Drumuri de distanțe minime

Drumuri minime

- G=(N,A);
- w:A-> $\Re$ funcție de cost asociată arcelor grafului
- cost(u..v)= costul drumului u..v
- d(v)=costul drumului descoperit
- δ(u,v)= costul drumului optim u..v; δ(u,v)=∞ dacă v∉ R(u)
- tată(v)=predecesorul lui v pe drumul s..v

Parcurgerea în lățime

- **Parc_lățime(G,s)**

Pentru fiecare u∈ N **repetă**

d[u] ← ∞

culoare[u]←alb

tată[u] ← null

d[s] ← 0;

Q ← new_queue(s)

Cât timp(not empty?(Q)) **repetă**

u ← extract(Q) //extrage din V primul element

Pentru fiecare (v din succesorii lui u) **repetă**

Dacă culoare[v]=alb

Atunci culoare[v]←gri;

insert(v,Q)

d[v]←d[u]+1

tată[v]←u

culoare[u]←negru

Drumuri minime de sursă unică

- G=(N,A);
- s∈ N – nodul sursă
- w:A-> $\mathfrak{R}$ funcție de cost asociată arcelor grafului
- cost(u..v)= costul drumului u..v
- d(v)=costul drumului descoperit
- δ(u,v)= costul drumului optim u..v; δ(u,v)=∞ dacă v∉ R(u)
- tată(v)=predecesorul lui v pe drumul s..v

Parcurgerea în lățime - d=nr. arce

- **Parc_lățime(G,s)**

Pentru fiecare u∈ N **repetă**

d[u] ← ∞

culoare[u]←alb

tată[u] ← null

d[s] ← 0;

Q ← new_queue(n)

Insert(s,Q)

Cât timp(not empty?(Q)) **repetă**

u ← extract(Q) //extrage din Q primul element

Pentru fiecare (v din succesorii lui u) **repetă**

Dacă culoare[v]=alb

Atunci culoare[v]←gri;

insert(v,Q)

d[v]←d[u]+1

tată[v]←u

culoare[u]←negru

Algoritmul lui Dijkstra - d=Σw(x,y),

- **Dijkstra(G,s)**

  Q ← new_queue(n)

  Pentru fiecare u∈ N **repetă**

    d[u] ← ∞

    Insert(u,Q)

    tată[u] ← null

  d[s] ← 0;

  cât timp(not empty?(Q)) **repetă**

    u ← min_extract(Q) //extrage din V elementul cu d[u] minim

    Pentru fiecare (v din succesorii lui u) **repetă**

      Dacă (d[v]>d[u]+w(u,v))

      a**tunci** d[v] ← d[u]+w(u,v)

        tată[v]←u

Algoritmul lui Prim (AMA)

- **Prim(G,s)**

  Q ← new_queue(n)

  Pentru fiecare u∈ N **repetă**

    d[u] ← ∞

    Insert(u,Q)

    tată[u] ← null

  d[s] ← 0;

  cât timp(not empty?(Q)) **repetă**

    u ← min_extract(Q) //extrage din V elementul cu d[u] minim

    Pentru fiecare (v din succesorii lui u) **repetă**

      Dacă (d[v]>d[u]+w(u,v))

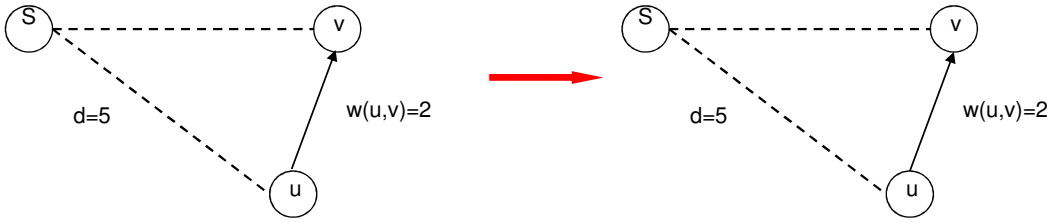
      a**tunci** d[v] ← w(u,v)

        tată[v]←u

Funcție relaxare(u,v)=

dacă d[v]>d[u]+w(u,v) **atunci** d[v]←d[u]+w(u,v}

d=10 d=7



Algoritmul lui Dijkstra

- **Dijkstra(G,s)**

  Q ← new_queue(n)

  Pentru fiecare u∈ N **repetă**

    d[u] ← ∞

    Insert(u,Q)

    tată[u] ← null

  d[s] ← 0;

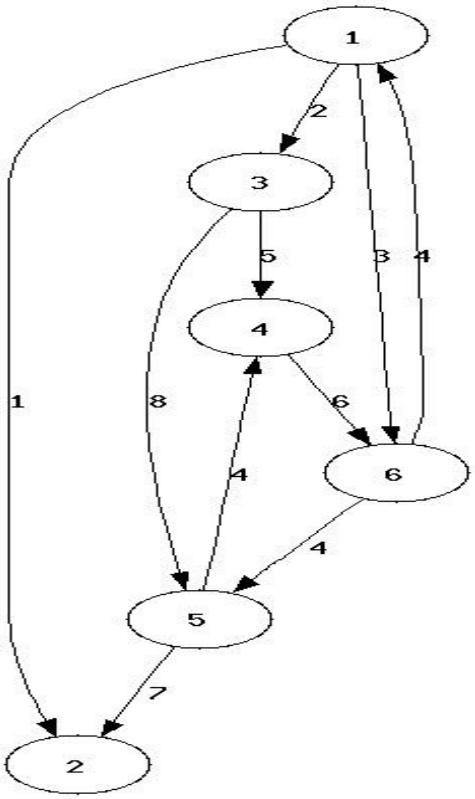
  cât timp(not empty?(Q)) **repetă**

    u ← min_extract(Q) //extrage din Q elementul cu d[u] minim

    Pentru fiecare (v din succesorii lui u) **repetă**

      Relaxare(u,v)

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



Complexitate

- depinde de min_extract – coadă cu priorități
- Operații ce trebuie realizate pe coadă * frecvența
  - new_queue
  - insert - n
  - delete - n
  - member? - n
  - decrease_val - a
  - empty? - n

Implementare cu vectori

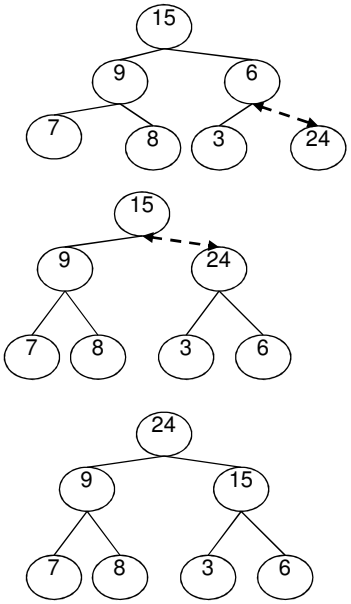
- Costuri
  - insert – 1*n=n
  - delete – n*n=n^2 (necesită cautarea minimului)
  - member? – 1*n=n
  - decrease_val – 1*a=a
  - empty? – 1*n=n
- Cea mai bună metodă pentru grafuri “dese” $a \approx n^2$

Implementare cu heap binar

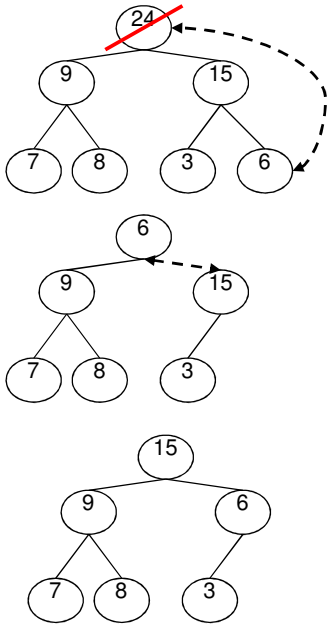
- Heap binar – structura de date de tip arbore binar + 2 restricții
  - fiecare nivel este complet; ultimul se umple de la stanga la dreapta (arbore binar esențial complet)
  - $\forall u \in$ Heap; $u \geq st(u)$ && $u \geq dr(u)$ unde $\geq$ este o relatie de ordine pe multimea pe care sunt definite elementele heapului

Heap Binar

insert



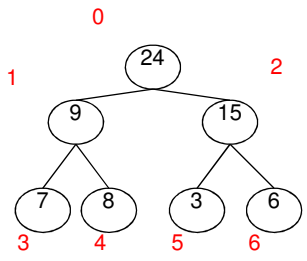
delete



Implementare Heap Binar

- implementare folosind vectori
- pozitie[i]=unde se gaseste in indexul de valori elementul de pe pozitia i din heap
- reverse[i]=unde se gaseste in heap elementul de pe pozitia i din valoare
- implementare disponibila la [2]

index	0	1	2	3	4	5	6
valoare	7	6	15	8	24	9	3
Pozitie	4	5	2	0	3	6	1
Reverse	3	6	2	4	0	1	5



Heap Binar

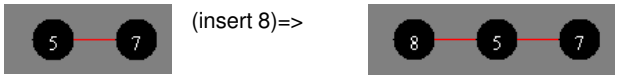
- Costuri
  - insert – logn*n=nlogn
  - delete – logn*n=nlogn
  - member? – 1*n=n
  - decrease_val – logn*a=alogn
  - empty? – 1*n=n
- eficient daca graful are muchii putine comparativ cu numarul de noduri

Heap Fibonacci

- poate fi format din mai multi arbori
- cheia unui parinte <= cheia oricarui copil
- fiind dat un nod u si un heap H
  - tată(u) – parintele lui u
  - copil(u) – legatura catre unul din copiii lui u
  - st(u), dr(u) – legatura la fratii din stanga si din dreapta (cei de pe primul nivel sunt legati intre ei astfel)
  - grad(u) – numarul de copii ai lui u
  - min(H) – cel mai mic nod din H
  - n(H) – numarul de noduri din H

Operatii Heap Fibonacci

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



- min – accesibil direct - min(H) – O(1)
- extrage-min O(logn) – cost amortizat!
  - muta copiii minimului pe prima coloana
  - **consolideaza** heap-ul

Operatii Heap Fibonacci

- Consolidare Heap
  - cat timp exista 2 arbori cu grade diferite Arb(x) si Arb(y);x<y
    - Arb(y) adaugat ca si copil al lui x
    - grad[x]++;

Complexitate Dijkstra

- Listă $O(n^2 + a)$
- Heap binar $O(a \log n)$
- Heap Fibonacci $O(n \log n + a)$

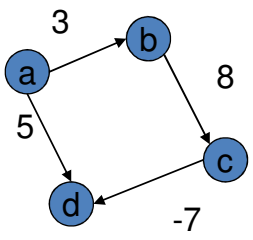
Probleme:

Drumuri de lg. Maximă în DAG-uri

Drum critic

Problemă la Dijkstra (I)

- Exemplu rulare
  - d[a]=0;d[b]=d[c]=d[d]=∞
  - d[b]=3; d[d]=5;
  - d[c]=11;
  - **d este extras din coada!** In momentul extragerii din coada distanta pana la nodul d se considera a fi calculata si a fi optima
  - Se extrage nodul c; d[d] nu va mai fi actualizata – nodul d fiind deja eliminat din coada.



Problemă la Dijkstra (II)

- => algoritmul nu functioneaza pentru grafuri ce contin 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 ⇒ ⇒ $327.00. [208.10 × 1.5714]
- 1 oz. gold ⇒ 455.2 Francs ⇒ 304.39 Euros ⇒ $327.28. [455.2 × .6677 × 1.0752]

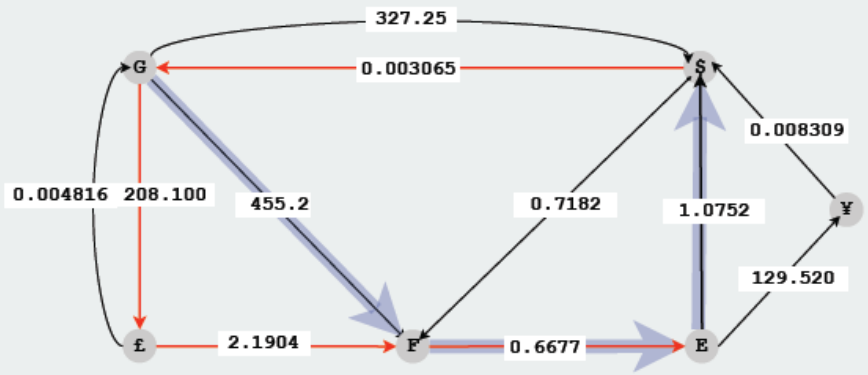
Currency	£	Euro	¥	Franc	$	Gold
UK Pound	1.0000	0.6853	0.005290	0.4569	0.6368	208.100
Euro	1.4599	1.0000	0.007721	0.6677	0.9303	304.028
Japanese Yen	189.050	129.520	1.0000	85.4694	120.400	39346.7
Swiss Franc	2.1904	1.4978	0.011574	1.0000	1.3929	455.200
US Dollar	1.5714	1.0752	0.008309	0.7182	1.0000	327.250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

*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne
http://www.cs.princeton.edu/~rs/AlgsDS07/

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.

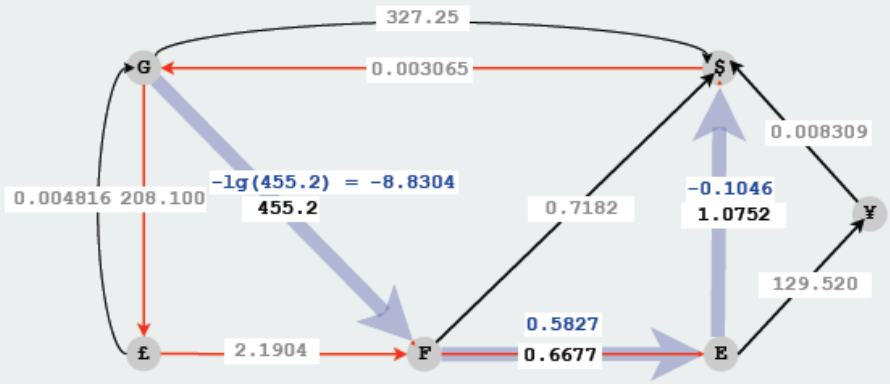


*slide din cursul de algoritmi de la Princeton – Sedgewick&Wayne[1]

Exemplu practic – muchii de cost negativ (III)

Reduce to shortest path problem by taking logs

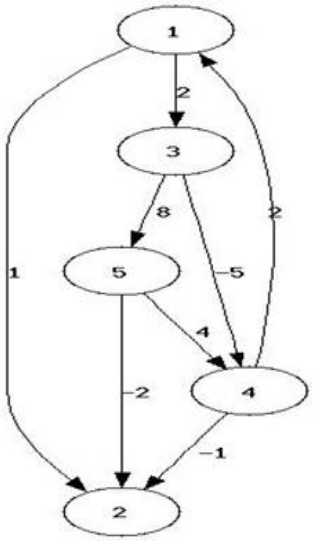
- Let weight(v-w) = – lg (exchange rate from currency v to w)
- multiplication turns to addition
- Shortest path with costs c corresponds to best exchange sequence.



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

Cicluri de cost negativ

- $\delta(u,v) = \begin{cases} \sum w(x,y),\ (x,y) \in u..v \\ \quad (u..v \text{ fiind drumul optim } s..u) \\ \infty \text{ daca nu exista drum } u..v \end{cases}$
- Daca exista pe drumul u..v un ciclu de cost negativ x..y=>
  - $\delta(u,v) = \delta(u,v) + cost(x..y) < \delta(u,v)$
  - => costul este $-\infty$
  - => $\delta(u,v) = -\infty$



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

Algoritmul Bellman-Ford

Funcție BellmanFord(G,s)

Pentru fiecare u∈ N **repetă**

d[u] ← ∞

culoare[u]←alb

d[s] ← 0

Pentru i ← 1 **la** n **repetă**

Pentru fiecare a∈ A **repetă**

Relaxează(u,v)

Pentru fiecare a∈ A **repetă**

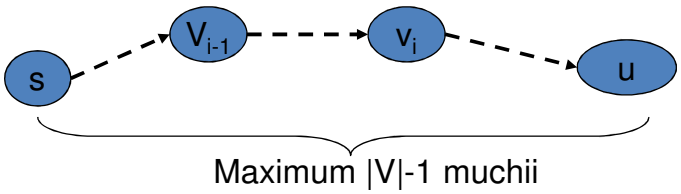
dacă d[v]>d[u]+w(u,v) atunci fail (”ciclu negativ”)

Corectitudine(I)

- G=(V,E), w:E->$\Re$ functie de cost asociata; daca G nu contine ciclu de cost negativ atunci dupa |n|-1 iteratii ale relaxarii fiecarei muchii d[v]=δ(s,v) pentru $\forall v \in R(s)$

Corectitudine (II)

La pasul i va fi relaxata muchia v_{i-1},v_i



Pentru i ← 1 **la** n **repetă**
    Pentru fiecare a∈ A **repetă**
        Relaxează(u,v)

Complexitate Bellman-Ford

- cazul defavorabil:
  - **Pentru** i ← 1 **la** n **repetă** —— n
    Pentru fiecare a∈ A **repetă** —— a *
      Relaxează(u,v) =>O(na)

Optimizari Bellman-Ford

- Observatie!
  - Daca d[v] nu se modifica la pasul i atunci nu trebuie sa relaxam niciuna din muchiile care pleaca din v la pasul i+1
  - => pastram o coada cu varfurile modificate (o singura copie)

Bellman-Ford optimizat

- function BellmanFordOpt(G,s)
  - for each v in V[G]
    - d[v]= ∞;
    - p[v]=null;
    - marked[v]=false;
    - Q=∅;
  - d[s]=0; marked[s]=true;Enqueue(Q,s)
  - while(Q!=∅)
    - u=Dequeue(Q); marked[u]=false;
    - for each (u,v) in E[G]
      - if d[v]>d[u]+w(u,v)
        - » d[v]=d[u]+w(u,v);
        - » p[v]=u;
        - » if(marked[v]==false) {marked[v]=true; Enqueue(Q,v);}
- Obs: nu mai detecteaza cicluri negative!

Floyd-Warshall (Roy-Floyd)

- Algoritm prin care se calculeaza distantele minime intre oricare 2 noduri dintr-un graf
- Exemplu clasic de programare dinamica
- Idee: la pasul k se calculeaza cel mai bun cost intre u si v folosind cel mai bun cost u..i si cel mai bun cost i..v calculat pana in momentul respectiv
- se aplica pentru grafuri ce nu contin cicluri de cost negativ

Notații

- G=(V,E); V={1,2,..n}
- w:VxV->$\Re$; w(i,i)=0; w(i,j)=∞ daca (i,j)$\notin$ E
- d^k(i,j)=costul drumului i..j construit a.i. drumul trece doar prin noduri din multimea {1,2,..,k}
- δ(i,j)= costul drumului optim i..j
- δ^k(i,j)= costul drumului optimi i..j ce trece doar prin noduri din multimea {1,2,..,k}
- p^k(i,j)= predecesorul lui j pe drumul i..j ce trece doar prin noduri din multimea {1,2,..,k}

Floyd - Warshall

- Cum calculam $d^k(i,j)$
  - $d^0(i,j)=w(i,j)$
  - $d^k(i,j)=\min\{d^{k-1}(i,j), d^{k-1}(i,k)+d^{k-1}(k,j)\}$ pt $0<k\leq n$
- $d^n(i,j)=\delta(i,j)$

Algoritm Floyd-Warshall

- FW(G)
- for(i=1;i<=n;i++)
  - for(j=1;j<=n;j++)
    - $d^0(i,j)=w(i,j)$
    - if(w(i,j)==∞)
      - $p^0(i,j)$=null;
    - else $p^0(i,j)=i$;
- for(k=1;k<=n;k++)
  - for(i=1;i<=n;i++)
    - for(j=1;j<=n;j++)
      - if($d^{k-1}(i,j)>d^{k-1}(i,k)+d^{k-1}(k,j)$)
        - » $d^{k-1}(i,j)=d^{k-1}(i,k)+d^{k-1}(k,j)$
        - » $p^k(i,j)= p^{k-1}(k,j)$;
      - else
        - » $d^{k-1}(i,j)=d^{k-1}(i,j)$
        - » $p^k(i,j)= p^{k-1}(i,j)$;

Observatie

- $d^k(k,j)=d^{k-1}(k,k)+d^{k-1}(k,j)=d^{k-1}(k,j)$
- $d^k(i,k)=d^{k-1}(i,k)+d^{k-1}(k,k)=d^{k-1}(i,k)$
- => putem folosi o singura matrice in loc de n
- => algoritm modificat pentru a fi in $O(n^2)$ spatiu

Algoritm Floyd-Warshall

- FW2(G)
- for(i=1;i<=n;i++)
  - for(j=1;j<=n;j++)
    - d(i,j)=w(i,j)
    - if(w(i,j)==∞)
      - tată(i,j)=null;
    - else p(i,j)=i;
- for(k=1;k<=n;k++)
  - for(i=1;i<=n;i++)
    - for(j=1;j<=n;j++)
      - if(d(i,j)>d(i,k)+d(k,j))
        - » d(i,j)=d(i,k)+d(k,j)
        - » tată(i,j)= p(k,j);

Algoritmul lui Johnson

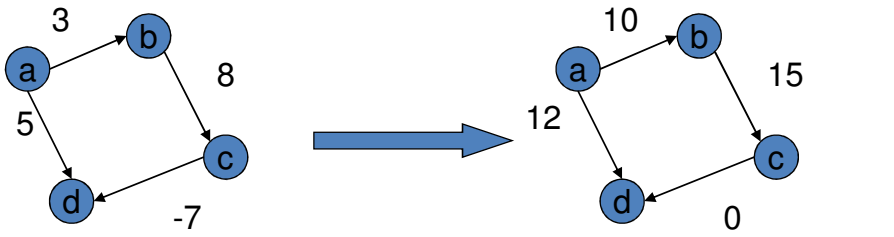
- pentru grafuri rare
- foloseste liste de adiacenta
- bazat pe Dijkstra si Bellman-Ford
- complexitate: $O(n^2 \log n + na)$
  - mai buna decat Floyd-Warshall pentru grafuri rare

Idee algoritm Johnson

- daca graful are numai muchii pozitive
  - se aplica Dijkstra pentru fiecare nod – cost $n(n\log n+a) = n^2\log n+na$
- altfel se calculeaza costuri pozitive pentru fiecare muchie mentinand proprietatile
  - $w1(u,v)>0\ \forall(u,v)\in E$
  - p este drum minim utilizand w $\Leftrightarrow$ p este drum minim utilizand w1

Constructie w1

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



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

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

nu functioneaza

Constructie w1

- w1(u..v)=w(u..v)+h(u)-h(v)
- unde H:V->$\Re$
- se adauga un nod s;
- se uneste s cu toate nodurile grafului prin muchii de cost 0
- se aplica Bellman-Ford pe acest graf=> h(v)=δ(s,v)
- =>w1(u,v)=w(u,v)+h(u)-h(v)

Algoritm Johnson

- Johnson(G)
  - G'=(V',E'); V'=V∪S; E'=E∪(s,u), ∀u∈V; w(s,u)=0;
  - if(bf(G))==false
    - fail "ciclu negativ"
  - else
    - foreach v∈V'
      - h(v)=δ(s,v); //calculat prin BF
    - foreach (u,v)∈E'
      - w1(u,v)=w(u,v)+h(u)-h(v)
    - foreach(u∈V')
      - Dijkstra(G',w1,u)
      - foreach (v∈V')
        - » d(u,v)=δ1(u,v)+h(v)-h(u)

Concluzii Floyd-Warshall - Johnson

- Algoritmi ce gasesc drumurile minime intre oricare 2 noduri din graf
- functioneaza pe grafuri cu muchii ce au costuri negative (dar care nu au cicluri de cost negativ)
- floyd-warshall optim pentru grafuri dese
- johnson mai bun pentru grafuri rare

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*,