

Proiectarea Algoritmilor

Curs 9 – Drumuri de cost minim

Bibliografie

- [1] R. Sedgewick, K. Wayne - Algorithms and Data Structures Fall 2007 – Curs Princeton -
<http://www.cs.princeton.edu/~rs/AlgsDS07/>
- [2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*

Objective

- “Descoperirea” algoritmilor de identificare a drumurilor de cost minim.
- Recunoașterea caracteristicilor acestor algoritmi.

Reminder (I)

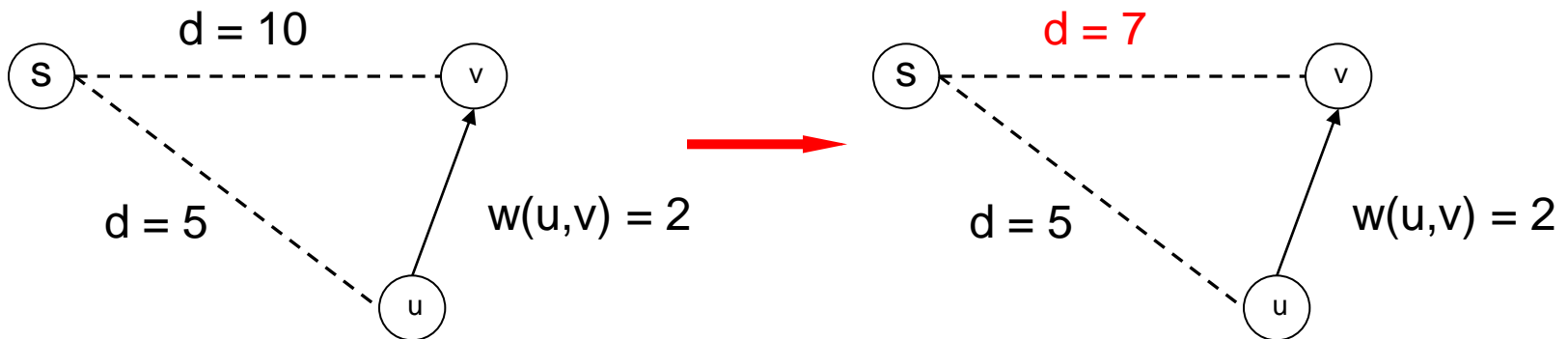
- $G = (V, E)$;
- $s \in V$ – nodul sursă;
- $w : E \rightarrow \mathbb{R}$ funcție de cost asociată arcelor grafului;
- $\text{cost}(u..v)$ = costul drumului $u..v$ (aditiv);
- $d[v]$ = costul drumului descoperit $s..v$;
- $\delta(u, v)$ = costul drumului optim $u..v$;
 - $\delta(u, v) = \infty$ dacă $v \notin R(u)$
 - $\delta(u, v) = \sum w(x, y), (x, y) \in u..v$ ($u..v$ fiind drumul optim);
- $p[v]$ = predecesorul lui v pe drumul $s..v$.

Reminder (II)

- Relaxarea arcelor:

- **Dacă $d[v] > d[u] + w(u,v)$, atunci**

- $d[v] = d[u] + w(u,v)$;
- $p[v] = u$



Concluzii Dijkstra (I)

- Dijkstra(G,s)

- Pentru fiecare nod u ($u \in V$)

- $d[u] = \infty$; $p[u] = \text{null}$;

- $d[s] = 0$;

- $Q = \text{construiește_coada}(V)$ // coadă cu priorități

- Cât timp ($Q \neq \emptyset$)

- $u = \text{ExtrageMin}(Q)$; // extrage din V elementul cu $d[u]$ minim

- // $Q = Q - \{u\}$ – se execută în cadrul lui ExtrageMin

- Pentru fiecare nod v ($v \in Q$ și v din succesorii lui u)

- Dacă ($d[v] > d[u] + w(u,v)$)

- $d[v] = d[u] + w(u,v)$ // actualizez distanța

- $p[v] = u$ // și părintele

Complexitate?

Vectori - $O(V^2)$

HB - $O(E \log V)$

HF - $O(V \log V + E)$

Concluzii Dijkstra (II)

- Implementarea trebuie realizată în funcție de tipul grafului pe care lucrăm:
 - vectori pentru grafuri “dese”;
 - heap pentru grafuri “rare”.
- Heapul Fibonacci este mai eficient decât heapul binar dar mai dificil de implementat.

Corectitudine Dijkstra – Reminder(I)

- **Lemă 25.1 (Subdrumurile unui drum minim sunt drumuri optimale):** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = v_1 v_2 \dots v_k$ un drum optim de la v_1 la v_k . Atunci pentru orice i și j cu $1 \leq i \leq j \leq k$, subdrumul lui p de la v_i la v_j este un drum minim.
- **Corolar 25.2:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = s..uv$ un drum optim de la s la v . Atunci costul optim al acestui drum poate fi scris ca $\delta(s, v) = \delta(s, u) + w(u, v)$.
- **Lemă 25.3:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. $\forall (u, v) \in E$ avem $\delta(s, v) \leq \delta(s, u) + w(u, v)$.

Corectitudine Dijkstra – Reminder(II)

- **Lemă 25.5:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. $\forall v \in V$, $d[v]$ obținut de algoritmul lui Dijkstra respectă $d[v] \geq \delta(s, v)$. În plus, odată atinsă valoarea $\delta(s, v)$, ea nu se mai modifică.
- **Lemă 25.7:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Fie $p = s..uv$ un drum optim de la s la v . Dacă $d[u] = \delta(s, u)$ la un moment dat, atunci începând cu momentul imediat următor relaxării arcului (u, v) avem $d[v] = \delta(s, v)$.

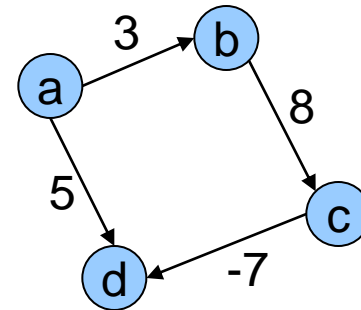
Corectitudine Dijkstra

- **Teoremă.** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată **nenegativă**. La terminarea aplicării algoritmului Dijkstra pe acest graf plecând din sursa s vom avea $d[v] = \delta(s, v)$ pentru $\forall v \in V$.
- **Dem:** prin reducere la absurd se demonstrează că la scoaterea din Q a fiecărui nod v avem $d[v] = \delta(s, v)$ și egalitatea se menține și ulterior.
 - Pp. u e primul nod pt. care $d[u] \neq \delta(s, u)$ la scoaterea din Q . $u \neq s$ pt. că altfel $d[u] = \delta(s, u) = 0$ și $u \in R(s)$ pt. că altfel $d[u] = \delta(s, u) = \infty$. \Rightarrow La scoaterea lui u din Q , \exists drum $s..u$ și fie p drumul optim $s..u$ a.î. $p = s..xy..u$, unde $x \notin Q$ iar $y \in Q$.
 - (Dacă nu impunem $y \in Q$, atunci toate nodurile până la u au fost scoase din Q și u a rămas în vârful cozii. Cum u e primul nod pentru care $d[u] \neq \delta(s, u)$, iar calea e optimă, avem situația din **Lema 25.7** \Rightarrow La scoaterea lui u din Q , $d[u] = \delta(s, u)$.)
 - Cum u e primul nod pt. care $d[u] \neq \delta(s, u) \Rightarrow d[x] = \delta(s, x)$ la momentul extragerii lui u din $Q \rightarrow d[y] = \delta(s, y)$ prin relaxarea (x, y) (conf. **Lema 25.7**).
 - y precede u pe drumul $p \Rightarrow d[y] = \delta(s, y) \leq \delta(s, u) \leq d[u]$ (conf. **Lema 25.5**).
 - Cum $y \in Q$ la momentul scoaterii lui u din $Q \Rightarrow d[u] \leq d[y]$
 - $\Rightarrow d[y] = \delta(s, y) = \delta(s, u) = d[u]$ – Contrazice ipoteza! $\Rightarrow d[u] = \delta(s, u)$ și conf. **Lema 25.5**, egalitatea se menține și ulterior.

Problemă Dijkstra

- Exemplu rulare:

- $d[a] = 0$; $d[b] = d[c] = d[d] = \infty$
- $d[b] = 3$; $d[d] = 5$;
- $d[c] = 11$;



- **d este extras din coadă!** În momentul extragerii din coadă distanța până la nodul d se consideră a fi calculată și a fi optimă.
- Se extrage nodul c; $d[d]$ nu va mai fi actualizată – nodul d fiind deja eliminat din coadă.

- → Algoritmul nu funcționează pentru grafuri ce conțin arce de cost negativ!

Exemplu practic – arce de cost negativ (I)

Currency conversion. Given currencies and exchange rates, what is best way to convert one ounce of gold to US dollars?

- 1 oz. gold \Rightarrow \$327.25. [208.10 \times 1.5714]
- 1 oz. gold \Rightarrow £208.10 \Rightarrow \Rightarrow \$327.00.
- 1 oz. gold \Rightarrow 455.2 Francs \Rightarrow 304.39 Euros \Rightarrow \$327.28.

[455.2 \times .6677 \times 1.0752]

Currency	£	Euro	¥	Franc	\$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3929	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
Gold (oz.)	0.004816	0.003295	0.0000255	0.002201	0.003065	1.0000

*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]



Exemplu practic – arce de cost negativ (II)

Graph formulation.

- Vertex = currency.
- Edge = transaction, with weight equal to exchange rate.
- Find path that maximizes **product** of weights.



*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

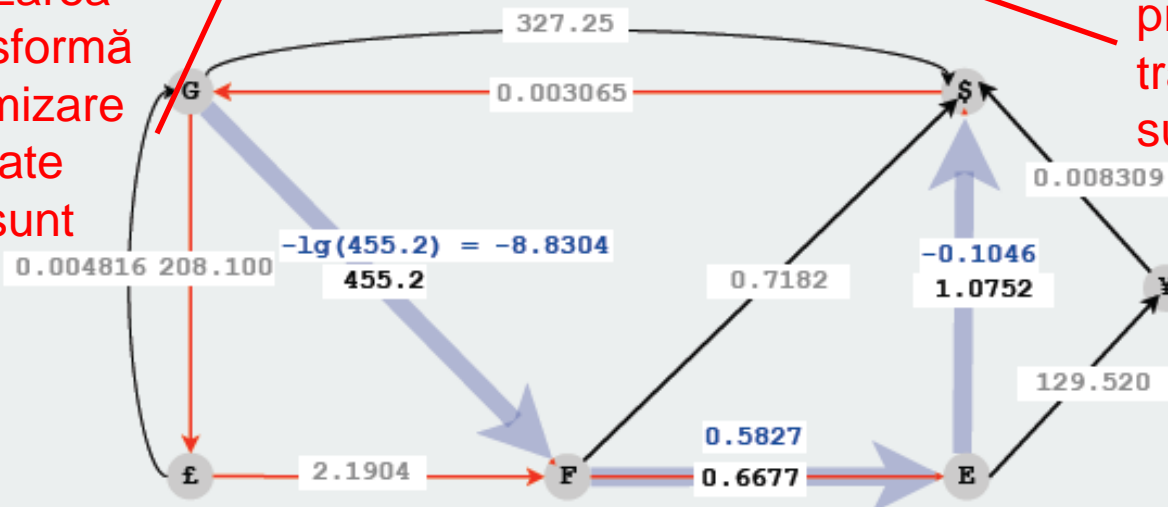
Exemplu practic – arce de cost negativ (III)

Reduce to shortest path problem by taking logs

- Let $\text{weight}(v-w) = -\lg$ (exchange rate from currency v to w)
- multiplication turns to addition
- Shortest path with costs c corresponds to best exchange sequence.

Maximizarea se transformă în minimizare dacă toate arcele sunt negate.

Prin logaritmare produsul se transformă în sumă.



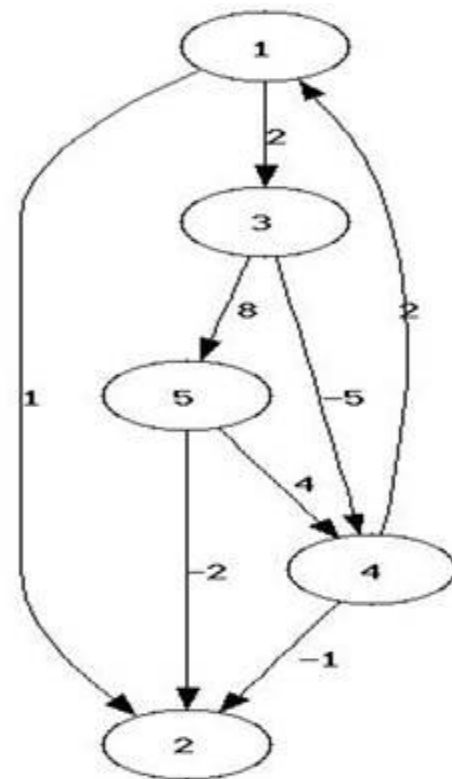
*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

Cicluri de cost negativ

$$\delta(u, v) = \begin{cases} \sum w(x,y), (x,y) \in u..v \\ (u..v \text{ fiind drumul optim}); \\ \infty, \text{ dac\u0103 nu exist\u0103 drum} \\ u..v. \end{cases}$$

- Dacă există pe drumul $u..v$ un ciclu de cost negativ $x..y \rightarrow$

- $\delta(u,v) = \delta(u,v) + \text{cost}(x..y) < \delta(u,v)$
- \rightarrow valoarea lui $\delta(u,v)$ va sc\u0103dea continuu \rightarrow costul este $-\infty$
- $\rightarrow \delta(u,v) = -\infty$

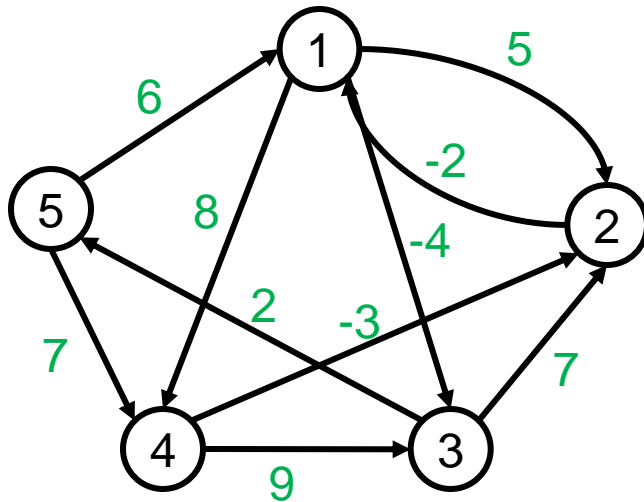


1-3-4 ciclu de cost negativ(-1) \rightarrow toate costurile din graf sunt $-\infty$

Algoritmul Bellman-Ford

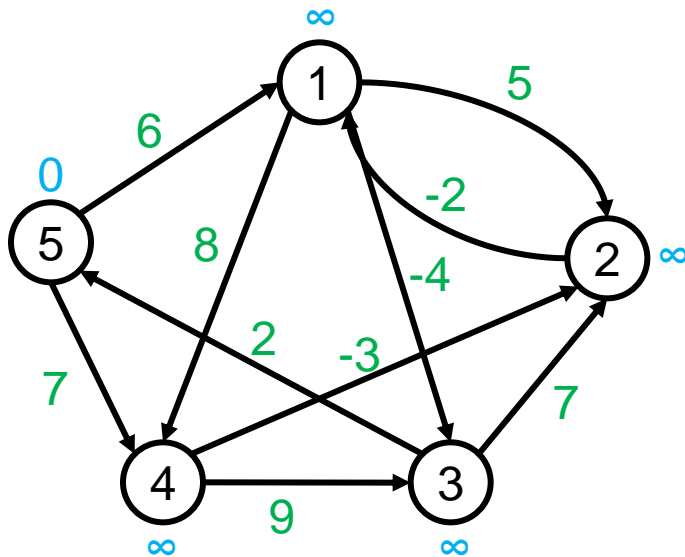
- **BellmanFord(G, s)** // $G=(V, E), s=sursa$
 - **Pentru fiecare** nod v ($v \in V$) // inițializări
 - $d[v] = \infty$;
 - $p[v] = null$;
 - $d[s] = 0$; // actualizare distanță de la s la s
 - **Pentru i de la 1 la $|V| - 1$** // pentru fiecare pas pornind din s
// spre restul nodurilor se încearcă construcția unor drumuri
// optime de dimensiune i
 - **Pentru fiecare** (u, v) din E
// pentru arcele ce pleacă de la nodurile deja considerate
 - **Dacă** $d[v] > d[u] + w(u, v)$ **atunci** // se relaxează arcele corespunzătoare
 - $d[v] = d[u] + w(u, v)$;
 - $p[v] = u$;
 - **Pentru fiecare nod v** ($v \in Q$ și v din succesorii lui u)
 - **Dacă** $d[v] > d[u] + w(u, v)$ **atunci**
 - **Eroare** ("ciclu negativ");

Exemplu Bellman-Ford (I)



- BellmanFord(G,s)
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță până la s
 - Pentru i de la 1 la $|V| - 1$ // pt. // fiecare pas de la s spre $V-s$
 - Pentru fiecare (u,v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

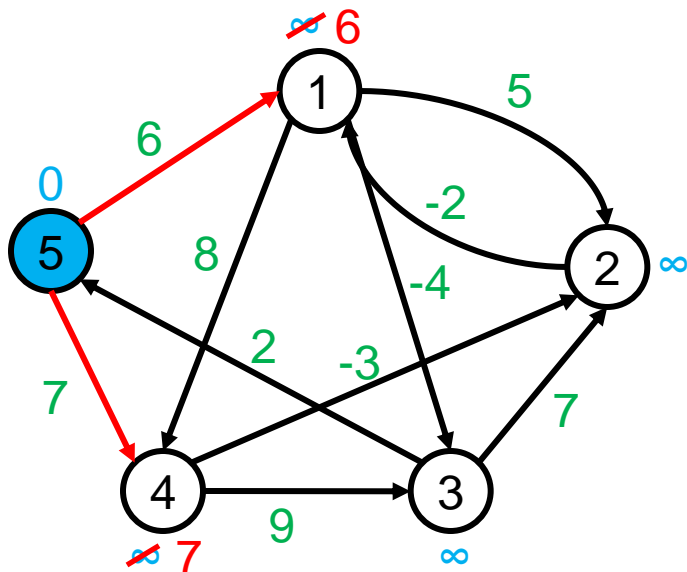
Exemplu Bellman-Ford (II)



$d[1] = d[2] = d[3] = d[4] = d[5] = \infty$
 $d[5] = 0$

- BellmanFord(G,s)
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță până la s
 - Pentru i de la 1 la $|V| - 1$ // pt. // fiecare pas de la s spre $V-s$
 - Pentru fiecare (u,v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

Exemplu Bellman-Ford (III)



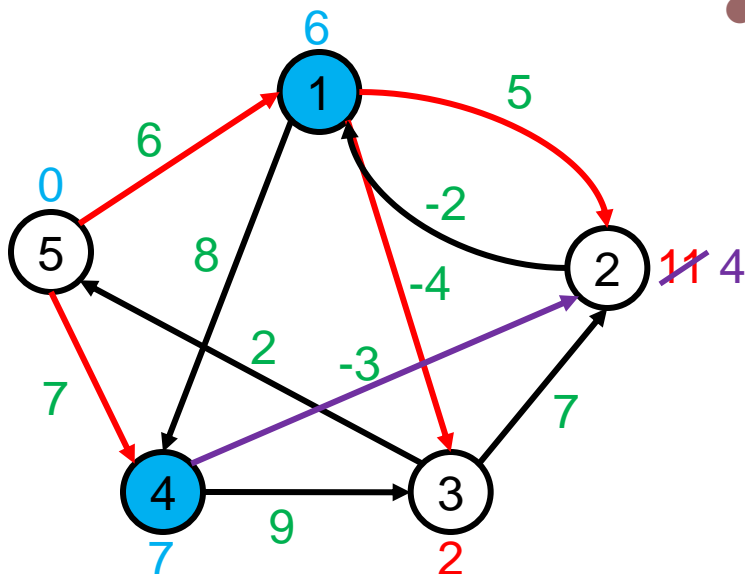
Pas 1: relaxare (5,1) și (5,4)

$d[1] = 6$, $p[1] = 5$

$d[4] = 7$, $p[4] = 5$

- BellmanFord(G, s)
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță până la s
 - Pentru i de la 1 la $|V| - 1$ // pt. // fiecare pas de la s spre $V-s$
 - Pentru fiecare (u, v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u, v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u, v)$;
 - $p[v] = u$;
 - Pentru fiecare (u, v) din E
 - Dacă $d[v] > d[u] + w(u, v)$ atunci
 - Eroare ("ciclu negativ");

Exemplu Bellman-Ford (IV)



Pas 2: relaxare (1,2) și (1,3)

$d[2] = 11$, $p[2] = 1$

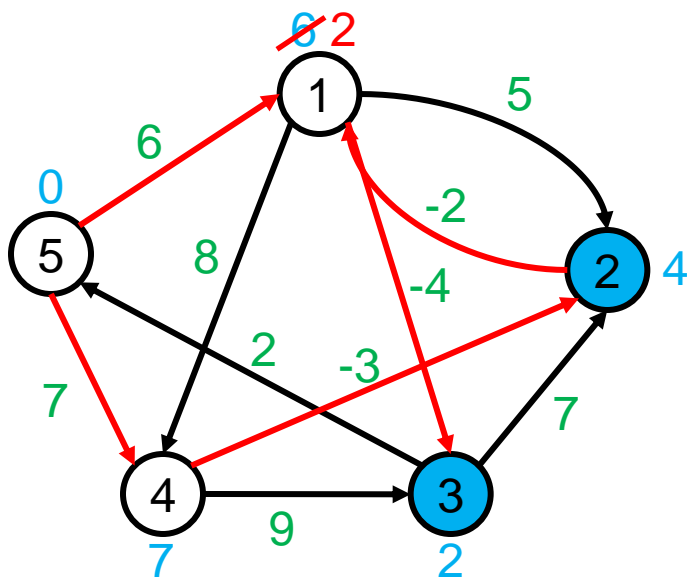
$d[3] = 2$, $p[3] = 1$

relaxare (4,2)

$d[2] = 4$, $p[2] = 4$

- BellmanFord(G,s)
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță până la s
 - Pentru i de la 1 la $|V| - 1$ // pt. // fiecare pas de la s spre $V-s$
 - Pentru fiecare (u,v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

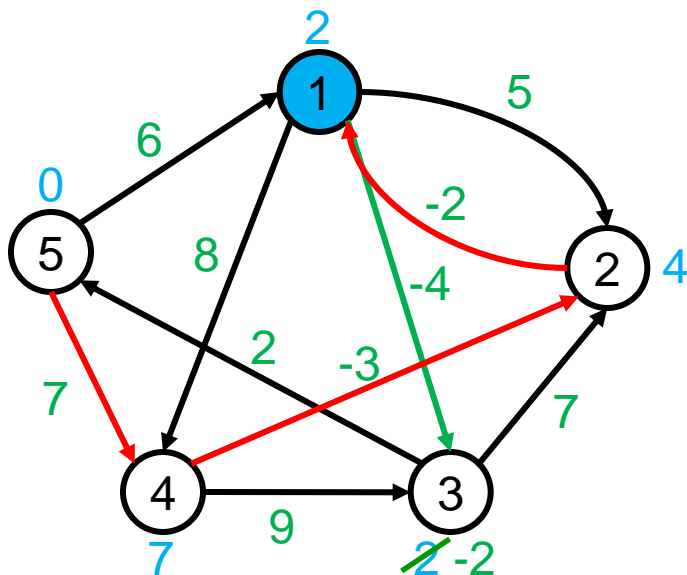
Exemplu Bellman-Ford (V)



Pas 3: relaxare (2,1)
 $d[1] = 2$, $p[1] = 2$

- BellmanFord(G,s)
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță până la s
 - Pentru i de la 1 la $|V| - 1$ // pt. // fiecare pas de la s spre $V-s$
 - Pentru fiecare (u,v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

Exemplu Bellman-Ford (VI)



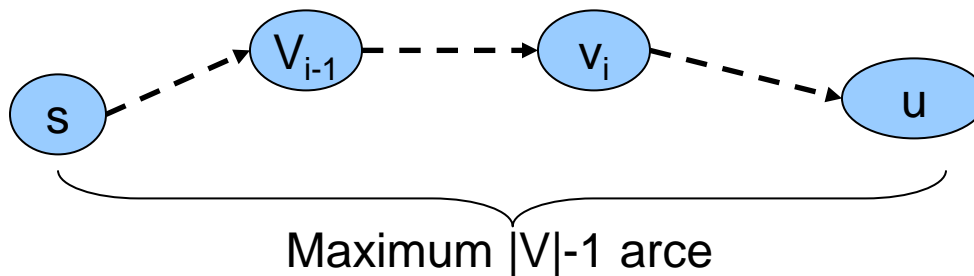
Pas 4: relaxare (1,3)
 $d[3] = -2, p[3] = 1$

- BellmanFord(G,s)
 - Pentru fiecare v din V // init
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $d[s] = 0$; // actualizare distanță până la s
 - Pentru i de la 1 la $|V| - 1$ // pt. // fiecare pas de la s spre $V-s$
 - Pentru fiecare (u,v) din E // pt. // arcele ce pleacă de la nodurile // deja considerate
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele // corespunzătoare
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - Eroare ("ciclu negativ");

Corectitudine Bellman-Ford (I)

- **Lemă 25.12:** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată; dacă G **nu conține ciclu de cost negativ** atunci după $|V| - 1$ iterații ale relaxării fiecărui arc avem $d[v] = \delta(s, v)$ pentru $\forall v \in R(s)$.
- **Dem prin inducție:**
 - Fie $s = v_0, v_1 \dots v_k = u$ un drum minim în graf cu $k \leq |V| - 1$.

La pasul i va fi relaxat arcul v_{i-1}, v_i



Corectitudine Bellman-Ford (II)

- **Demonstrăm că** în pasul i : $d[v_i] = \delta(s, v_i)$ și se menține până la sfârșit.
- P_0 : (inițializare) $\rightarrow d[s] = d[v_0] = 0 = \delta(s, s) = \delta(s, v_0)$ și conf. **Lema 25.5**, relația se menține până la sfârșit.
- $P_{i-1} \rightarrow P_i$:
 - P_{i-1} : $d[v_{i-1}] = \delta(s, v_{i-1})$,
 - În pasul i se relaxează arcul (v_{i-1}, v_i) , \Rightarrow conf. **Lema 25.7**
 $\Rightarrow d[v_i] = d[v_{i-1}] + (v_{i-1}, v_i) = \delta(s, v_{i-1}) + (v_i, v_{i-1}) = \delta(s, v_i)$.
Conf. **Lema 25.5**, relația se menține până la sfârșit.
 - Cum $i \in (1, |V|-1) \rightarrow$ relația e adevărată pentru toate nodurile accesibile din $s \rightarrow d[v] = \delta(s, v)$, $\forall v \in R(s)$.

Corectitudine Bellman-Ford (III)

- **Teorema.** $G = (V, E)$, $w : E \rightarrow \mathbb{R}$ funcție de cost asociată. Algoritmul Bellman-Ford aplicat acestui graf plecând din sursa s nu returnează EROARE dacă G nu conține cicluri negative, iar la terminare $d[v] = \delta(s, v)$ pentru $\forall v \in V$. Dacă G conține cel puțin un ciclu negativ accesibil din s , atunci algoritmul întoarce EROARE.
- **Dem:** pe baza **Lemei 25.12**.
 - Dacă \nexists ciclu negativ:
 - $d[v] = \delta(s, v) \forall v \in R(s)$
 - $d[v] = \delta(s, v) = \infty, \forall v \notin R(s)$ (inițializare)
 - $\rightarrow d[v] \leq d[u] + w(u, v) \rightarrow$ nu se întoarce eroare (conf. **Lema 25.3**)
 - Dacă \exists ciclu negativ \rightarrow în cei $|V| - 1$ pași se scad costurile drumurilor, iar în final ciclul se menține \rightarrow Eroare

Optimizări Bellman-Ford

- **Observație!**

- Dacă $d[v]$ **nu se modifică** la pasul i atunci **nu trebuie sa relaxăm** niciunul din arcele care pleacă din v la pasul $i + 1$.
- \Rightarrow păstrăm o coadă cu vârfurile modificate (o singură copie).

Bellman-Ford optimizat

- BellmanFordOpt(G,s)
 - Pentru fiecare nod v ($v \in V$)
 - $d[v] = \infty$;
 - $p[v] = \text{null}$;
 - $\text{marcat}[v] = \text{false}$; // marcăm nodurile pentru care am făcut relaxare
 - $Q = \emptyset$; // coadă cu priorități
 - $d[s] = 0$; $\text{marcat}[s] = \text{true}$; Introdu(Q, s);
 - Cât timp ($Q \neq \emptyset$)
 - $u = \text{ExtrageMin}(Q)$; $\text{marcat}[u] = \text{false}$; // extrag minimul
 - Pentru fiecare nod v ($v \in Q$ și v din succesorii lui u)
 - Dacă $d[v] > d[u] + w(u,v)$ atunci // relaxez arcele ce pleacă din u
 - $d[v] = d[u] + w(u,v)$;
 - $p[v] = u$;
 - Dacă ($\text{marcat}[v] == \text{false}$) { $\text{marcat}[v] = \text{true}$; Introdu(Q, v);}
- Observație: nu mai detectează cicluri negative!

Complexitate Bellman-Ford

- cazul defavorabil:

- Pentru i de la 1 la $|V| - 1$
- Pentru fiecare (u,v) din E
 - Dacă $d[v] > d[u] + w(u,v)$ atunci
 - $d[v] = d[u] + w(u,v);$
 - $p[v] = u;$

$$\begin{array}{r} V \\ * \\ E \\ \hline O(VE) \end{array}$$

Floyd-Warshall (Roy-Floyd)

- Algoritm prin care se calculează distanțele minime între oricare 2 noduri dintr-un graf (drumuri optime multipunct-multipunct).
- Exemplu clasic de programare dinamică.
- **Idee:** la pasul k se calculează cel mai bun cost între u și v folosind cel mai bun cost $u..k$ și cel mai bun cost $k..v$ calculat până în momentul respectiv.
- Se aplică pentru grafuri ce nu conțin cicluri de cost negativ.

Notatii

- $G = (V, E)$; $V = \{1, 2, \dots, n\}$;
- $w : V \times V \rightarrow \mathbb{R}$; $w(i, i) = 0$; $w(i, j) = \infty$ dacă $(i, j) \notin E$;
- $d^k(i, j)$ = costul drumului $i..j$ construit astfel încât drumul trece doar prin noduri din mulțimea $\{1, 2, \dots, k\}$;
- $\delta(i, j)$ = costul drumului optim $i..j$; $\delta(i, j) = \infty$ dacă $\nexists i..j$;
- $\delta^k(i, j)$ = costul drumului optim $i..j$ ce trece doar prin noduri din mulțimea $\{1, 2, \dots, k\}$; $\delta^k(i, j) = \infty$ dacă $\nexists i..j$;
- $p^k(i, j)$ = predecesorul lui j pe drumul $i..j$ ce trece doar prin noduri din mulțimea $\{1, 2, \dots, k\}$.

Teorema Floyd - Warshall

- **Teoremă:** Fie formulele de mai jos pentru calculul valorii $d^k(i,j)$, $0 < k \leq n$:
 - $d^0(i,j) = w(i,j)$;
 - $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$, pentru $0 < k \leq n$;

Atunci $d^n(i,j) = \delta(i,j)$, pentru $\forall i, j \in V$

- **Dem:**
 - Prin inducție după k dem. că $d^k(i,j) = \delta^k(i,j)$. (next slide)
 - Pt. $k = n$, $i..j$ trece prin $\forall v \in V$ si avem $d^k(i,j) \leq d^{k-1}(i,j)$, $\forall k = 1, n \rightarrow d^n(i,j) \leq d^{k-1}(i,j)$, $\forall k = 1, n$
 - Din $d^k(i,j) = \delta^k(i,j) \rightarrow d^n(i,j) = \delta^n(i,j) \leq d^{k-1}(i,j) = \delta^{k-1}(i,j)$, $\forall k = 1, n \rightarrow d^n(i,j) = \delta^n(i,j) = \delta(i,j)$

Demonstrație teorema Floyd - Warshall

- **$K = 0$** : 0 noduri intermediare $\rightarrow i..j = (i,j)$, la fel ca inițializarea $d^0(i,j) = w(i,j)$;
- **$0 < k \leq n$** : $d^{k-1}(i,j) = \delta^{k-1}(i,j) \rightarrow d^k(i,j) = \delta^k(i,j)$
 - a) **$k \notin$ drumului optim $i..j$** : drumul optim **nu se modifică**
($\delta^{k-1}(i,j) = \delta^k(i,j) \leq \delta^{k-1}(i,k) + \delta^{k-1}(k,j)$)
 - $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$
 $d^k(i,j) = \min\{\delta^{k-1}(i,j), \delta^{k-1}(i,k) + \delta^{k-1}(k,j)\} = \delta^{k-1}(i,j) = \delta^k(i,j)$
 - b) **$k \in$ drumului optim $i..j$** : $i..j$ se descompune în $i..k$ și $k..j$ optime
($\delta^{k-1}(i,k) = d^{k-1}(i,k)$ și $\delta^{k-1}(k,j) = d^{k-1}(k,j)$) și
 $\delta^k(i,j) = \delta^{k-1}(i,k) + \delta^{k-1}(k,j)$.
 - $i..j$ optim $\rightarrow \delta^k(i,j) \leq \delta^{k-1}(i,j)$
 - $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$
 - $d^k(i,j) = \min\{\delta^{k-1}(i,j), \delta^{k-1}(i,k) + \delta^{k-1}(k,j)\} = \delta^{k-1}(i,k) + \delta^{k-1}(k,j) = \delta^k(i,j)$

Algoritm Floyd-Warshall

I Floyd-Warshall(G)

I ● Pentru i de la 1 la n

I ● Pentru j de la 1 la n // inițializări

- $d^0(i,j) = w(i,j)$
- Dacă $(w(i,j) == \infty)$
 - $p^0(i,j) = \text{null};$
- Altfel $p^0(i,j) = i;$

I ● Pentru k de la 1 la n

I ● Pentru i de la 1 la n

I ● Pentru j de la 1 la n

- Dacă $(d^{k-1}(i,j) > d^{k-1}(i,k) + d^{k-1}(k,j))$ // determinăm minimul
 - $d^k(i,j) = d^{k-1}(i,k) + d^{k-1}(k,j)$
 - $p^k(i,j) = p^{k-1}(k,j);$ // și actualizăm părintele
- Altfel
 - $d^k(i,j) = d^{k-1}(i,j)$
 - $p^k(i,j) = p^{k-1}(i,j);$

Complexitate?

$O(V^3)$

Complexitate
spațială?

$O(V^3)$

Observație

- Putem folosi o singură matrice în loc de n ?
- **Problemă:** în pasul k , pentru $k < i$ și $k < j$, $d(i,k)$ și $d(k,j)$ folosite la calculul $d(i,j)$ sunt $d^k(k,j)$ și $d^k(i,k)$ în loc de $d^{k-1}(k,j)$ și $d^{k-1}(i,k)$. Dacă dem. că $d^k(k,j) = d^{k-1}(k,j)$ și $d^k(i,k) = d^{k-1}(i,k)$, atunci putem folosi o singură matrice.
- **Dar:**
 - $d^k(k,j) = d^{k-1}(k,k) + d^{k-1}(k,j) = d^{k-1}(k,j)$
 - $d^k(i,k) = d^{k-1}(i,k) + d^{k-1}(k,k) = d^{k-1}(i,k)$
- → Algoritm modificat pentru a folosi o singura matrice → **complexitate spațială: $O(n^2)$.**

Algoritm Floyd-Warshall

Floyd-Warshall2(G)

- Pentru i de la 1 la n

- Pentru j de la 1 la n // inițializări

- $d(i,j) = w(i,j)$
 - Dacă $(w(i,j) == \infty)$
 - $p(i,j) = \text{null};$
 - Altfel $p(i,j) = i;$

- Pentru k de la 1 la n

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- Dacă $(d(i,j) > d(i,k) + d(k,j))$ // determinăm minimul
 - $d(i,j) = d(i,k) + d(k,j)$
 - $p(i,j) = p(k,j);$ // și actualizăm părintele

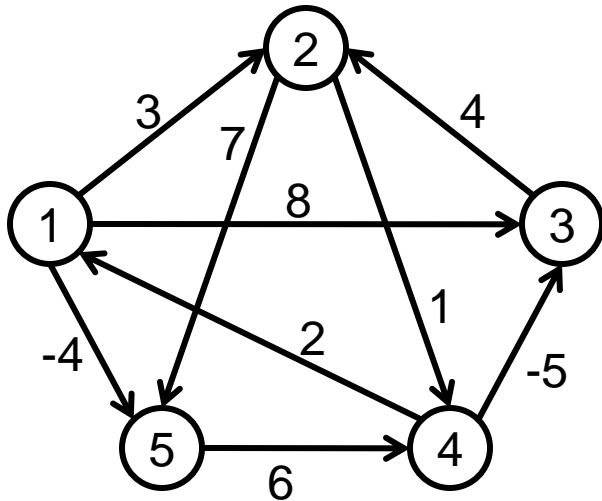
Complexitate?

$O(V^3)$

Complexitate
spațială?

$O(V^2)$

Exemplu (I)



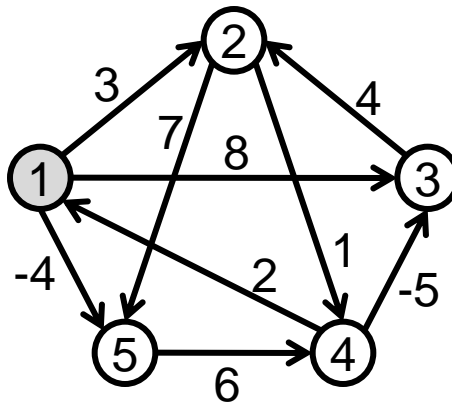
$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \text{nil} & 4 & \text{nil} & \text{nil} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (II)

$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \mathbf{5} & -5 & 0 & \mathbf{-2} \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



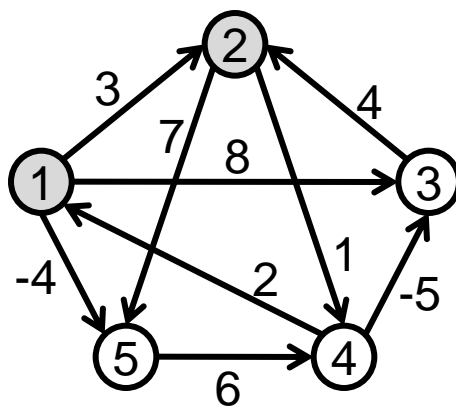
$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \text{nil} & 4 & \text{nil} & \text{nil} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & \mathbf{1} & 4 & \text{nil} & \mathbf{1} \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (III)

$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



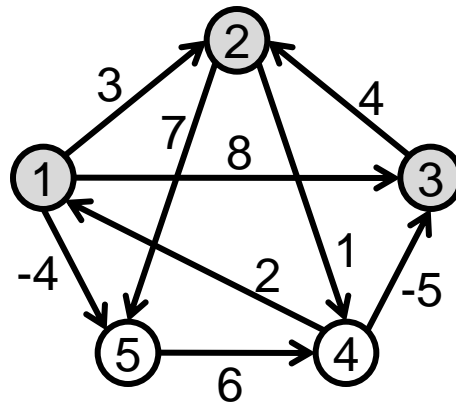
$$p = \begin{bmatrix} \text{nil} & 1 & 1 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & \text{nil} & \text{nil} \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (IV)

$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$



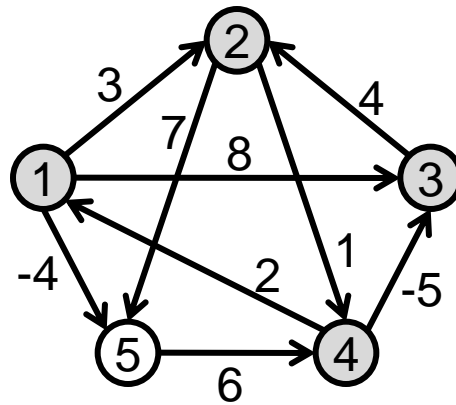
$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 1 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 3 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

Exemplu (V)

$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$



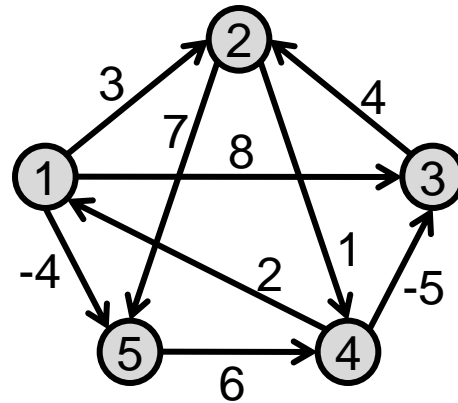
$$p = \begin{bmatrix} \text{nil} & 1 & 1 & 2 & 1 \\ \text{nil} & \text{nil} & \text{nil} & 2 & 2 \\ \text{nil} & 3 & \text{nil} & 2 & 2 \\ 4 & 3 & 4 & \text{nil} & 1 \\ \text{nil} & \text{nil} & \text{nil} & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & 1 & 4 & 2 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

Exemplu (VI)

$$D = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

$$D = \begin{bmatrix} 0 & \color{red}{1} & \color{red}{-3} & \color{red}{2} & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$



$$p = \begin{bmatrix} \text{nil} & 1 & 4 & 2 & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

$$p = \begin{bmatrix} \text{nil} & \color{red}{3} & \color{red}{4} & \color{red}{5} & 1 \\ 4 & \text{nil} & 4 & 2 & 1 \\ 4 & 3 & \text{nil} & 2 & 1 \\ 4 & 3 & 4 & \text{nil} & 1 \\ 4 & 3 & 4 & 5 & \text{nil} \end{bmatrix}$$

Închiderea tranzitivă (I)

- Fie $G = (V, E)$. **Închiderea tranzitivă** a lui E e un $G^* = (V, E^*)$, unde

$$E^*(i, j) = \begin{cases} 1, & \text{dacă } \exists i..j \\ 0, & \text{dacă } \nexists i..j \end{cases}$$

- Poate fi determinată prin modificarea algoritmului Floyd-Warshall:
 - $\min \Rightarrow$ operatorul boolean sau (\vee)
 - $+$ \Rightarrow operatorul boolean și (\wedge)

Închiderea tranzitivă (II)

Închidere_tranzitivă(G)

- Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^*(i,j) = ((i,j) \in E) \vee (i = j)$ // inițializări
- Pentru k de la 1 la n
 - Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^*(i,j) = E^*(i,j) \vee (E^*(i,k) \wedge E^*(k,j))$

Complexitate?

$O(V^3)$

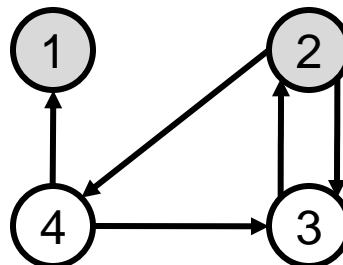
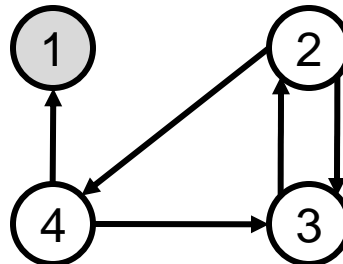
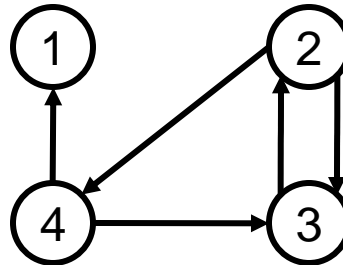
Complexitate spațială?

$O(V^2)$

Exemplu (I)

Închidere_tranzitivă(G)

- Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^*(i,j) = (i,j) \in E \vee i = j$
// inițializări
- Pentru k de la 1 la n
 - Pentru i de la 1 la n
 - Pentru j de la 1 la n
 - $E^*(i,j) = E^*(i,j) \vee (E^*(i,k) \wedge E^*(k,j))$



$$T^{(0)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$T^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

$$T^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & \textcolor{red}{1} \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Exemplu (II)

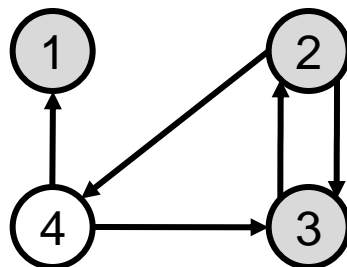
Închidere_tranzitivă(G)

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- $E^*(i,j) = (i,j) \in E \vee i = j$

// inițializări



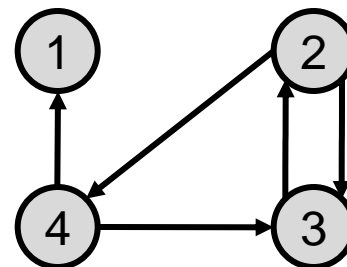
$$T^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & \color{red}{1} & 1 & 1 \end{bmatrix}$$

- Pentru k de la 1 la n

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- $E^*(i,j) = E^*(i,j) \vee (E^*(i,k) \wedge E^*(k,j))$



$$T^{(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \color{red}{1} & 1 & 1 & 1 \\ \color{red}{1} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Exemplu (II)

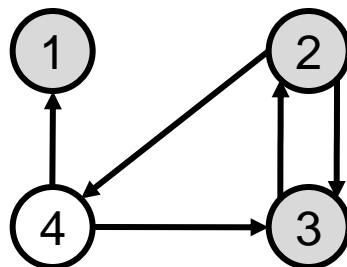
Închidere_tranzitivă(G)

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- $E^*(i,j) = (i,j) \in E \vee i = j$

// inițializări



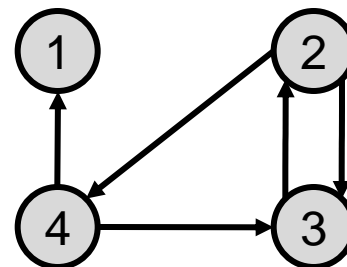
$$T^{(3)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & \color{red}{1} & 1 & 1 \end{bmatrix}$$

- Pentru k de la 1 la n

- Pentru i de la 1 la n

- Pentru j de la 1 la n

- $E^*(i,j) = E^*(i,j) \vee (E^*(i,k) \wedge E^*(k,j))$



$$T^{(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \color{red}{1} & 1 & 1 & 1 \\ \color{red}{1} & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

Algoritmul lui Johnson

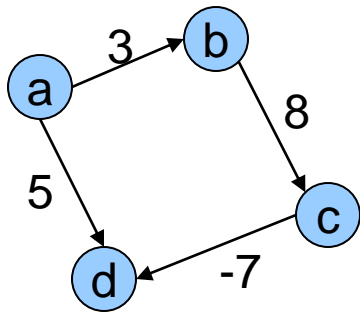
- Pentru grafuri rare.
- Folosește liste de adiacență.
- Bazat pe Dijkstra și Bellman-Ford.
- Complexitate: $O(V^2 \log V + VE)$
 - Mai bună decât Floyd-Warshall pentru grafuri rare.

Idee algoritm Johnson

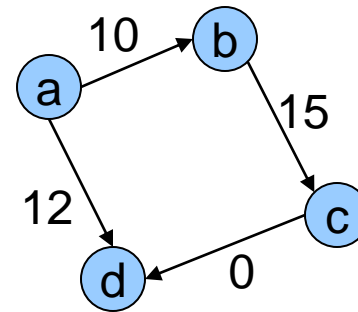
- Dacă graful are **numai arce pozitive**:
 - se aplică **Dijkstra pentru fiecare nod** \Rightarrow cost $V^2 \log V$.
- Altfel **se calculează costuri pozitive pentru fiecare arc menținând proprietățile**:
 - $w_1(u,v) \geq 0, \forall (u,v) \in E$;
 - p este drum minim utilizând $w \Leftrightarrow p$ este drum minim utilizând w_1 .

Construcție w_1 (I)

- **Idee 1:** identificare arcul cu cel mai mic cost – c ; adunare la costul fiecărui arc valoarea c ;



$$\text{cost}(a..b..d) < \text{cost}(a,d)$$



$$\text{cost}(a..b..d) > \text{cost}(a,d)$$

Nu funcționează!!!!

Construcție w_1 (II)

- **Idee 2:** $w_1(u,v) = w(u,v) + h(u) - h(v)$;
- unde $h : V \rightarrow \mathbb{R}$;
- se adaugă un nod s ;
- se unește s cu toate nodurile grafului prin arce de cost 0;
- se aplică BF pe acest graf $\Rightarrow h(v) = \delta(s,v)$;
- $\rightarrow w_1(u,v) = w(u,v) + h(u) - h(v)$.

Algoritm Johnson

● Johnson(G)

- $G' = (V', E')$;
- $V' = V \cup \{s\}$; // adăugăm nodul s
- $E' = E \cup (s, u), \forall u \in V; w(s, u) = 0$; // și îl legăm de toate nodurile
- Dacă $BF(G', s)$ e fals // aplic BF pe G'
 - Eroare “ciclu negativ”
- Altfel
 - Pentru fiecare $v \in V$
 - $h(v) = \delta(s, v)$; // calculat prin BF
 - Pentru fiecare $(u, v) \in E$
 - $w_1(u, v) = w(u, v) + h(u) - h(v)$ // calculez noile costuri pozitive
 - Pentru fiecare $(u \in V)$
 - $Dijkstra(G, w_1, u)$ // aplic Dijkstra pentru fiecare nod
 - Pentru fiecare $(v \in V)$
 - $d(u, v) = \delta_1(u, v) + h(v) - h(u)$ // calculez costurile pe graful inițial

Exemplu (I)

BellmanFord(G, s) // $G=(V,E), s=sursa$

Pentru fiecare v din V // inițializări

$d[v] = \infty$; $p[v] = \text{null}$;

$d[s] = 0$; // actualizare distanță de la s la s

Pentru i de la 1 la $|V| - 1$ // se construiesc drumuri optime de dimensiune i

Pentru fiecare (u,v) din E // pentru arcele ce pleacă de la nodurile deja considerate

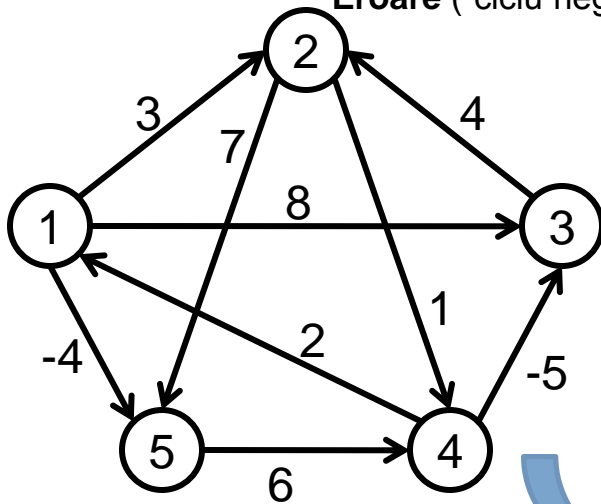
Dacă $d[v] > d[u] + w(u,v)$ atunci // se relaxează arcele corespunzătoare

$d[v] = d[u] + w(u,v)$; $p[v] = u$;

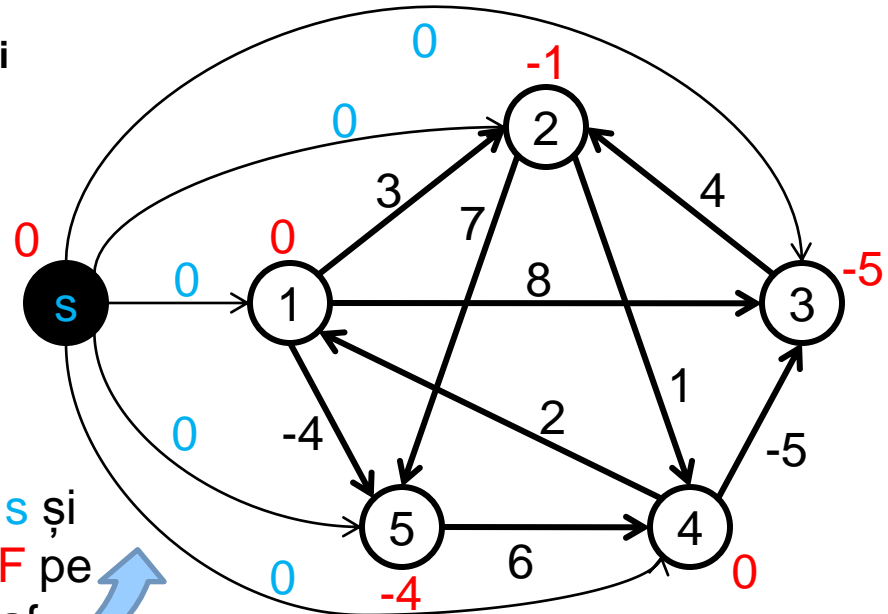
Pentru fiecare (u,v) din E

Dacă $d[v] > d[u] + w(u,v)$ atunci

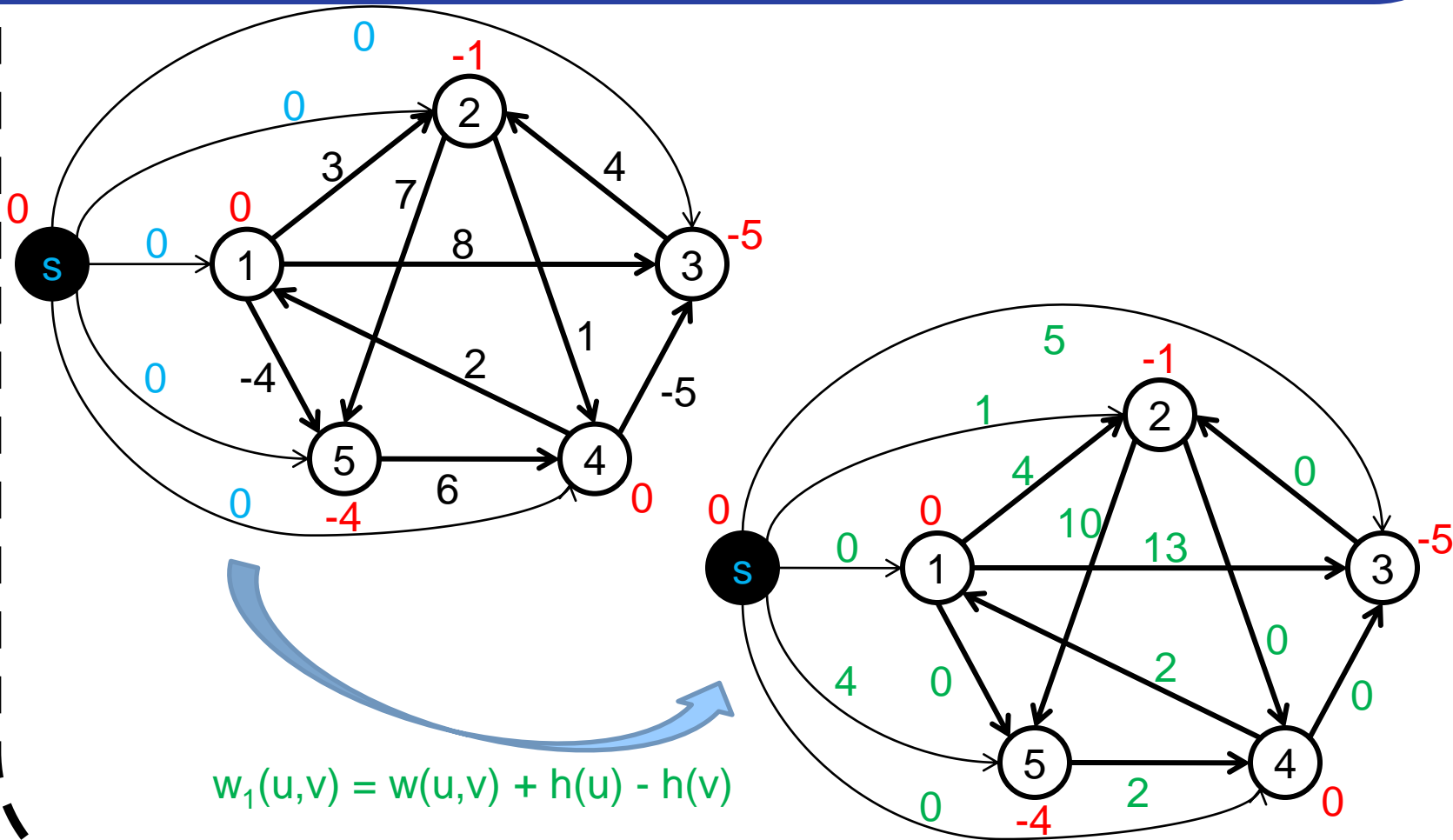
Eroare ("ciclu negativ");



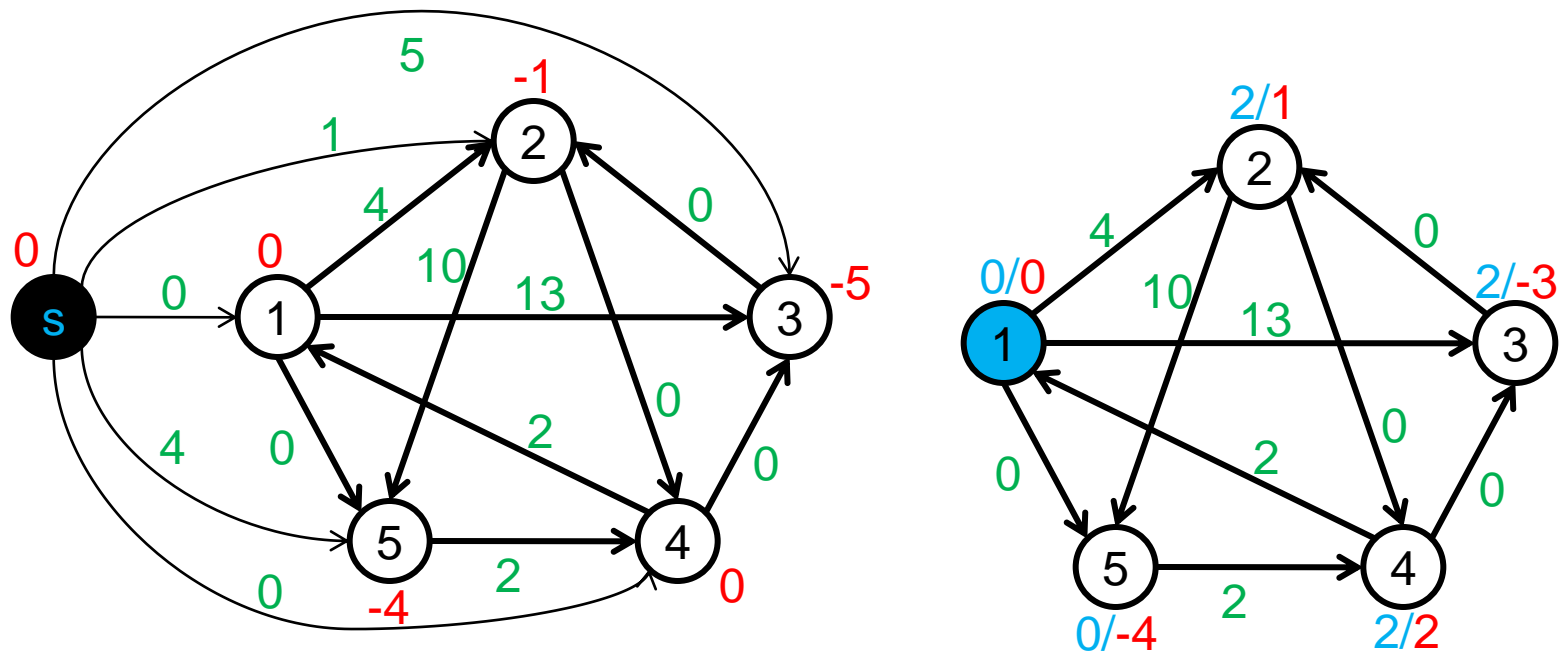
Adaug s și
aplic BF pe
noul graf.



Exemplu (II)



Exemplu (III)



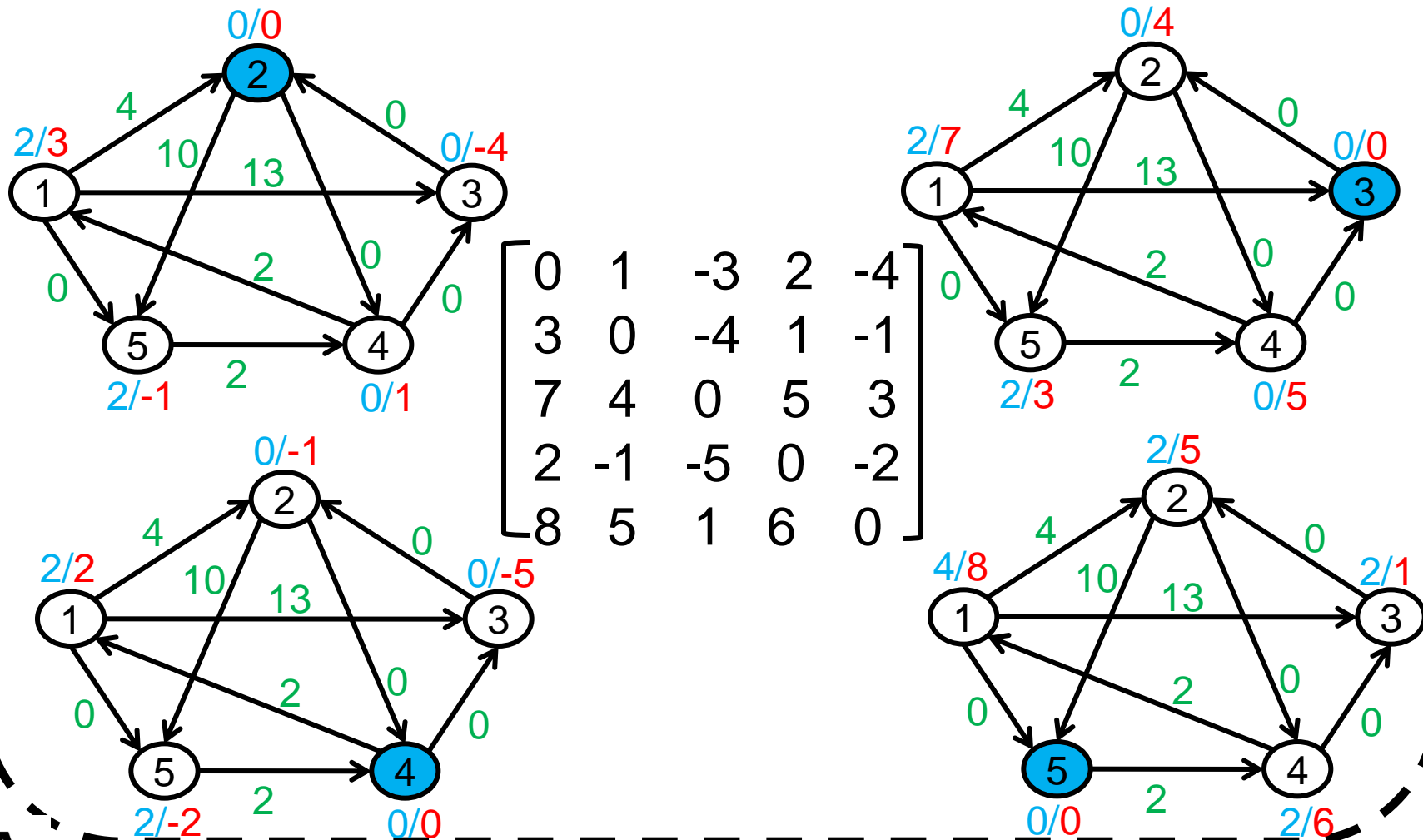
Eliminăm s

Aplicăm Dijkstra din fiecare nod ($\delta_1(u,v)$).

Refacem distanțele:

$$d(u,v) = \delta_1(u,v) + h(v) - h(u)$$

Exemplu (IV)



Concluzii Floyd-Warshall & Johnson

- Algoritmi ce găsesc drumurile minime între oricare 2 noduri din graf.
- Funcționează pe grafuri cu arce ce au costuri negative (dar care nu au cicluri de cost negativ).
- Floyd-Warshall e optim pentru grafuri dese.
- Johnson e mai bun pentru grafuri rare.

ÎNTREBĂRI?

Bibliografie curs 10

- [1] http://monalisa.cacr.caltech.edu/monalisa__Service_Applications__Monitoring_VRVS.html
- [2] <http://www.cobblestoneconcepts.com/ucgis2summer2002/guo/guo.html>
- [3] Giumale – Introducere in Analiza Algoritmilor cap. 5.5
- [4] R. Sedgewick, K Wayne – curs de algoritmi Princeton 2007
www.cs.princeton.edu/~rs/AlgsDS07/ 01UnionFind si 14MST
- [5] http://www.pui.ch/phred/automated_tag_clustering/
- [6] Cormen – Introducere în Algoritmi cap. 24