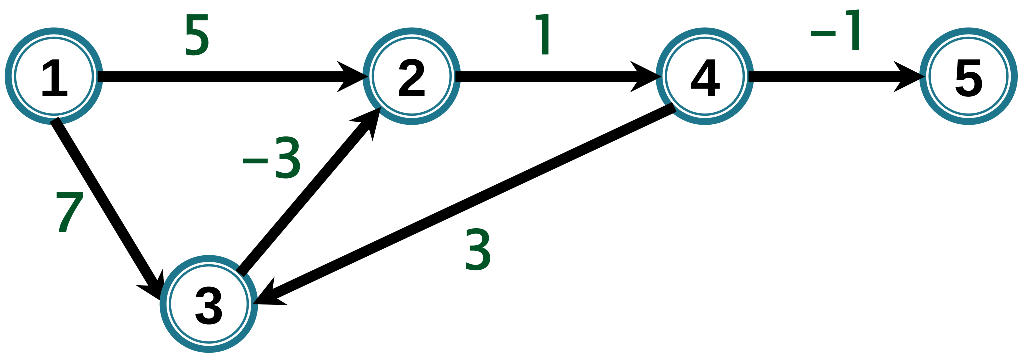
1 2

1 3

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

	1	2	3	4	5
d	0	4	7	6	5

Etapa 2



Relaxăm

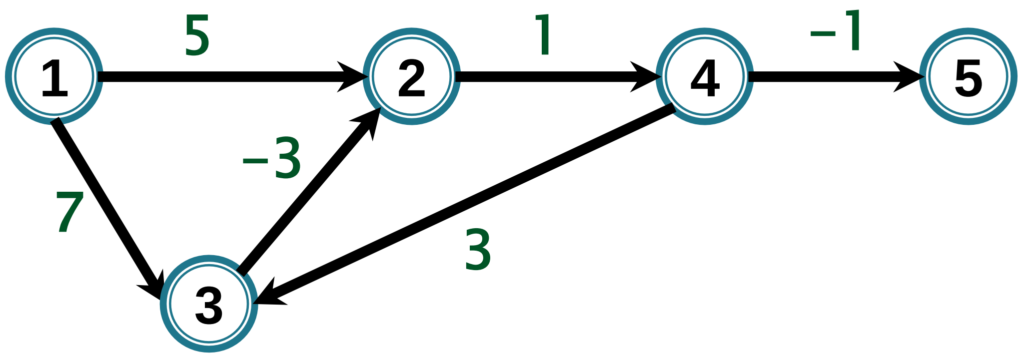
1 2

1 3

2 4

	1	2	3	4	5
d	0	4	7	5	5

Etapa 2



Relaxăm

1 2

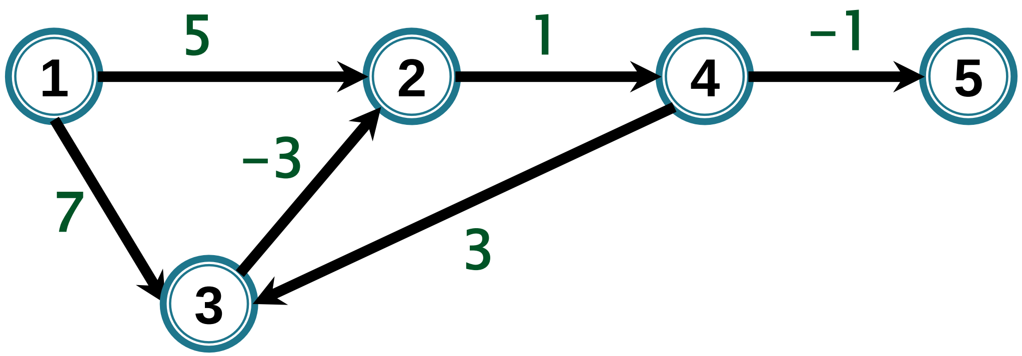
1 3

2 4

4 3

	1	2	3	4	5
d	0	4	7	5	5

Etapa 2



Relaxăm

1 2

1 3

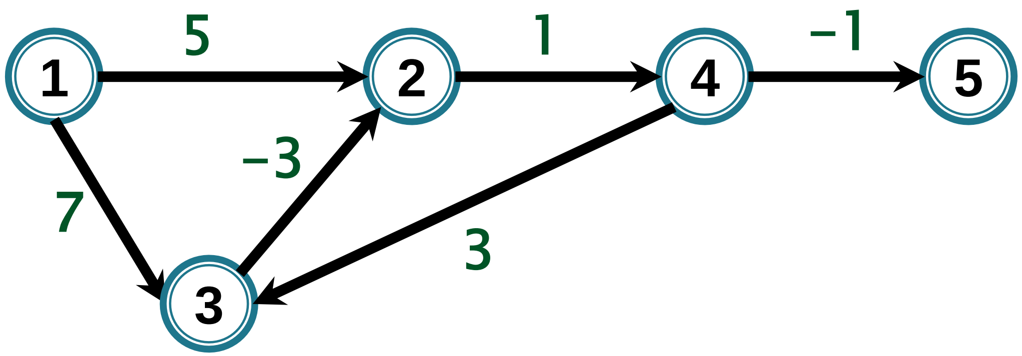
2 4

4 3

4 5

	1	2	3	4	5
d	0	4	7	5	4

Etapa 2



Relaxăm

1 2

1 3

2 4

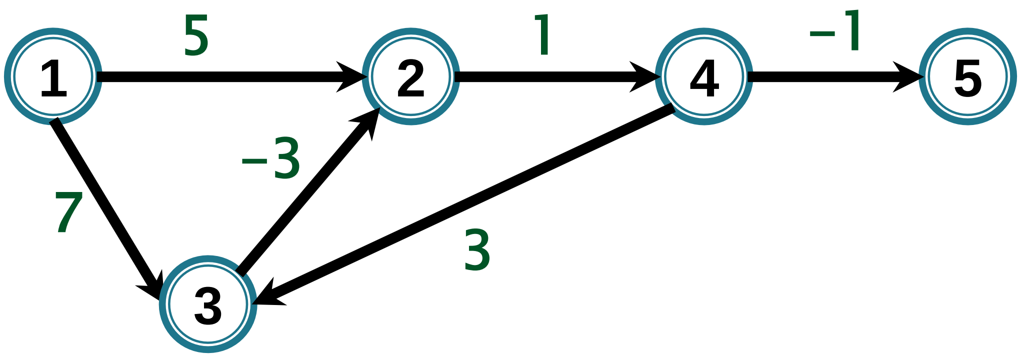
4 3

4 5

3 2

	1	2	3	4	5
d	0	4	7	5	4

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

Algoritmul lui BELLMAN - FORD

- **Optimizări**

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

  - Oprire când nu s-au mai actualizat etichete

Algoritmul lui BELLMAN - FORD

- **Optimizări**

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

  - Putem ține evidența acestora astfel:

    - folosind un vector de vizitat în care marcăm vârfurile a căror etichetă s-a actualizat la etapa curentă

    - SAU - o coadă cu vârfurile a căror etichetă s-a modificat până la etapa curentă => implementare diferită

```
queue<int> q; //coada cu vf a caror eticheta s-a actualizat
d[s] = tata[s] = 0;
q.push(s);
in_q[s] = 1; //daca un varf este deja in coada - il marchez
while (!q.empty()) {
    u = q.front();
    q.pop();
    in_q[u] = 0;
    for(j=0;j<la[u].size();j++){
        v = la[u][j].first;
        w_uv = la[u][j].second;
        if (d[u]<infinit && d[u] + w_uv < d[v]) {
            d[v] = d[u] + w_uv ;
            tata[v] = u
            if (in_q[v]==0) {
                q.push(v);
                in_q[v] = 1;
            }
        }
    }
}
```

Corectitudinea Algoritmului lui Bellman-Ford

Corectitudine

- **Demonstrație:** Inducție după numărul de etape (o etapa = relaxarea tuturor muchiilor) următoarea proprietate:

După k iterații

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

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

La final vom avea $\delta(s,x) \leq d[x] \leq \delta_{n-1}(s,x) = \delta(s,x)$,

deci $d[x] = \delta(s,x)$

Corectitudine

- Demonstram inductiv: $d[x] \leq \delta_k(s,x)$

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

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

Corectitudine

- Demonstram inductiv: $d[x] \leq \delta_k(s,x)$

  $=$ costul minim al unui s-x drum cu **cel mult** k arce

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

- $k-1 \Rightarrow k$. Presupunem că înainte de iterația k

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

Eticheta unui vârf y la iterația k se actualizează astfel:

Corectitudine

se relaxează toate arcele

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

Corectitudine

ipoteza de inducție

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

Corectitudine

ipoteza de inducție

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

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

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

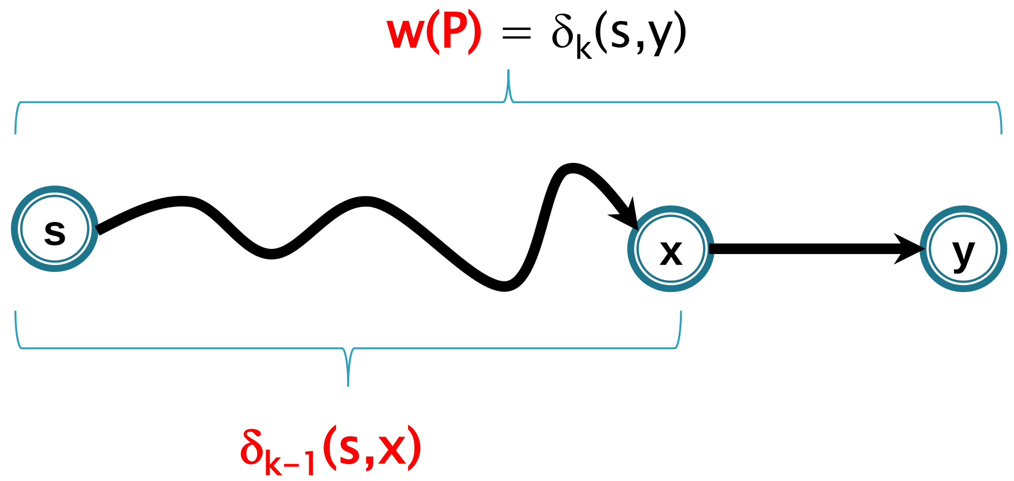
Corectitudine

Varianta 2.

- k-1 => k. Presupunem că înainte de iterația k

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

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

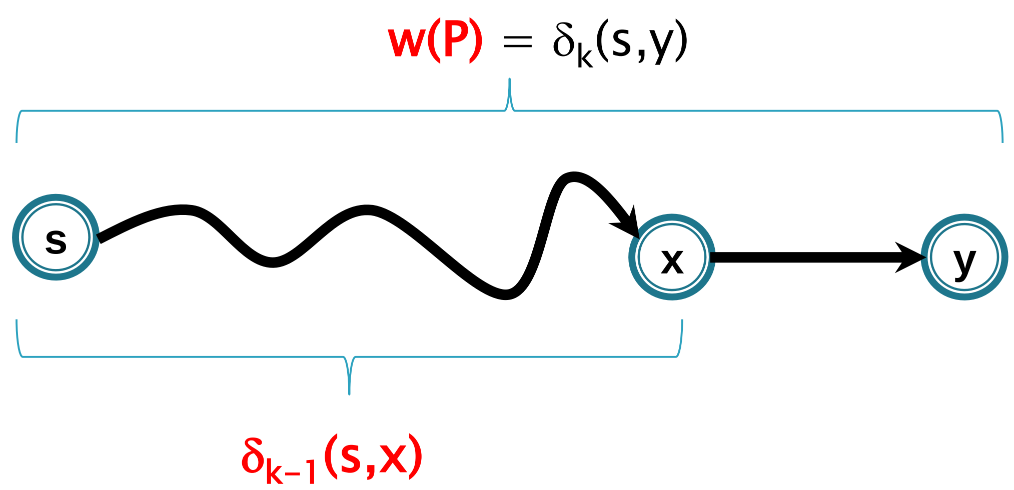


Corectitudine

Varianta 2.

=> $[s \underline{P} x]$ este s-x drum cu cost minim printre cele cu cel mult k-1 arce, deci are cost $\delta_{k-1}(s,x)$ (din ip. ind.)

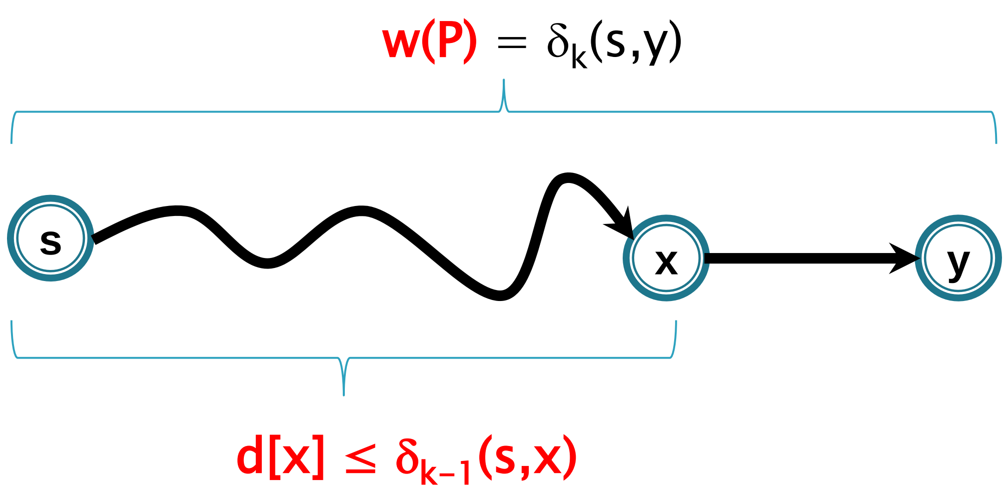
=> $d[x] \leq \delta_{k-1}(s,x)$



Corectitudine

După relaxarea arcului xy:

$$
\begin{aligned}
d[y] \le\ & d[x] + w(xy) \le \\
& \le \delta_{k-1}(s,x) + w(xy) = \\
& = w([s\ \underline{P}\ x]) + w(xy) = w(P) = \delta_k(s,y)
\end{aligned}
$$



Detectarea de circuite negative din s

Detectarea de circuite negative din s

- **Există un circuit negativ în G <u>accesibil din s</u> $\Leftrightarrow$**

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

$\Leftrightarrow$ După n-1 iterații există un arc uv cu

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

Detectarea de circuite negative accesibile din s

Demonstrație: Arătăm că

nu există circuite negative accesibile din s $\Leftrightarrow$

nu se mai fac actualizări la pasul n

Detectarea de circuite negative accesibile din s

Demonstrație: Arătăm că

nu există circuite negative accesibile din s $\Leftrightarrow$

nu se mai fac actualizări la pasul n

- Dacă nu există circuite negative => nu se mai fac actualizări la pasul n (din corectitudine)

Detectarea de circuite negative accesibile din s

Demonstrație: Arătăm că

nu există circuite negative accesibile din s $\Leftrightarrow$

nu se mai fac actualizări la pasul n

- Dacă nu se mai fac actualizări la pasul n, pentru orice circuit $C=[v_0,\ldots,v_p, v_0]$ avem $d[v_{i+1}] \leq d[v_i]+w(v_iv_{i+1})$

Detectarea de circuite negative accesibile din s

Demonstrație: Arătăm că

nu există circuite negative accesibile din s $\Leftrightarrow$

nu se mai fac actualizări la pasul n

- Dacă nu se mai fac actualizări la pasul n, pentru orice circuit $C=[v_0,\ldots,v_p, v_0]$ avem $d[v_{i+1}] \leq d[v_i]+w(v_i v_{i+1})$

Însumând pe circuit:

$d[v_o]+\ldots+d[v_p] \leq d[v_o]+\ldots+d[v_p] + w(v_o v_1)+\ldots+w(v_p v_o)$

$\Rightarrow\ 0 \leq w(v_o v_1)+\ldots+w(v_p v_o) = w(C)$

Algoritmul lui BELLMAN – FORD

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

 pentru k = 1,n-1 executa
     pentru fiecare uv∈E executa
           daca d[u]+w(u,v)<d[v] atunci
                  d[v] = d[u]+w(u,v)
                  tata[v] = u

 pentru fiecare uv∈E executa
      daca d[u]+w(u,v)<d[v] atunci
           d[v] = d[u]+w(u,v)
           tata[v] = u
           STOP, exista circuit negativ ACCESIBIL DIN s
```

Detectarea de circuite negative accesibile din s

Afișarea circuitului negativ detectat – folosind tata:

Detectarea de circuite negative accesibile din s

Afișarea circuitului negativ detectat – folosind tata:

- Fie v un vârf al cărui etichetă s-a actualizat la pasul suplimentar n (!!v nu aparține neapărat unui circuit)

Detectarea de circuite negative accesibile din s

Afișarea circuitului negativ detectat – folosind tata:

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

Detectarea de circuite negative accesibile din s

Pentru implementarea folosind coadă

```
queue<int> q; d[s] = tata[s] = 0;
q.push(s);
in_q[s] = 1;
while (!q.empty()) {
     u = q.front();
     q.pop();
     in_q[u] = 0;
     for(j=0;j<la[u].size();j++){
         v = la[u][j].first;
         w_uv = la[u][j].second;
          if (d[u]<infinit && d[u] + w_uv < d[v]) {
               d[v] = d[u] + w_uv ;
               tata[v] = u
               if (in_q[v]==0) {
                    q.push(v);
                    in_q[v] = 1;
```

Cum detectăm circuit negativ?

```
               }
     }
}
```

```
queue<int> q; d[s] = tata[s] = 0;
q.push(s);
in_q[s] = 1;
while (!q.empty()) {
    u = q.front();
    q.pop();
    in_q[u] = 0;
    for(j=0;j<la[u].size();j++){
        v = la[u][j].first;
        w_uv = la[u][j].second;
        if (d[u]<infinit && d[u] + w_uv < d[v]) {
            d[v] = d[u] + w_uv ;
            tata[v] = u
            if (in_q[v]==0) {
                q.push(v);
                in_q[v] = 1;
                nr[v]++; //de cate ori a fost inserat in q
                if (nr[v]>n)??
                    return v;
            }
        }
    }
}
```

```
queue<int> q; d[s] = tata[s] = 0;
q.push(s);
in_q[s] = 1;
while (!q.empty()) {
     u = q.front();
     q.pop();
      in_q[u] = 0;
     for(j=0;j<la[u].size();j++){
         v = la[u][j].first;
         w_uv = la[u][j].second;
        if (d[u]<infinit && d[u] + w_uv < d[v]) {
              d[v] = d[u] + w_uv ;
              tata[v] = u
              if (in_q[v]==0) {
                   q.push(v);
                   in_q[v] = 1;
              }
              lung[v]=lung[u]+1 //numarul de arce din drum?
              if (lung[v] > n-1) return v;

         }
     }
}
```