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-> $\Re$ funcție de cost asociată arcelor grafului;
- cost(u..v) = costul drumului u..v (aditiv);
- d(v) = costul drumului descoperit s..v;
- δ(u,v) = costul drumului optim u..v; δ(u,v)=∞ dacă $v \notin R(u)$
  - δ(u,v) = Σw(x,y), $(x,y) \in$ u..v (u..v fiind drumul optim);
- p(v) = predecesorul lui v pe drumul s..v.

Reminder (II)

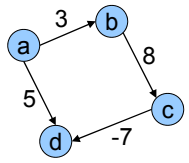
- Dijkstra(G,s)
  - ***Pentru fiecare*** (u ∈ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - ***Cat timp*** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] minim
    - // Q = Q - {u} – se execută in cadrul lui ExtrageMin
    - ***Pentru fiecare*** (v ∈ Q si v din succesorii lui u)
      - ***Daca*** (d[v] > d[u] + w(u,v))
        - d[v] = d[u] + w(u,v) // actualizez distanța
        - p[v] = u // si părintele

Corectitudine Dijkstra

- Teorema. G = (V,E), w:E->ℜ funcție de cost asociată nenegativă. La terminarea aplicării algoritmului Dijkstra pe acest graf plecând din sursa s vom avea d[v] = δ(s,v) pentru ∀v∈V.
- Dem: prin reducere la absurd se demonstrează că la scoaterea din Q a fiecărui nod u avem d[u]= δ(s,u).

Problema 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!** In momentul extragerii din coadă distanța pană la nodul d se consideră a fi calculată si 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 muchii de cost negativ!

Exemplu practic – muchii 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 ⇒ [208.10 × 1.5714] ⇒ $327.00.
- 1 oz. gold ⇒ 455.2 Francs ⇒ 304.39 Euros ⇒ $327.28. [455.2 × .6677 × 1.0752]

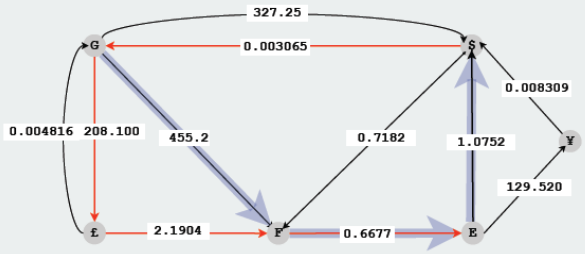
Currency	£	Euro	¥	Franc	$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3929	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250
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 – muchii de cost negativ (II)

Graph formulation.

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

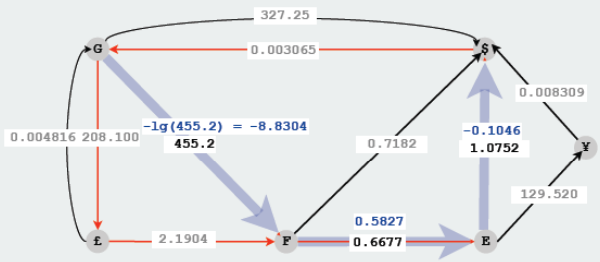


*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

Exemplu practic – muchii de cost negativ (III)

Reduce to shortest path problem by taking logs

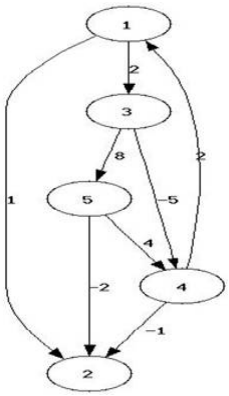
- Let weight(v-w) = – lg (exchange rate from currency v to w)
- multiplication turns to addition
- Shortest path with costs c corresponds to best exchange sequence.



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

Cicluri de cost negativ

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



- 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) = -∞

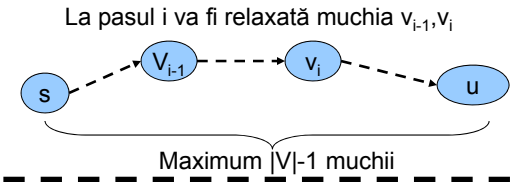
1-3-4 ciclu de cost negativ(-1)→ toate costurile din graf sunt -∞

Algoritmul Bellman-Ford

- BellmanFord(G,s) // G=(V,E),s=sursa
  - ***Pentru fiecare*** v in V[G] // 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 de la s spre V-s
    - ***Pentru fiecare*** (u,v) in E[G] // 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) in E[G]
    - ***Dacă*** d[v] > d[u] + w(u,v) ***atunci***
      - ***Eroare*** (”ciclu negativ”);

Corectitudine(I)

- Lemă: 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 fiecărei muchii avem d[v] = δ(s,v) pentru $\forall v \in R(s)$.
- Dem prin inducție:
  - Fie $s = v_0, v_1 \ldots v_k = u$ o cale in graf cu $k \leq |V| - 1$.



Corectitudine (II)

Demonstrăm că in pasul i: $d[v_i] = \delta(s,v_i)$.

P_0: (inițializare)→ $d[s] = d[v_0] = 0 = \delta(s,s) = \delta(s,v_0)$.

$P_{i-1} \rightarrow P_i$:

P_{i-1}: $d[v_{i-1}] = \delta(s,v_{i-1})$,

In pasul i se relaxează muchia (v_{i-1},v_i) ⇨

$d[v_i] = d[v_{i-1}] + (v_{i-1},v_i) = \delta(s,v_{i-1}) + (v_i,v_{i-1}) = \delta(s,v_i)$.

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 (III)

- Teorema. G = (V,E), w:E->$\Re$ funcție de cost asociată. Algoritmul BellmanFord aplicat acestui graf plecând din sursa s nu returnează EROARE dacă G nu conține cicluri negative, iar la terminare d[v] = δ(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 anterioare.
  - Dacă $\nexists$ ciclu negativ:
    - d[v] = δ(s,v) $\forall v \in R(s)$
    - d[v] = δ(s,v) = ∞, $\forall v \notin R(s)$ (inițializare)
    - → d[v] ≤ d[u] + w(u,v) → nu se întoarce eroare
  - Dacă $\exists$ ciclu negativ → in cei |V|-1 pasi se scad costurile muchiilor, iar in final ciclul se menține → Eroare

Optimizări Bellman-Ford

- Observație!
  - Dacă d[v] nu se modifică la pasul i atunci nu trebuie sa relaxăm niciuna din muchiile 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*** v in V[G]
    - 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);
  - ***Cat timp*** (Q != ∅)
    - u = ExtrageMin(Q); marcat[u] = false; // extrag minimul
    - ***Pentru fiecare*** (u,v) in E[G]
      - ***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) in E[G] —— * 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 intre 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 intre u si v folosind cel mai bun cost u..k si cel mai bun cost k..v calculat până in momentul respectiv.
- Se aplică pentru grafuri ce nu conțin cicluri de cost negativ.

Notații

- G = (V,E); V = {1,2,..n};
- w:VxV->$\Re$; w(i,i) = 0; w(i,j) = ∞ 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};
- δ(i,j) = costul drumului optim i..j; δ(i,j) = ∞ dacă $\nexists$ i..j;
- $\delta^k(i,j)$ = costul drumului optimi i..j ce trece doar prin noduri din mulțimea {1,2,..,k}; δ^k (i,j) = ∞ dacă $\nexists$ i..j;
- $p^k(i,j)$ = predecesorul lui j pe drumul i..j ce trece doar prin noduri din multimea {1,2,..,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 → $d^n(i,j) \leq d^{k-1}(i,j)$, $\forall$ k = 1,n
  - Din $d^k(i,j) = \delta^k(i,j)$ → $d^n(i,j) = \delta^n(i,j) \leq d^{k-1}(i,j) = \delta^{k-1}(i,j)$, $\forall$ 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 $d^0(i,j) = w(i,j)$;
- $0 < k \leq n$: $d^{k-1}(i,j) = \delta^{k-1}(i,j)$ → $d^k(i,j) = \delta^k(i,j)$
  - 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 in i..k si k..j optime

    $(\delta^{k-1}(i,k) = d^{k-1}(i,k)$ si $\delta^{k-1}(k,j) = d^{k-1}(k,j))$ si

    $\delta^k(i,j) = \delta^{k-1}(i,k) + \delta^{k-1}(k,j)$.
  - i..j optim → $\delta^k(i,j) \leq \delta^{k-1}(i,j)$
  - $d^k(i,j) = \min\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\}$

    $d^k(i,j) = \min\{\delta^{k-1}(i,j), \delta^{k-1}(i,k) + \delta^{k-1}(k,j)\} = \delta^{k-1}(i,k) + \delta^{k-1}(k,j) = \delta^k(i,j)$

Algoritm Floyd-Warshall

Floyd-Warshall(G)

- ***Pentru*** (i = 1 ; i ≤ n ; i++)
  - ***Pentru*** (j = 1 ; j ≤ n ; j++) // inițializări
    - $d^0(i,j) = w(i,j)$
    - ***Dacă*** (w(i,j) == ∞)
      - $p^0(i,j) = null;$
    - ***Altfel*** $p^0(i,j) = i;$
- ***Pentru*** (k = 1 ; k ≤ n ; k++)
  - ***Pentru*** (i = 1 ; i ≤ n ; i++)
    - ***Pentru*** (j = 1 ; j ≤ n ; j++)
      - ***Dacă*** ($d^{k-1}(i,j) > d^{k-1}(i,k) + d^{k-1}(k,j)$) // determinăm minimul
        - $d^k(i,j) = d^{k-1}(i,k) + d^{k-1}(k,j)$
        - $p^k(i,j) = p^{k-1}(k,j);$ // si 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

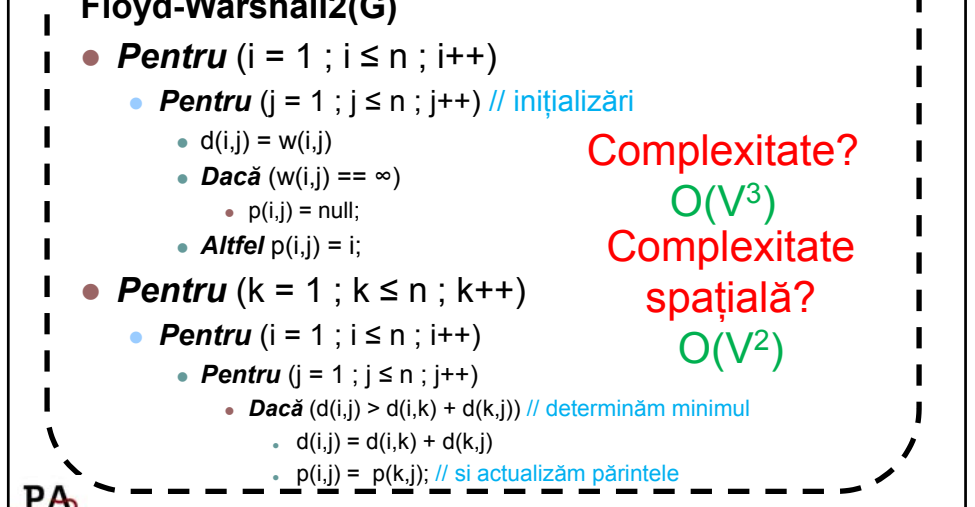
- Putem folosi o singură matrice in loc de n?
- Problemă: in pasul k, pt k < i si k < j, d(i,k) si d(k,j) folosite la calculul d(i,j) sunt $d^k(k,j)$ si $d^k(i,k)$ in loc de $d^{k-1}(k,j)$ si $d^{k-1}(i,k)$. Dacă dem. că $d^k(k,j)= d^{k-1}(k,j)$ si $d^k(i,k)=d^{k-1}(i,k)$, atunci putem folosi o singură 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

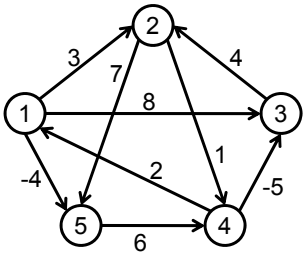
Floyd-Warshall2(G)

- ***Pentru*** (i = 1 ; i ≤ n ; i++)
  - ***Pentru*** (j = 1 ; j ≤ n ; j++) // inițializări
    - d(i,j) = w(i,j)
    - ***Dacă*** (w(i,j) == ∞)
      - p(i,j) = null;
    - ***Altfel*** p(i,j) = i;
- ***Pentru*** (k = 1 ; k ≤ n ; k++)
  - ***Pentru*** (i = 1 ; i ≤ n ; i++)
    - ***Pentru*** (j = 1 ; j ≤ n ; j++)
      - ***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); // si actualizăm părintele

Complexitate?
$O(V^3)$
Complexitate spațială?
$O(V^2)$



Exemplu (1)

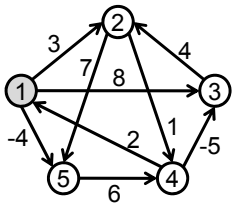


$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

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

Exemplu (2)

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



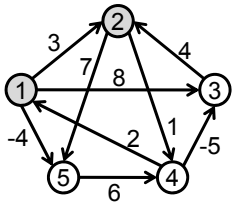
$$D = \begin{bmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

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

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

Exemplu (3)

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



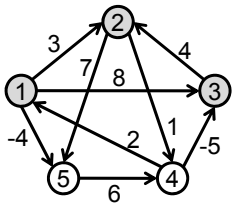
$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

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

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

Exemplu (4)

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



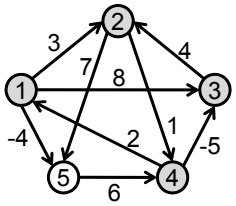
$$D = \begin{bmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{bmatrix}$$

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

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

Exemplu (5)

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



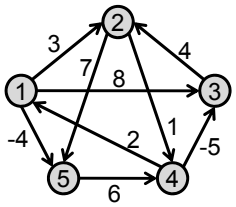
$$D = \begin{bmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{bmatrix}$$

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

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

Exemplu (6)

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



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

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

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

Închiderea tranzitivă

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

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

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

Închiderea tranzitivă (2)

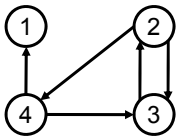
Închidere_tranzitivă(G)

- ***Pentru*** (i = 1 ; i ≤ n ; i++)
  - ***Pentru*** (j = 1 ; j ≤ n ; j++)
    - E* (i,j) = (i,j) ∈ E ∨ i=j // inițializări
- ***Pentru*** (k = 1 ; k ≤ n ; k++)
  - ***Pentru*** (i = 1 ; i ≤ n ; i++)
    - ***Pentru*** (j = 1 ; j ≤ n ; j++)
      - E* (i,j) = E* (i,j) ∨ (E* (i,k) ∧ E* (k,j))

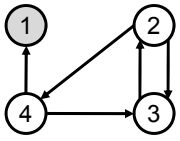
Complexitate? $O(V^3)$

Complexitate spațială? $O(V^2)$

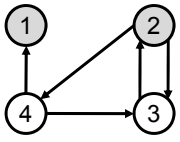
Exemplu (1)



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

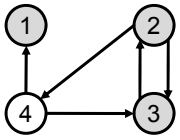


$$T^{(1)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

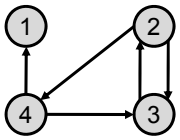


$$T^{(2)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{bmatrix}$$

Exemplu (2)



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



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

Algoritmul lui Johnson

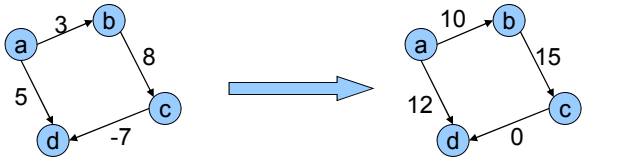
- Pentru grafuri rare.
- Folosește liste de adiacență.
- Bazat pe Dijkstra si Bellman-Ford.
- Complexitate: $O(V^2 \log V + VE)$
  - mai buna decât Floyd-Warshall pentru grafuri rare.

Idee algoritm Johnson

- Dacă graful are numai muchii pozitive:
  - se aplică Dijkstra pentru fiecare nod ⟹ cost $V^2 \log V + VE$.
- Altfel se calculează costuri pozitive pentru fiecare muchie menținând proprietățile:
  - $w_1(u,v) >= 0 \ \forall (u,v) \in E$;
  - p este drum minim utilizând w ⟺ p este drum minim utilizând w_1.

Construcție w_1

- Idee 1: identificare muchia cu cel mai mic cost – c; adunare la costul fiecărei muchii valoarea c;



cost(a..b..d) < cost(a,d)

cost(a..b..d) > cost(a,d)

Nu funcționează!!!!

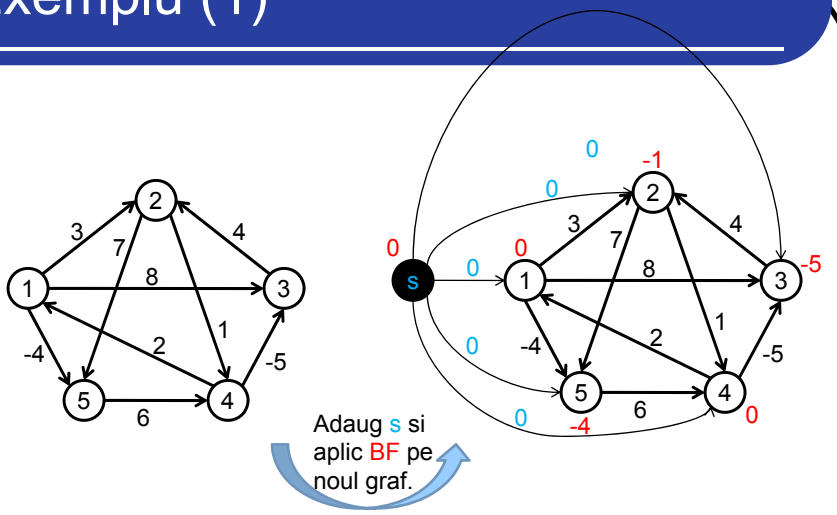
Construcție w_1

- Idee 2: $w_1(u..v) = w(u..v) + h(u) - h(v)$;
- unde $h:V->\Re$;
- se adaugă un nod s;
- se unește s cu toate nodurile grafului prin muchii de cost 0;
- se aplica BF pe acest graf => $h(v) = \delta(s,v)$;
- → $w_1(u,v) = w(u,v) + h(u) - h(v)$.

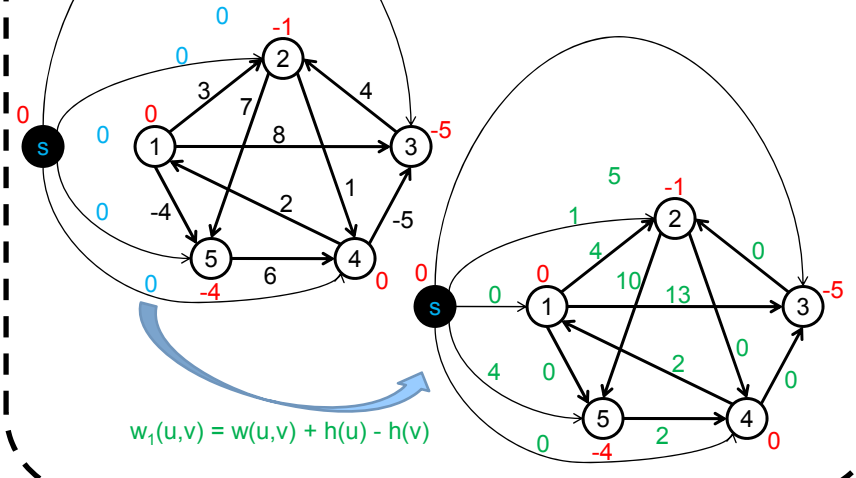
Algoritm Johnson

- Johnson(G)
  - G' = (V',E');
  - V' = V ∪ {s}; // adăugăm nodul s
  - E' = E ∪ (s,u), ∀u∈V; w(s,u) = 0; // si îl legăm de toate nodurile
  - ***Dacă*** BF(G') e fals // aplic BF pe G'
    - ***Eroare*** "ciclu negativ"
  - ***Altfel***
    - ***Pentru fiecare*** v∈V
      - h(v) = δ(s,v); // calculat prin BF
    - ***Pentru fiecare*** (u,v)∈E
      - $w_1(u,v) = w(u,v) + h(u) - h(v)$ // calculez noile costuri pozitive
    - ***Pentru fiecare*** (u∈V)
      - Dijkstra(G,w_1,u) // aplic Dijkstra pentru fiecare nod
      - ***Pentru fiecare*** (v∈V)
        - $d(u,v) = \delta_1(u,v) + h(v) - h(u)$ // calculez costurile pe graful inițial

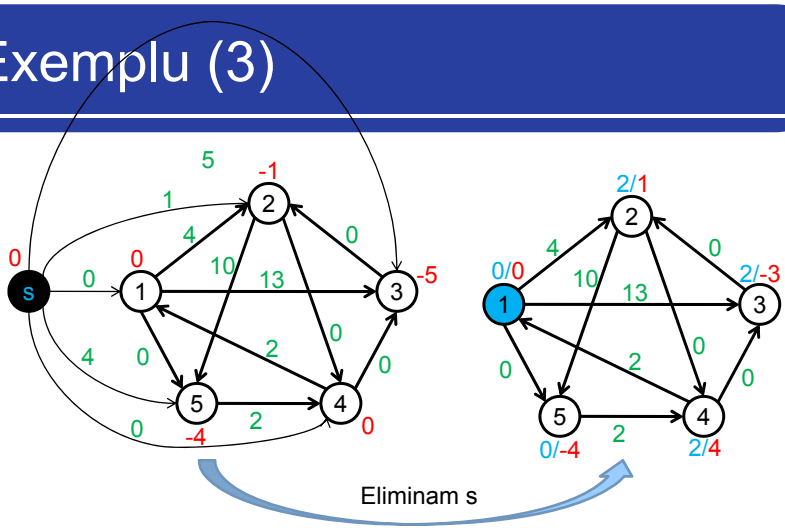
Exemplu (1)



Exemplu (2)

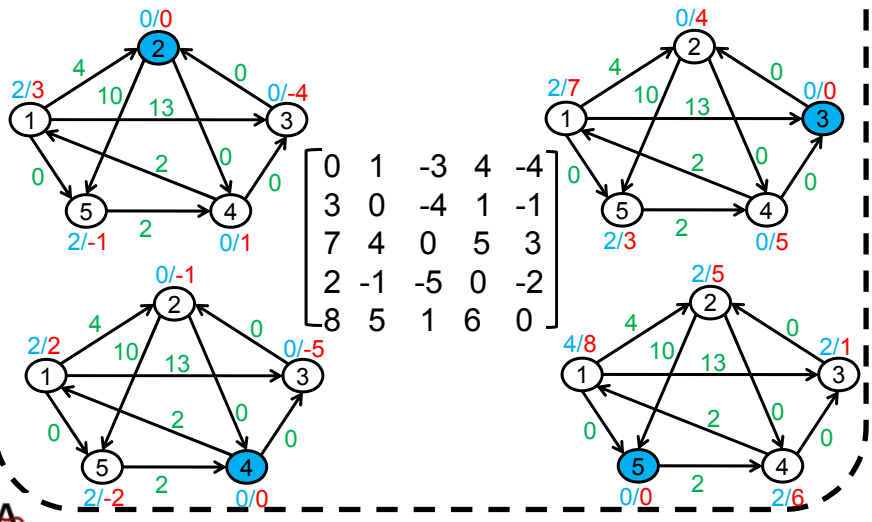


Exemplu (3)



Aplicam Dijkstra din fiecare nod ($\delta_1(u,v)$).
Refacem distanțele:
$d(u,v) = \delta_1(u,v) + h(v) - h(u)$

Exemplu (4)



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

Concluzii Floyd-Warshall & Johnson

- Algoritmi ce găsesc drumurile minime intre oricare 2 noduri din graf.
- Funcționează pe grafuri cu muchii 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 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