Proiectarea Algoritmilor

Curs 8 – 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*,

Obiective

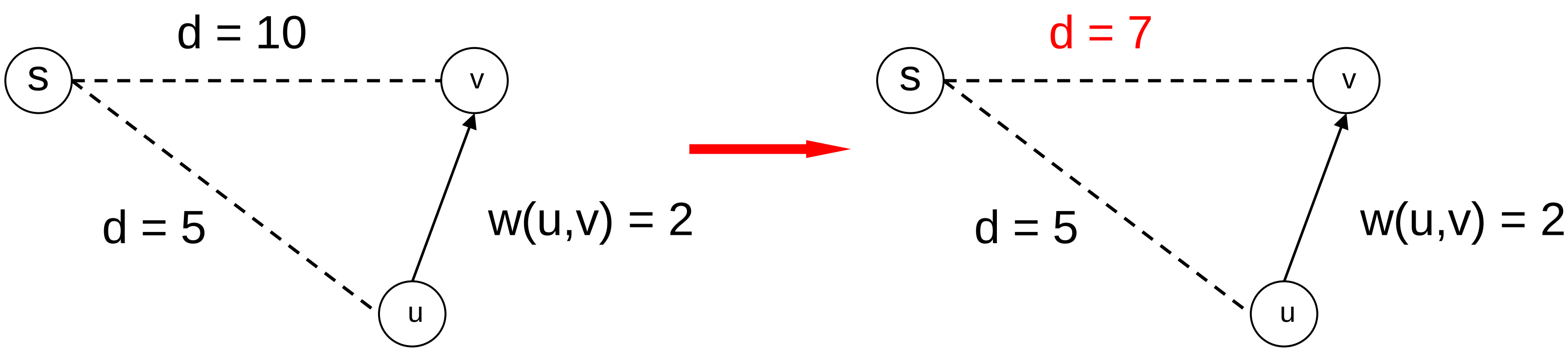
- “Descoperirea” algoritmilor de identificare a drumurilor de cost minim.
- Recunoașterea caracteristicilor acestor algoritmi.

Reminder (I)

- $G = (V,E)$;
- $s \in V$ – nodul sursă;
- $w : E \rightarrow \Re$ funcție de cost asociată arcelor grafului;
- 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 $v \notin R(u)$
  - $\delta(u,v) = \Sigma w(x,y)$, $(x,y) \in$ 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



Algoritmul lui Dijkstra (I)

- Folosește o coadă de priorități în care se adaugă nodurile în funcție de distanța cunoscută în momentul respectiv de la s până la nod.

- Se folosește **NUMAI** pentru costuri pozitive (w(u,v) > 0, $\forall$u,v$\in$V).

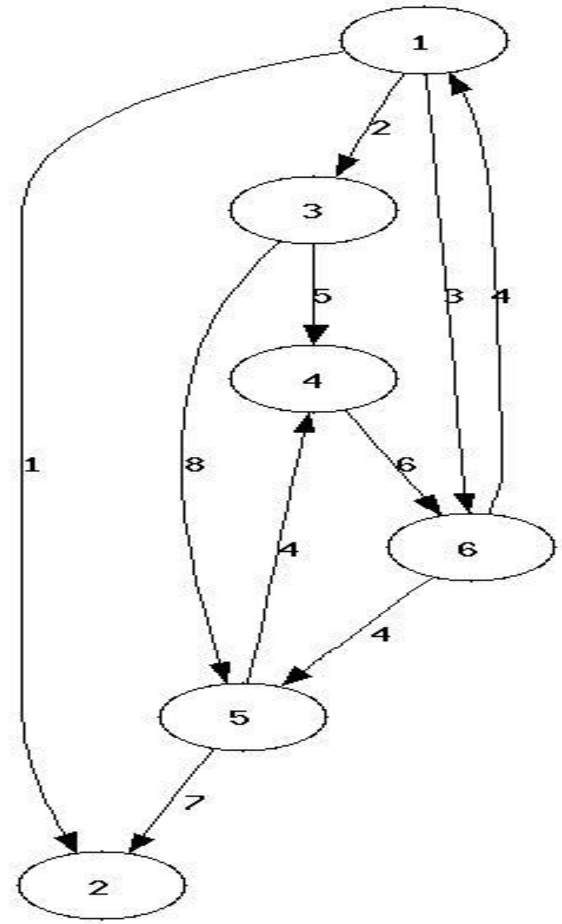
- Dijkstra_generic (G,s)
  - V = nodurile lui G
  - **Cât timp** (V != ∅)
    - u = nod din V cu d[u] min
    - V = V - {u}
    - **Pentru fiecare** (v ∊ succesorii lui u) relaxare_arc(u,v)

    // optimizare drum s..v pentru v ∊ succesorilor lui u

Algoritmul lui Dijkstra (II)

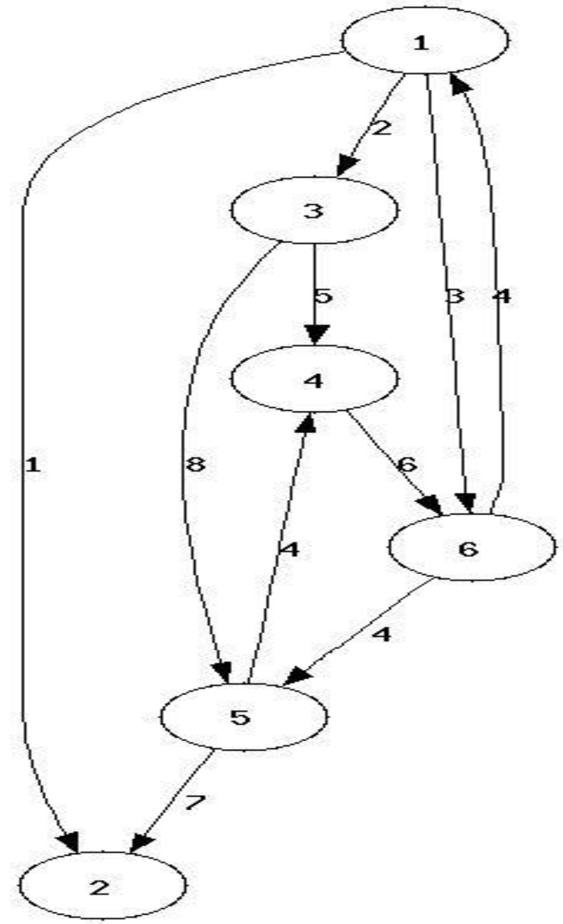
- Dijkstra(G,s)
  - **Pentru fiecare nod u** (u ∊ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] minim
    - // Q = Q - {u} – se execută în cadrul lui ExtrageMin
    - **Pentru fiecare nod v** (v ∊ 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

Exemplu (I)



Exemplu (II)

- d[1] = 0;
- (1): d[2] = 1; d[3] = 2; d[6] = 3;
- (2): d[4] = 7; d[5] = 10;
- (3): d[5] = 7;



- Dijkstra(G,s)
  - **Pentru fiecare nod u** (u ∊ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] // minim
    - // Q = Q - {u} – se execută în cadrul lui ExtrageMin
    - **Pentru fiecare nod v** (v ∊ 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 Dijkstra

- Depinde de ExtrageMin – coadă cu priorități.
- Operații ce trebuie realizate pe coadă + frecvența lor:
  - insert – V;
  - delete – V;
  - conține? – V;
  - micșorează_val – E;
  - este_vidă? – V.

- Dijkstra(G,s)
  - **Pentru fiecare nod u** ($u \in V$)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] minim
    - // Q = Q - {u} – se execută in cadrul lui ExtrageMin
    - **Pentru fiecare nod v** ($v \in Q$ si 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 // si părintele

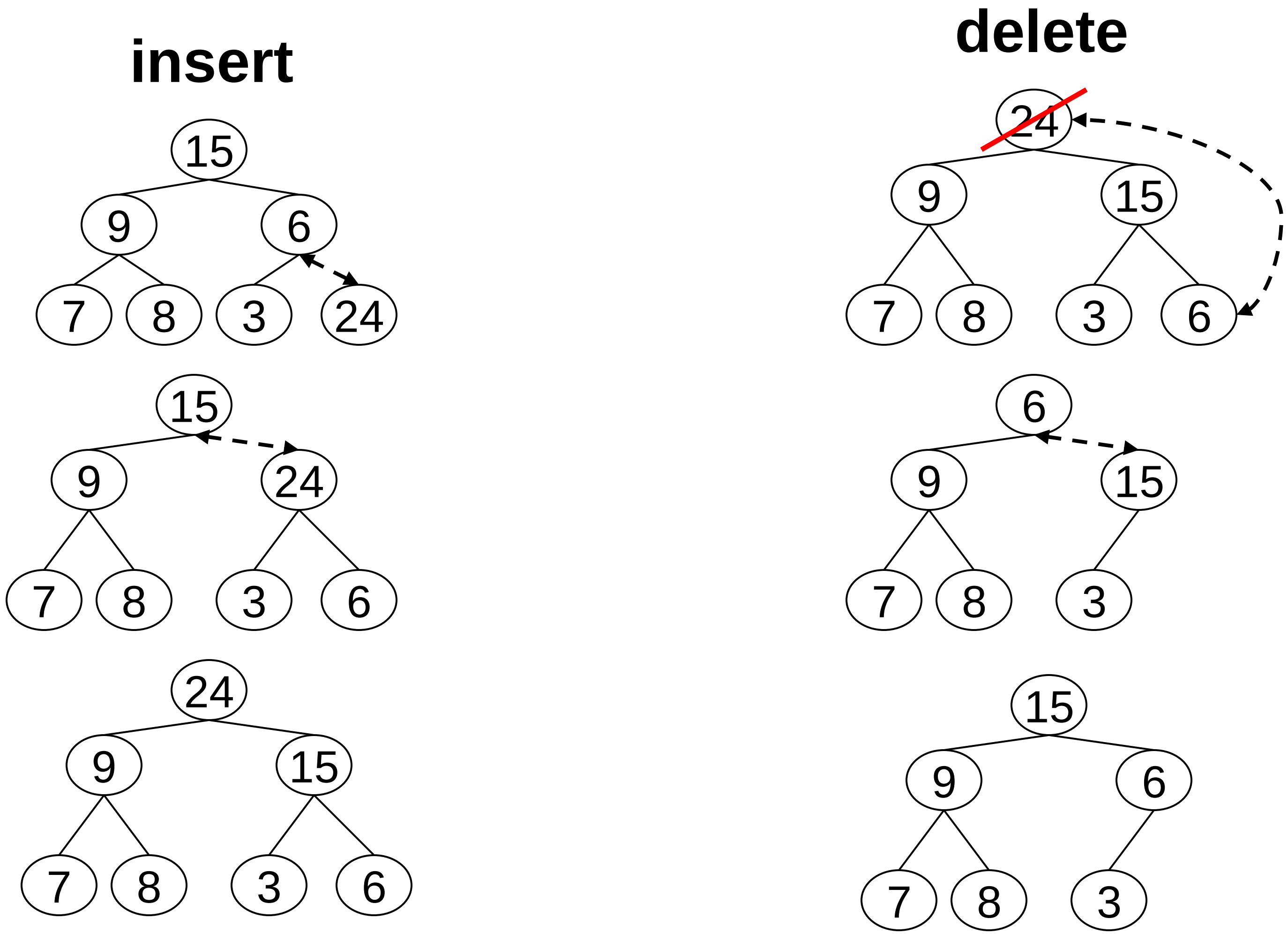
Implementare cu vectori

- Costuri:
  - insert – 1 * V = V;
  - delete – V * V = V^2 (necesită căutarea minimului);
  - conține? – 1 * V = V;
  - micșorează_val – 1 * E = E;
  - este_vidă? – 1 * V = V;
- Cea mai bună metodă pentru grafuri “dese” ($E \approx V^2$)!

Implementare cu heap binar

- Heap binar – structură de date de tip arbore binar + 2 constrângeri:
  - Fiecare nivel este complet; ultimul se umple de la stânga la dreapta;
  - $\forall u \in$ Heap; u $\geq$ răd(st(u)) și u $\geq$ răd(dr(u)) unde $\geq$ este o relație de ordine pe mulțimea pe care sunt definite elementele heapului.

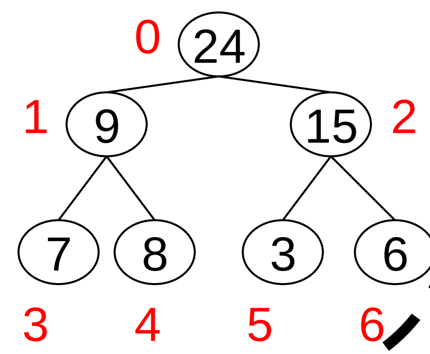
Operatii pe Heap Binar



Implementare Heap Binar

- Implementare folosind vectori.
- Poziție[i] = unde se găsește în indexul de valori elementul de pe poziția i din heap.
- Reverse[i] = unde se găsește în heap elementul de pe poziția i din valoare.
- Implementare disponibila la [3].

Index	0	1	2	3	4	5	6
Valoare	7	6	15	8	24	9	3
Poziție	4	5	2	0	3	6	1
Reverse	3	6	2	4	0	1	5



Heap Binar

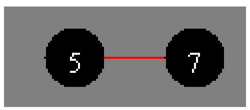
- Costuri:
  - insert – logV * V = VlogV;
  - delete – logV * V = VlogV;
  - conține? – 1 * V = V;
  - micșorează_val – logV * E = ElogV;
  - este_vidă? – 1 * V = V.
- Eficient dacă graful are arce puține comparativ cu numărul de noduri.

Heap Fibonacci

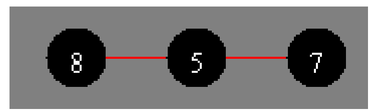
- Poate fi format din mai mulți arbori.
- Cheia unui părinte ≤ cheia oricărui copil.
- Fiind dat un nod u și un heap H:
  - p(u) – părintele lui u;
  - copil(u) – legătura către unul din copiii lui u;
  - st(u), dr(u) – legătura la frații din stânga și din dreapta (cei de pe primul nivel sunt legați între ei astfel);
  - grad(u) – numărul de copii ai lui u;
  - min(H) – cel mai mic nod din H;
  - n(H) – numărul de noduri din H.

Operatii Heap Fibonacci

- Inserare nod – O(1)
  - construiește un nou arbore cu un singur nod



(insert 8) →



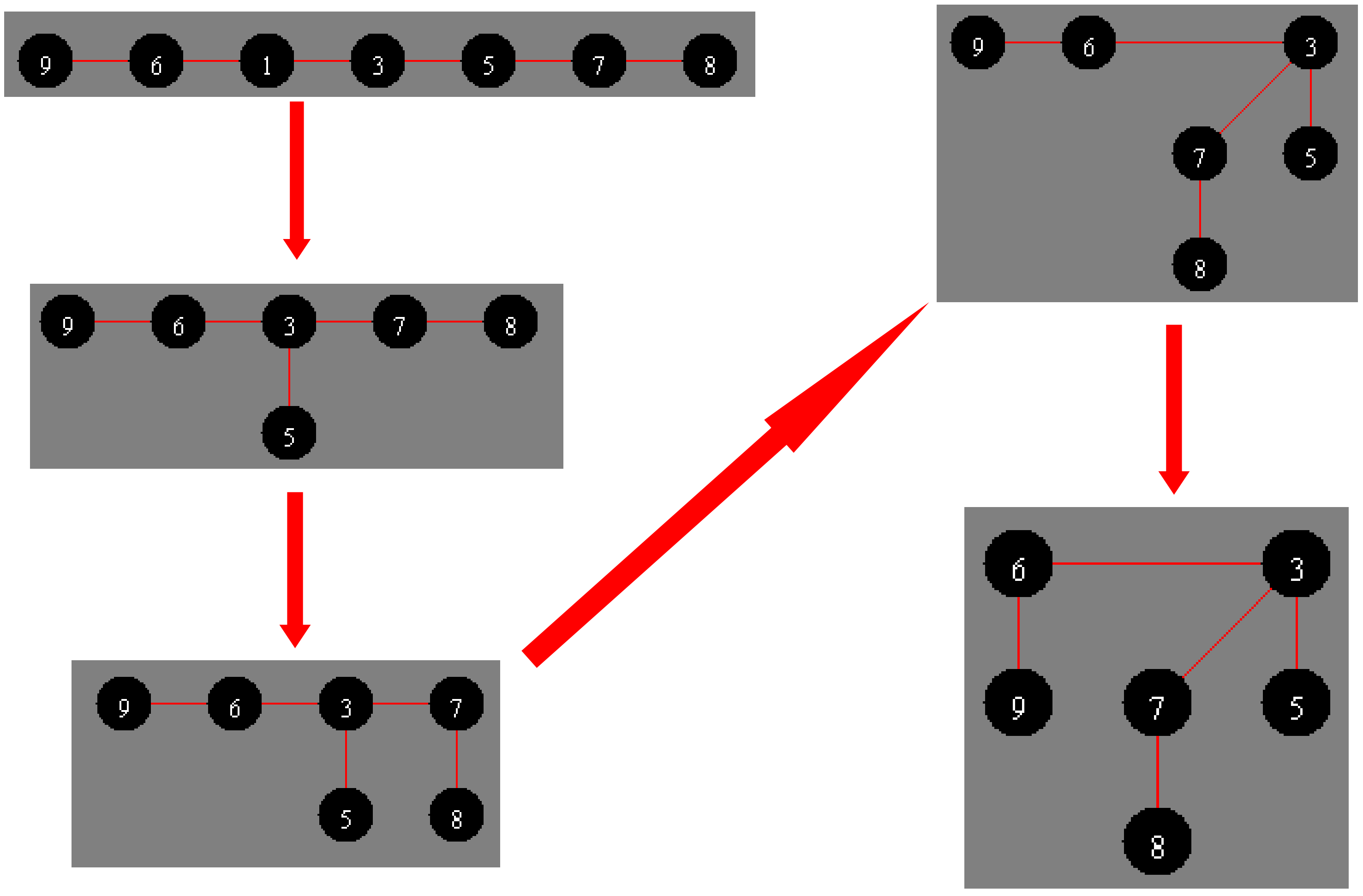
- Min – accesibil direct - min(H) – O(1)

- ExtrageMin O(logn) – cost amortizat!
  - Mută copiii minimului pe prima coloană;
  - **Consolidează** heap-ul.

Operatii Heap Fibonacci

- Consolidare Heap
  - Cât timp există 2 arbori cu grade egale Arb(x) și Arb(y), x < y:
    - Arb(y) adăugat ca și copil al lui x;
    - grad[x] ++;
- Applet și implementare disponibile la [4].

Consolidare Heap



Costuri Heap Fibonacci

- Costuri:
  - insert – 1 * V = V;
  - delete – logV * V = VlogV(amortizat!);
  - micșorează_val – 1 * E = E;
  - este_vidă? – 1 * V = V.

- Cea mai rapidă structură dpdv teoretic.

Concluzii Dijkstra (I)

Complexitate?

Vectori - $O(V^2)$

HB - O(E logV)

HF - O(V log V+E)

- Dijkstra(G,s)
  - **Pentru fiecare nod u** (u ∈ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] minim
    - // Q = Q - {u} – se execută în cadrul lui ExtrageMin
    - **Pentru fiecare nod v** (v ∈ 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

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 → $\Re$ funcție de cost asociată. Fie p = $v_1v_2...v_k$ 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 → $\Re$ 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 → $\Re$ 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: $G = (V,E)$, $w : E \rightarrow \Re$ 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: $G = (V,E)$, $w : E \rightarrow \Re$ 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 $d[v] = \delta(s,v)$.

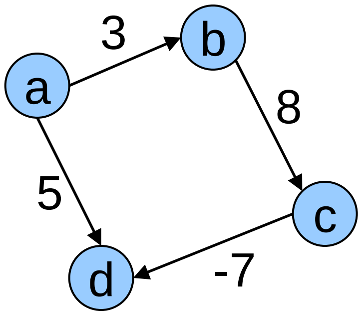
Corectitudine Dijkstra

- Teoremă. $G = (V,E)$, $w : E \rightarrow \Re$ 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 $d[u] = \delta(s,u) = \infty$. => 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 $y \in Q$.
  - Cum u e primul nod pt. care $d[u] \neq \delta(s,u)$ => $d[x] = \delta(s,x)$ 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 => $d[y] = \delta(s,y) \leq \delta(s,u) \leq d[u]$ (conf. Lema 25.5).
  - Cum $y \in Q$ la momentul scoaterii lui u din Q => $d[u] \leq d[y]$
  - => $d[y] = \delta(s,y) = \delta(s,u) = d[u]$ – Contrazice ipoteza! => $d[u] = \delta(s,u)$ și conf. Lema 25.5, egalitatea se menține și ulterior.

Problemă Dijkstra

- Exemplu rulare:
  - d[a] = 0; d[b] = d[c] = d[d] = ∞
  - d[b] = 3; d[d] = 5;
  - d[c] = 11;



  - **d este extras din coadă!** În momentul extragerii din coadă distanța pană 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 ⇒ $327.25.
- 1 oz. gold ⇒ £208.10 ⇒ ⇒ $327.00. [208.10 × 1.5714]
- 1 oz. gold ⇒ 455.2 Francs ⇒ 304.39 Euros ⇒ $327.28.

[455.2 × .6677 × 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.

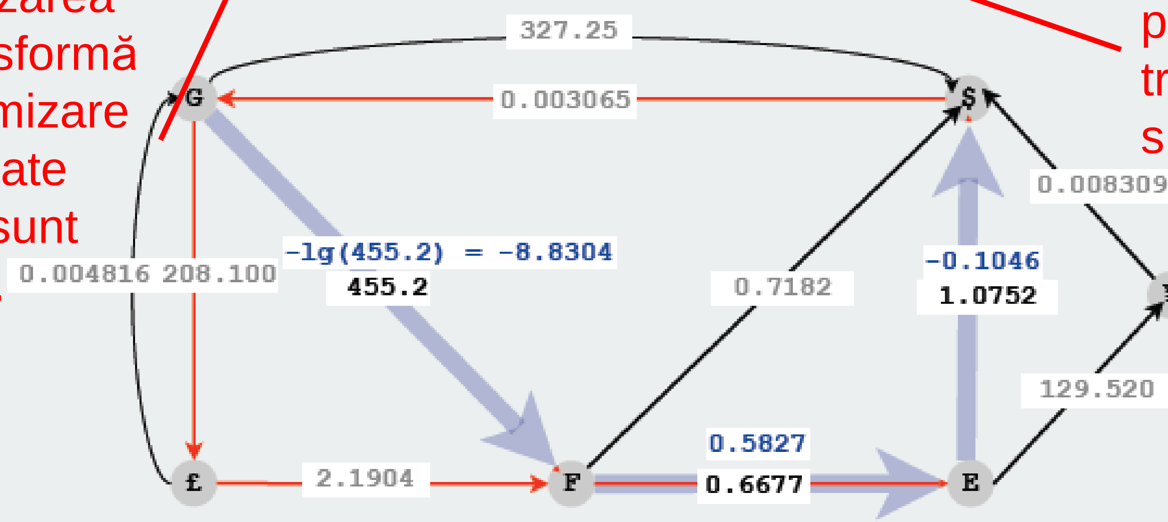
*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

Exemplu practic – arce de cost negativ (III)

Reduce to shortest path problem by taking logs

- Let 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.

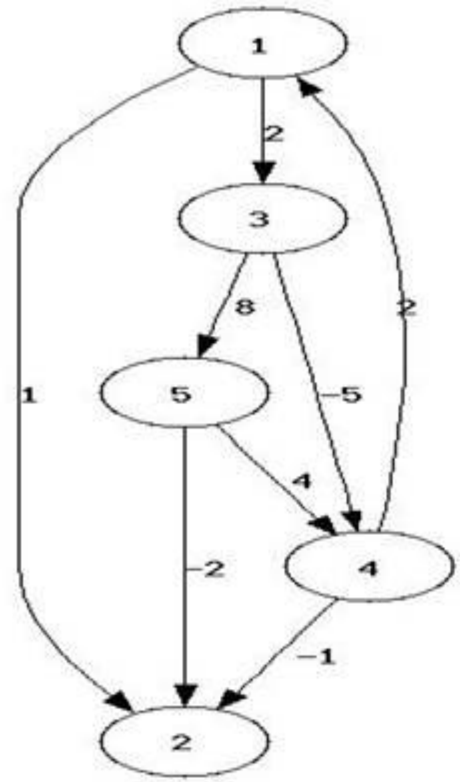


Prin logaritmare produsul se transformă în sumă.

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

Cicluri de cost negativ

$$\delta(u, v) = \begin{cases} \Sigma\, w(x,y),\ (x,y) \in u..v \\ \text{(u..v fiind drumul optim);} \\ \infty, \text{ dacă nu există drum u..v.} \end{cases}$$



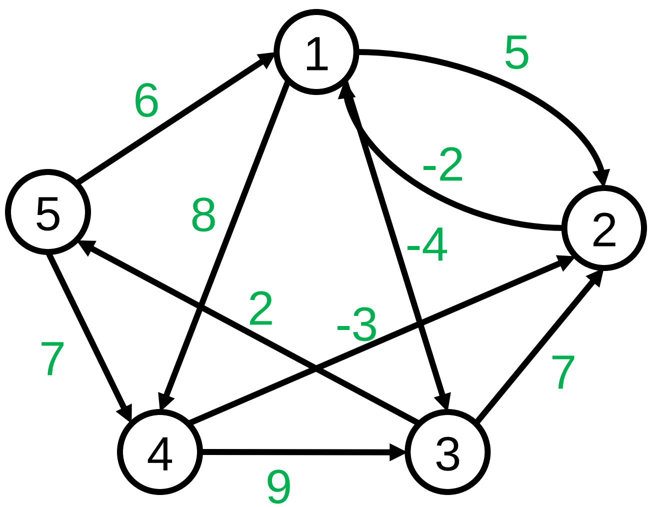
- Dacă există pe drumul u..v un ciclu de cost negativ x..y →
  - δ(u,v) = δ(u,v) + cost(x..y) < δ(u,v)
  - → valoarea lui δ(u,v) va scădea continuu → costul este -∞
  - → δ(u,v) = -∞

1-3-4 ciclu de cost negativ(-1)→ toate costurile din graf sunt -∞

Algoritmul Bellman-Ford

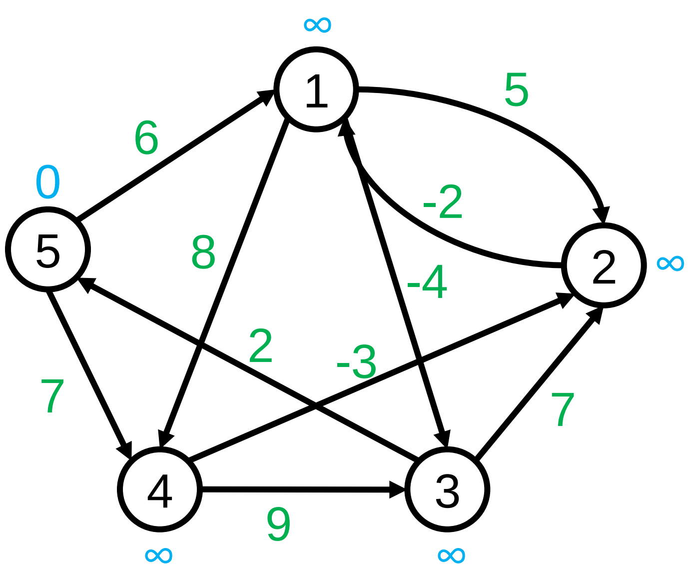
- BellmanFord(G,s) // G=(V,E),s=sursa
  - **Pentru fiecare** nod v (v ∈ V) // inițializări
    - d[v] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță de la s la s
  - **Pentru** ***i*** **de la** *1* **la** |*V*| *-1* // pentru fiecare pas pornind din s
    // spre restul nodurilor se încearcă construcția unor drumuri
    // optime de dimensiune i
    - **Pentru fiecare nod v** (v ∈ Q si v din succesorii lui u)
      // 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 ∈ Q si 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] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță pană 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");

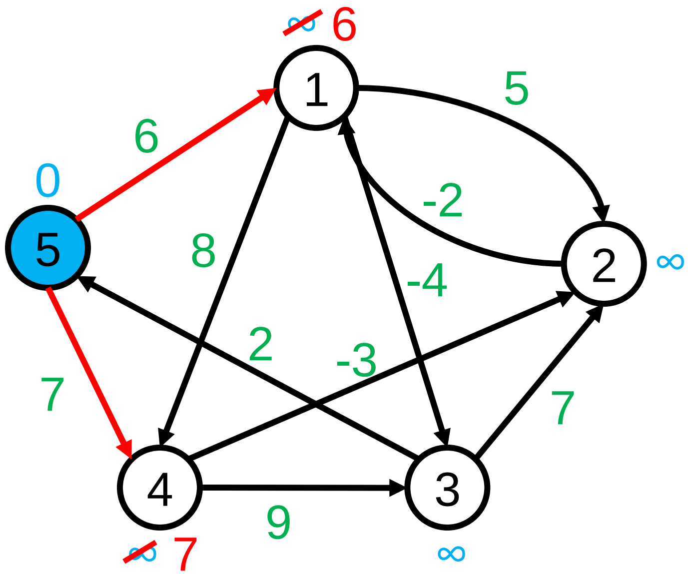
Exemplu Bellman-Ford (II)



d[1] = d[2] = d[3] = d[4] = d[5] = ∞
d[5] = 0

- BellmanFord(G,s)
  - **Pentru fiecare** v din V // init
    - d[v] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță pană 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”);

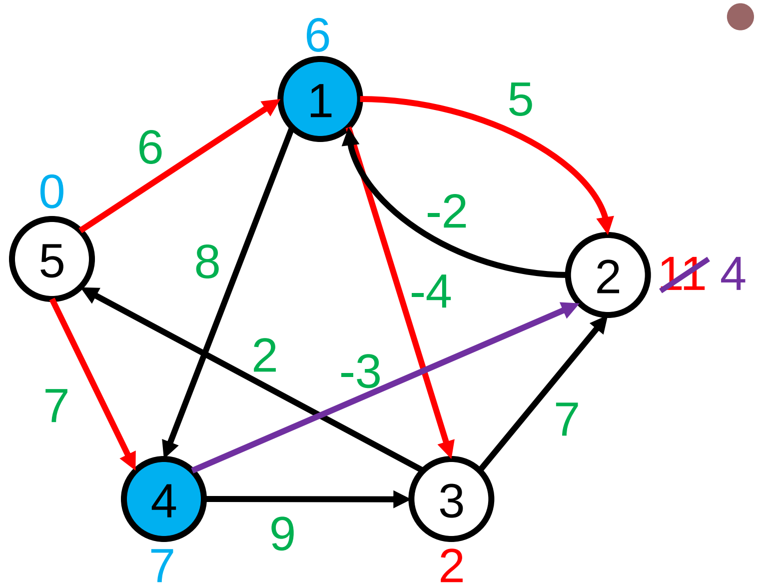
Exemplu Bellman-Ford (III)



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] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță pană 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");

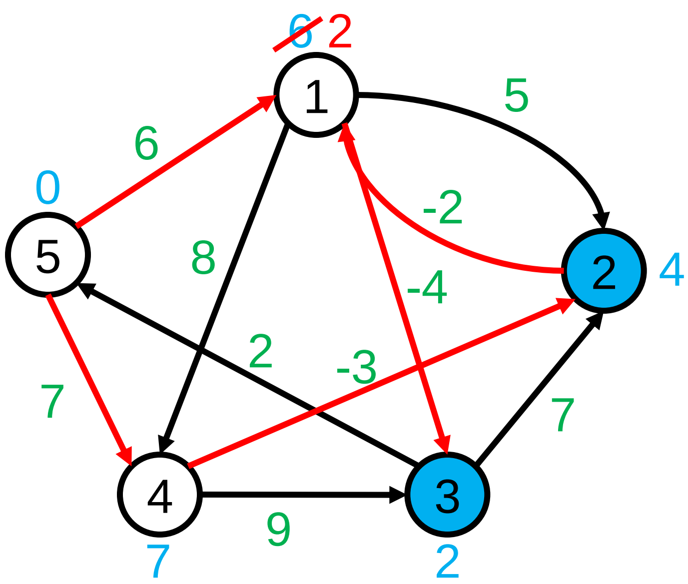
Exemplu Bellman-Ford (IV)



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] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță pană 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”);

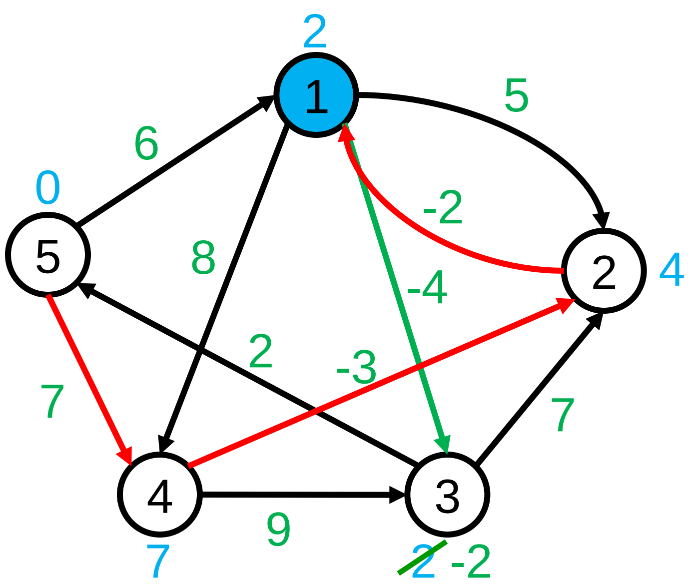
Exemplu Bellman-Ford (V)



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

- BellmanFord(G,s)
  - **Pentru fiecare** v din V // init
    - d[v] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță pană 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

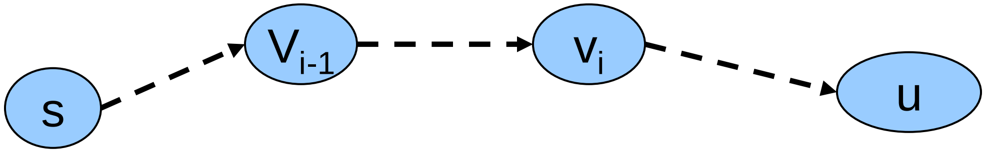
- BellmanFord(G,s)
  - **Pentru fiecare** v din V // init
    - d[v] = ∞;
    - p[v] = null;
  - d[s] = 0; // actualizare distanță pană la s
  - **Pentru i de la** 1 **la** |V| -1 // pt // fiecare pas de la s spre V-s
    - **Pentru fiecare** (u,v) din E // pt. // arcele ce pleacă de la nodurile // deja 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 → $\Re$ 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 $s = v_0, v_1 \ldots v_k = u$ un drum minim în graf cu $k \leq |V| - 1$.

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



Maximum |V|-1 arce

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) → $d[s] = d[v_0] = 0 = \delta(s,s) = \delta(s,v_0)$ 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), => conf. Lema 25.7 => $d[v_i] = d[v_{i-1}] + (v_{i-1},v_i) = \delta(s,v_{i-1}) + (v_i,v_{i-1}) = \delta(s,v_i)$. Conf. Lema 25.5, relația se menține până la sfârșit.
  - Cum $i \in (1,|V|-1)$ → relația e adevărată pentru toate nodurile accesibile din s → $d[v] = \delta(s,v)$, $\forall v \in R(s)$.

Corectitudine Bellman-Ford (III)

- Teorema. $G = (V,E)$, $w : E \rightarrow \Re$ 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 arcelor, 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.
  - => păstrăm o coadă cu vârfurile modificate (o singură copie).

Bellman-Ford optimizat

- BellmanFordOpt(G,s)
  - **Pentru fiecare nod v** (v ∊ V)
    - d[v] = ∞;
    - p[v] = null;
    - marcat[v] = false; // marcăm nodurile pentru care am făcut relaxare
    - Q = ∅; // coadă cu priorități
  - d[s] = 0; marcat[s] = true; Introdu(Q,s);
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); marcat[u] = false; // extrag minimul
    - **Pentru fiecare nod v** (v ∊ 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** (marcat[v] == false) {marcat[v] = true; Introdu(Q,v);}
- Observație: nu mai detectează cicluri negative!

Complexitate Bellman-Ford

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

O(VE)

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.

Notații

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

Teorema Floyd - Warshall

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

  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 $d^k(i,j) \le d^{k-1}(i,j)$, $\forall$ k = 1,n → $d^n(i,j) \le d^{k-1}(i,j)$, $\forall$ k = 1,n
  - Din $d^k(i,j) = \delta^k(i,j)$ → $d^n(i,j) = \delta^n(i,j) \le d^{k-1}(i,j) = \delta^{k-1}(i,j)$, $\forall$ k = 1,n → $d^n(i,j) = \delta^n(i,j) = \delta(i,j)$

Demonstrație teorema Floyd - Warshall

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

Algoritm Floyd-Warshall

Floyd-Warshall(G)

- **Pentru** i **de la** 1 **la** n
  - **Pentru** j **de la** 1 **la** n // inițializări
    - $d^0(i,j) = w(i,j)$
    - **Dacă** $(w(i,j) == \infty)$
      - $p^0(i,j) = null;$
    - **Altfel** $p^0(i,j) = i;$
- **Pentru** k **de la** 1 **la** n
  - **Pentru** i **de la** 1 **la** n
    - **Pentru** j **de la** 1 **la** n
      - **Dacă** $(d^{k-1}(i,j) > d^{k-1}(i,k) + d^{k-1}(k,j))$ // 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) == ∞)
      - p(i,j) = 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

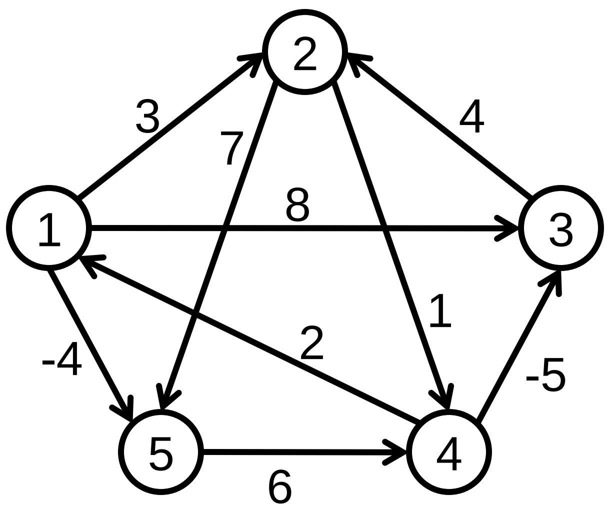
Complexitate?

$O(V^3)$

Complexitate spațială?

$O(V^2)$

Exemplu (I)



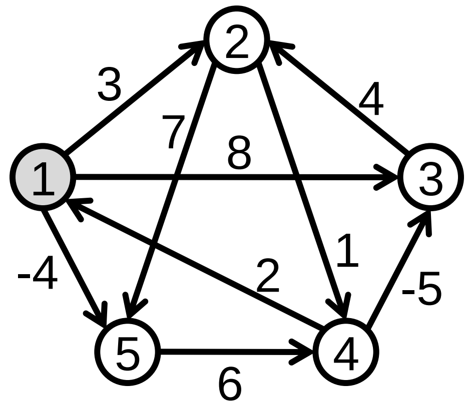
$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

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

Exemplu (II)

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

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

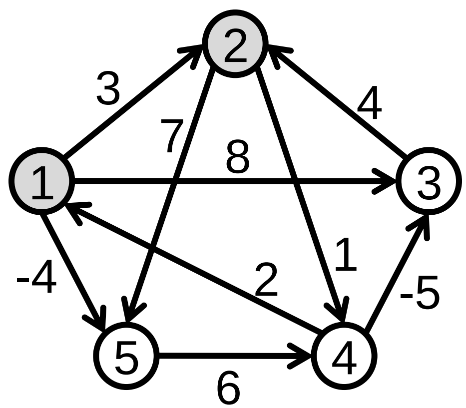


$$p = \begin{bmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & nil & 4 & nil & nil \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

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

Exemplu (III)

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

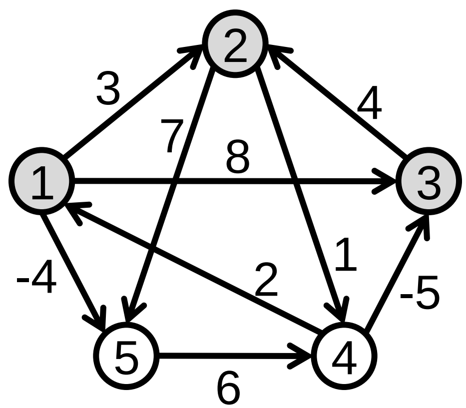


$$p = \begin{bmatrix} nil & 1 & 1 & nil & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & nil & nil \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix} \qquad p = \begin{bmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 1 & 4 & nil & 1 \\ nil & nil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

Exemplu (IV)

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

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



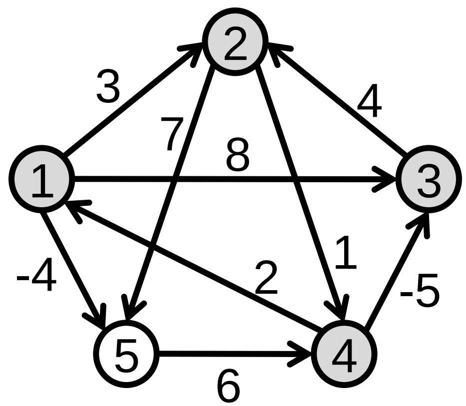
$$p = \begin{bmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 1 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

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

Exemplu (V)

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

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

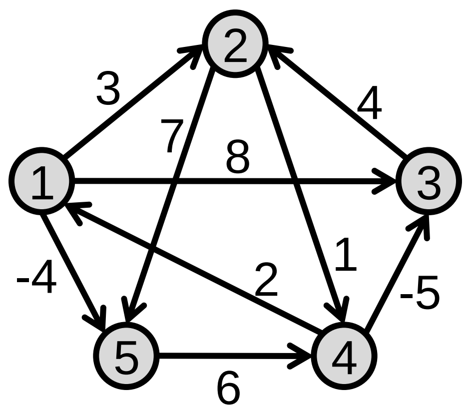


$$p = \begin{bmatrix} nil & 1 & 1 & 2 & 1 \\ nil & nil & nil & 2 & 2 \\ nil & 3 & nil & 2 & 2 \\ 4 & 3 & 4 & nil & 1 \\ nil & nil & nil & 5 & nil \end{bmatrix}$$

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

Exemplu (VI)

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



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

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 $\Longrightarrow$ operatorul boolean sau ($\vee$)
  - + $\Longrightarrow$ 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) ∊ E) ∨ (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) ∨ (E* (i,k) ∧ E* (k,j))

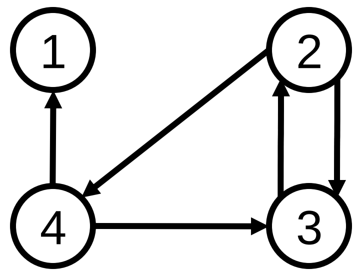
Complexitate? $O(V^3)$

Complexitate spațială? $O(V^2)$

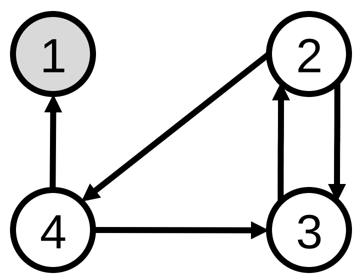
Exemplu (I)

Închidere_tranzitivă(G)

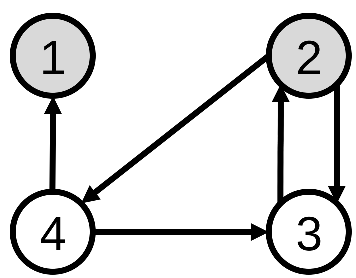
- **Pentru** i **de la** 1 **la** n
  - **Pentru** j **de la** 1 **la** n
    - E* (i,j) = (i,j) ∊ E ∨ i = j // inițializări
- **Pentru** k **de la** 1 **la** n
  - **Pentru** i **de la** 1 **la** n
    - **Pentru** j **de la** 1 **la** n
      - E* (i,j) = E* (i,j) ∨ (E* (i,k) ∧ E* (k,j))



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



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

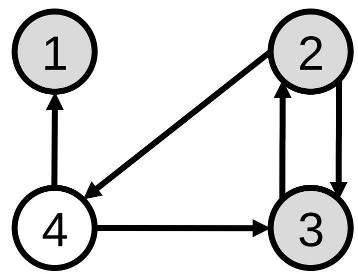


$$T^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

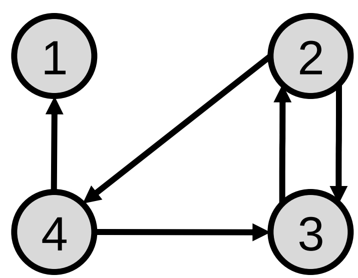
Exemplu (II)

Închidere_tranzitivă(G)

- **Pentru** i **de la** 1 **la** n
  - **Pentru** j **de la** 1 **la** n
    - E* (i,j) = (i,j) ∊ E ∨ 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) ∨ (E* (i,k) ∧ E* (k,j))



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



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

ÎNTREBĂRI?

Bibliografie curs 9

[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