UNIUNEA EUROPEANĂ

GUVERNUL ROMÂNIEI

Instrumente Structurale
2007-2013

Platformă de e-learning și curriculă e-content pentru învățământul superior tehnic

Proiectarea Algoritmilor

16. Algoritmul lui Dijkstra

Bibliografie

[1] Giumale – Introducere in Analiza Algoritmilor cap. 5.3, 5.4, 5.4.1

[2] Cormen – Introducere în Algoritmi cap. 20, 21, 25.1 si 25.2

[3] R. Sedgewick, K. Wayne - Algorithms and Data Structures Fall 2007 – Curs Princeton - http://www.cs.princeton.edu/~rs/AlgsDS07/

[4] Heap Fibonacci: http://www.cse.yorku.ca/~aaw/Jason/FibonacciHeapAnimation.html

Algoritmul lui Dijkstra (I)

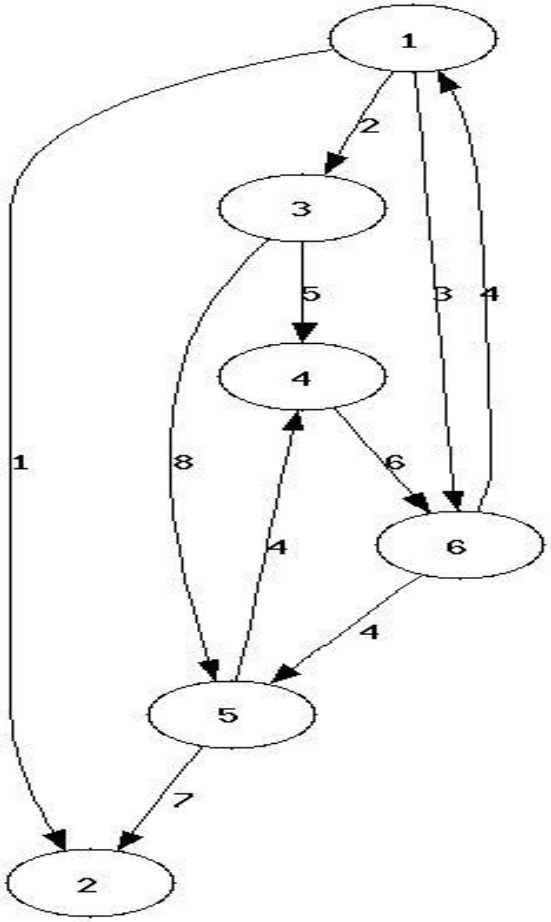
- Folosește o coada de priorități în care se adaugă nodurile în funcție de distanța cunoscută în momentul respectiv de la s până la nod.
- Se folosește **NUMAI** pentru costuri pozitive ($w(u,v) > 0, \forall u,v \in V$).
- Dijkstra_generic (G,s)
  - V = nodurile lui G
  - **Cât timp** (V != ∅)
    - 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** (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** (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

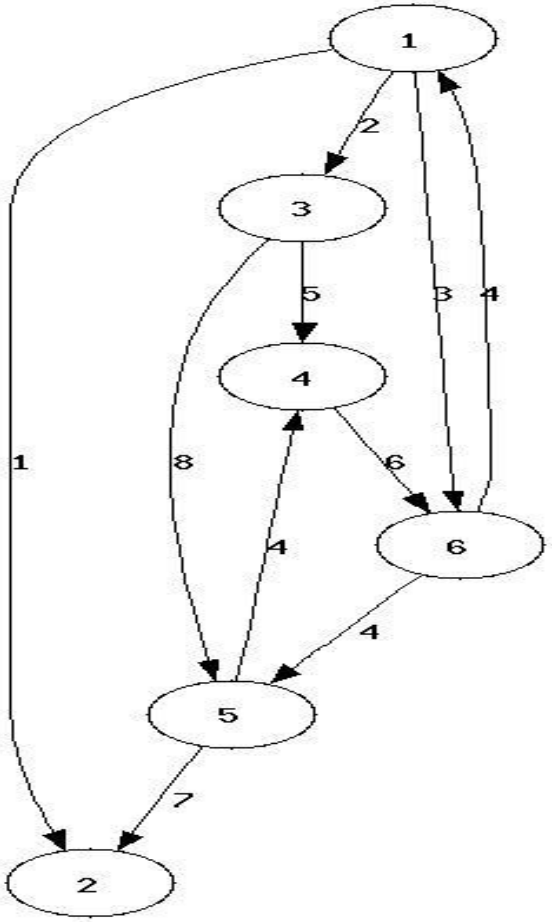
Exemplu (I)



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** (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** (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** (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 in cadrul lui ExtrageMin
    - **Pentru fiecare** (v ∈ 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

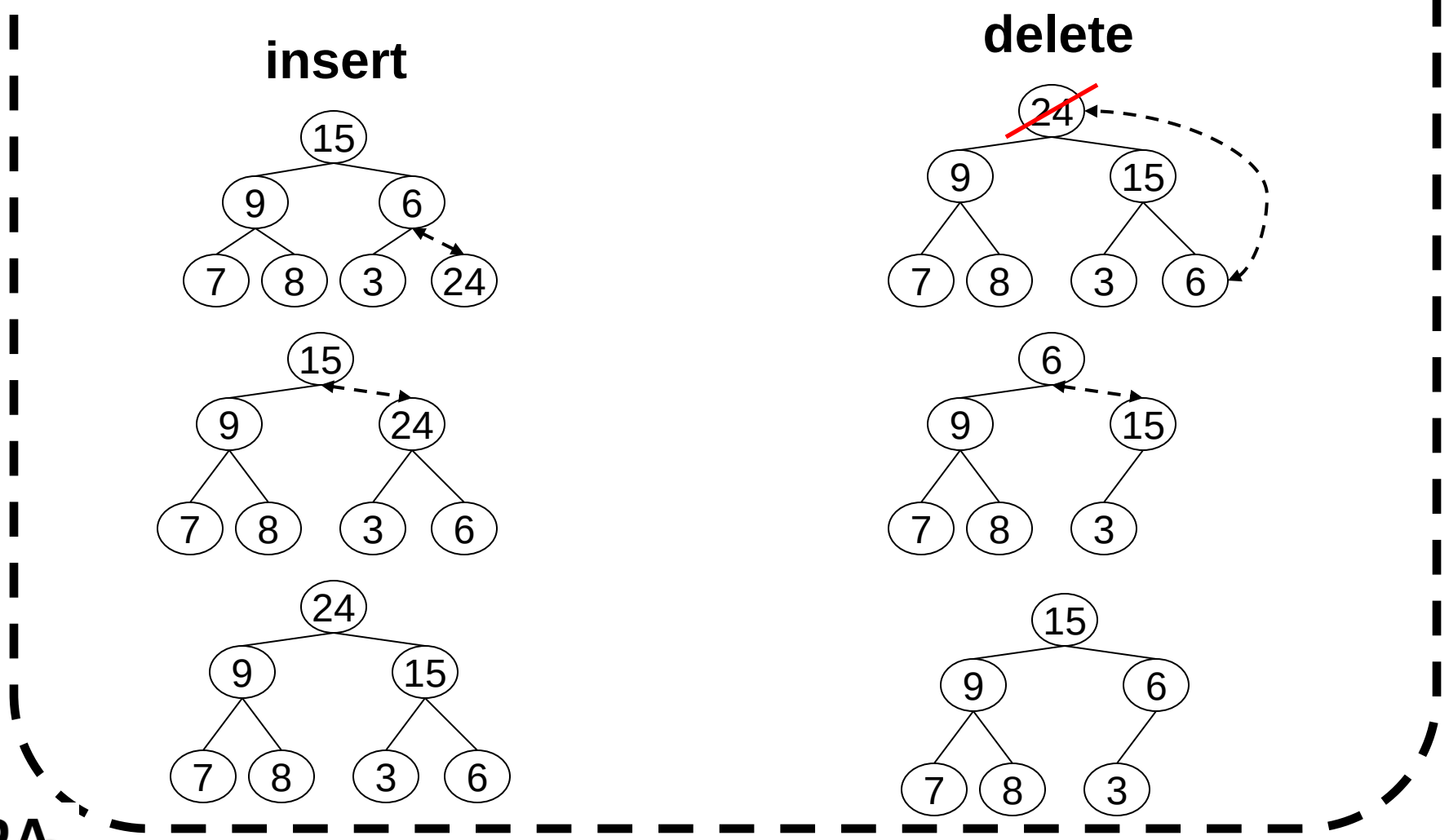
Implementare cu vectori

- 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)) && $u \geq$ răd(dr(u)) unde $\geq$ este o relație de ordine pe mulțimea pe care sunt definite elementele heapului.

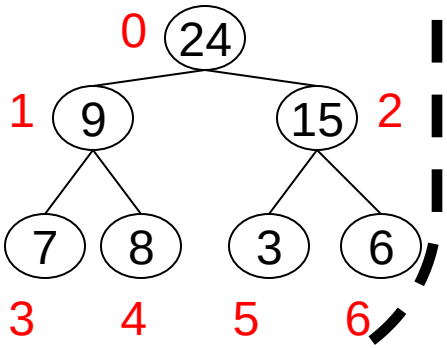
Operatii pe Heap Binar



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.

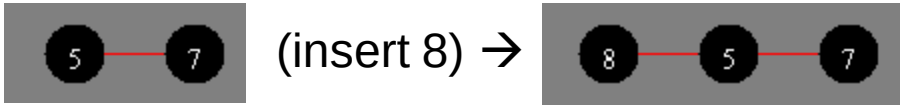
- Eficient dacă graful are muchii puține comparativ cu numărul de noduri.

Heap Fibonacci

- Poate fi format din mai mulți arbori.
- Cheia unui părinte ≤ cheia oricărui copil.
- Fiind dat un nod u si 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 intre 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

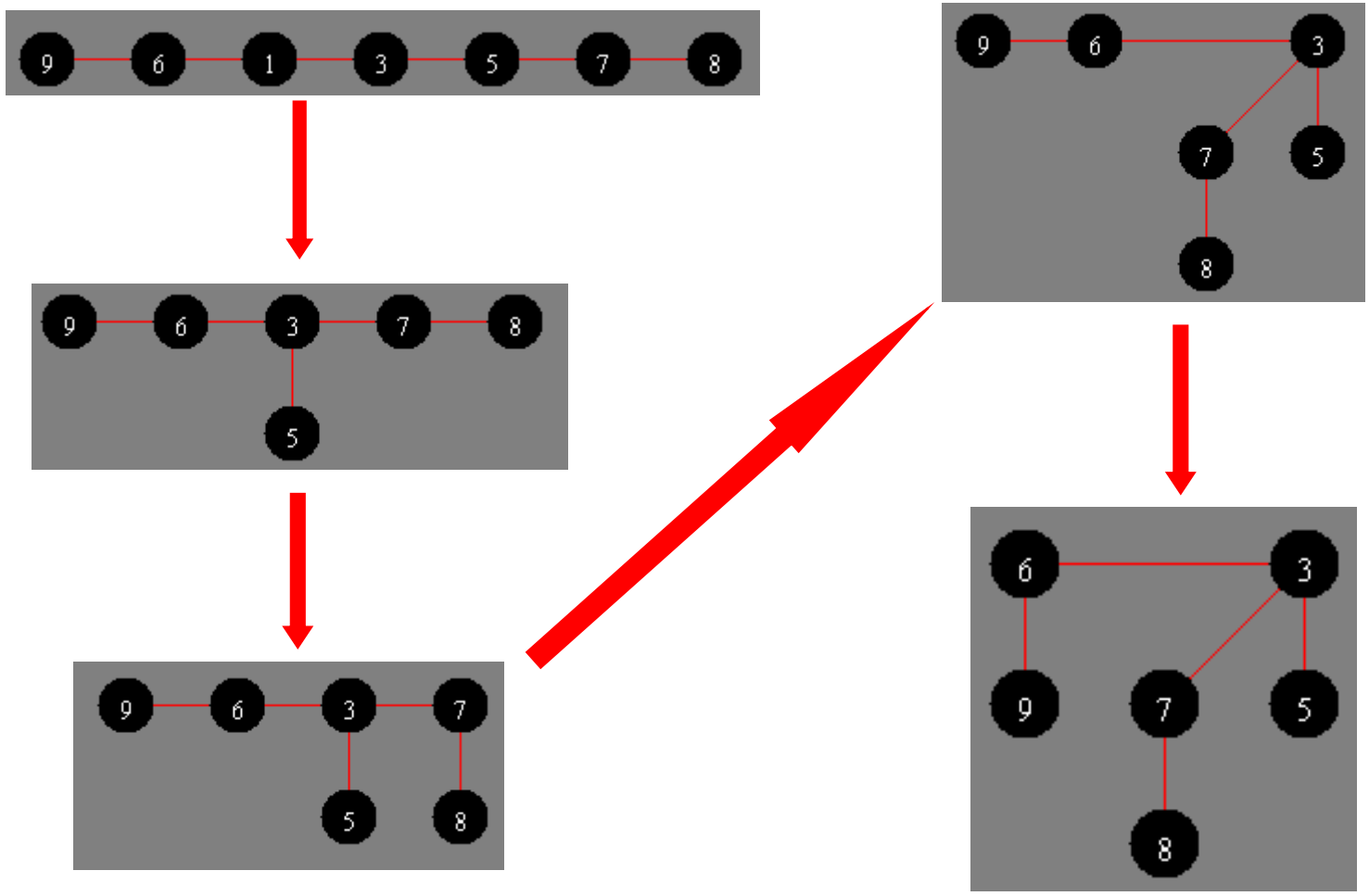


- Min – accesibil direct - min(H) – O(1)
- ExtrageMin O(logn) – cost amortizat!
  - Mută copiii minimului pe prima coloană;
  - **Consolidează** heap-ul.

Operatii Heap Fibonacci

- Consolidare Heap
  - Cât timp există 2 arbori cu grade diferite 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

- 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 – Optimalitatea drumurilor minime (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.

- Dem: Fie $p_{ij} = v_i..v_j$ subdrumul din p dintre v_i și v_j. → p = $v_1..v_i..v_j..v_k$ => cost (p) = cost ($v_1..v_i$) + cost ($v_i..v_j$) + cost ($v_j..v_k$).
- Pp. prin absurd că $v_i..v_j$ nu e optim => ∃p' a.i. cost (p') < cost ($v_i..v_j$) => p nu e drum minim → Contrazice ipoteza → p_{ij} este drum minim.

Corectitudine Dijkstra – Optimalitatea drumurilor minime (II)

- Corolar 25.2: G = (V,E), w : E → $\mathfrak{R}$ 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 δ(s,v) = δ(s,u) + w(u,v).
- Dem: Conform teoremei anterioare, s..u e un drum optim => cost (s..u) = δ(s,u).
- Lemă 25.3: G = (V,E), w : E → $\mathfrak{R}$ funcție de cost asociată. ∀ (u,v) ∈ E avem δ(s,v) ≤ δ(s,u) + w(u,v).
- Dem: Orice drum optim are costul mai mic ca al oricărui alt drum.

Corectitudine Dijkstra – Relaxarea muchiilor (I)

- Lemă 25.5: $G = (V,E)$, $w : E \to \Re$ 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.

- Dem:
  - $\forall v \in V$, $v \notin R(s) \to d[v] = \delta(s,v) = \infty$; $d[s] = \delta(s,s) = 0$ (inițializare)
  - Pt $v \in R(s)$, inițializare $\to d[v] = \infty \geq \delta(s,v)$. Dem. prin reducere la absurd că după oricâte relaxări, relația se menține. Fie v primul vârf pentru care relaxarea (u,v) determină $d[v] < \delta(s,v) \to$ după relaxarea (u,v): $d[u] + w(u,v) = d[v] < \delta(s,v) \leq \delta(s,u) + w(u,v) \to d[u] < \delta(s,u)$. Dar relaxarea nu modifică d[u], iar v e primul pentru care $d[v] < \delta(s,v)$. Contrazice presupunerea! => $d[v] \geq \delta(s,v)$, $\forall v \in V$
  - Cum $d[v] \geq \delta(s,v)$ => odată ajuns la $d[v] = \delta(s,v)$, ea nu mai scade. Cum relaxarea nu creste valorile => d[v] nu se mai modifică.

Corectitudine Dijkstra – Relaxarea muchiilor (II)

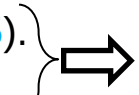
- Lemă 25.7: G = (V,E), w : E → $\mathfrak{R}$ funcție de cost asociată. Fie p = s..uv un drum optim de la s la v. Dacă d[u] = δ(s,u) la un moment dat, atunci începând cu momentul imediat următor relaxării muchiei (u,v) avem d[v] = δ(s,v).

- Dem:
  - Dacă înainte de relaxare d[v] > d[u] + w(u,v), prin relaxare → d[v] = d[u] + w(u,v). Altfel, d[v] ≤ d[u] + w(u,v) => după relaxare avem d[v] ≤ d[u] + w(u,v).
  - Cum d[u] = δ(s,u) și relaxarea (u,v) nu modifică d[u] => d[v] ≤ d[u] + w(u,v) = δ(s,u) + w(u,v) = δ(s,v) (conf. Corolar 25.2) → d[v] = δ(s,v)

Corectitudine Dijkstra

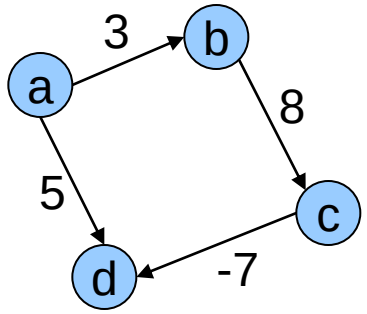
- Teoremă. G = (V,E), w : E → $\Re$ funcție de cost asociată nenegativă. La terminarea aplicării algoritmului Dijkstra pe acest graf plecând din sursa s vom avea d[v] = δ(s,v) pentru $\forall$ v $\in$ V.

- Dem: prin reducere la absurd se demonstrează că la scoaterea din Q a fiecărui nod v avem d[u] = δ(s,u) și egalitatea se menține și ulterior.
  - Pp. u e primul nod pt. care d[u] $\neq$ δ(s,u) la scoaterea din Q. u $\neq$ s pt. că altfel d[u] = δ(s,u) = 0 și u $\in$ R(s) pt.că altfel d[u] = δ(s,u) = ∞. => La scoaterea lui u din Q, $\exists$ drum s..u si fie p drumul optim s..u a.i. p = s..xy..u, unde x $\notin$ Q iar y $\in$ Q.
  - Cum u e primul nod pt. care d[u] $\neq$ δ(s,u) => d[x] = δ(s,x) la momentul extragerii lui u din Q → d[y] = δ(s,y) prin relaxarea (x,y) (conf. Lema 25.7).
  - y precede u pe drumul p => d[y] = δ(s,y) $\leq$ δ(s,u) $\leq$ d[u] (conf. Lema 25.5).
  - Cum y $\in$ Q la momentul scoaterii lui Q => d[u] $\leq$ d[y]
  - => d[y] = δ(s,y) = δ(s,u) = d[u] – Contrazice ipoteza! => d[u] = δ(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!** 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. [208.10 × 1.5714]
- 1 oz. gold ⇒ £208.10 ⇒ ⇒ $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]

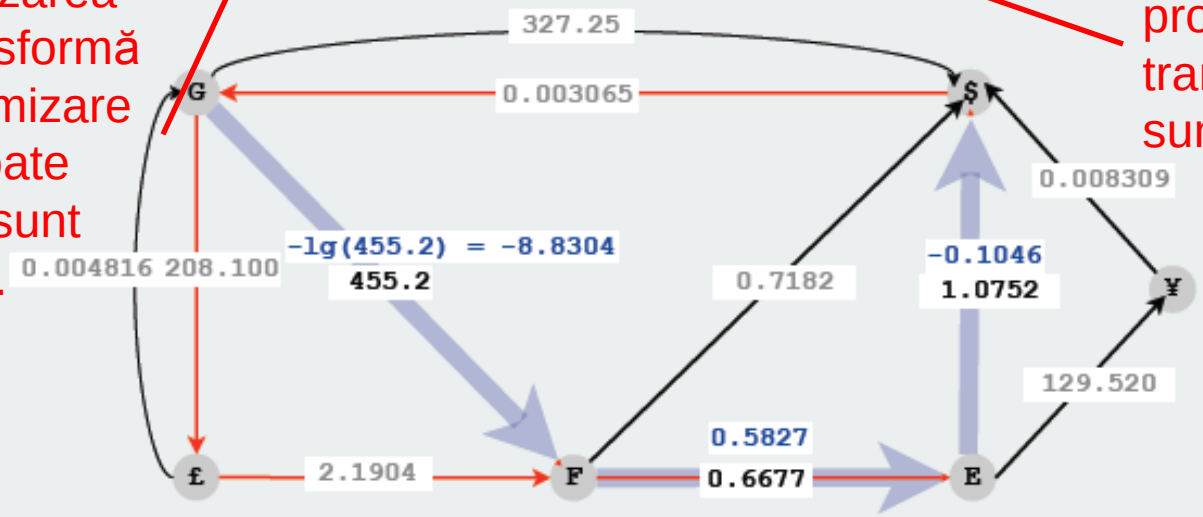
Exemplu practic – muchii de cost negativ (III)

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.

Maximizarea se transformă în minimizare dacă toate arcele sunt negate.

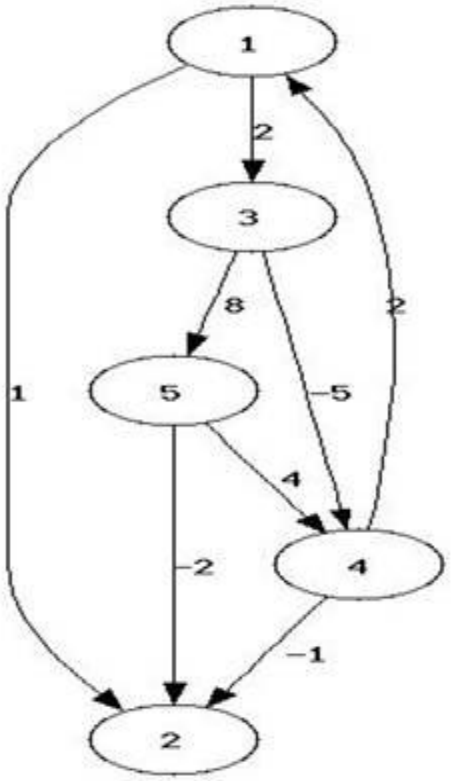
Prin logaritmare produsul se transformă în sumă.



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

Cicluri de cost negativ

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



- 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 -∞