

Laborator 09: Drumuri minime

Obiective laborator

- Înțelegerea conceptelor de cost, relaxare a unei muchii, drum minim
- Prezentarea și asimilarea algoritmilor pentru calculul drumurilor minime

Importanță – aplicații practice

Algoritmii pentru determinarea drumurilor minime au multiple aplicații practice și reprezintă clasa de algoritmi pe grafuri cel mai des utilizată:

- Rutare în cadrul unei rețele (telefonice, de calculatoare etc.)
- Găsirea drumului minim dintre două locații (Google Maps, GPS etc.)
- Stabilirea unei agende de zbor în vederea asigurării unor conexiuni optime
- Asignarea unui peer / server de fișiere în funcție de metricile definite pe fiecare linie de comunicație

Concepte

Costul unei muchii și al unui drum

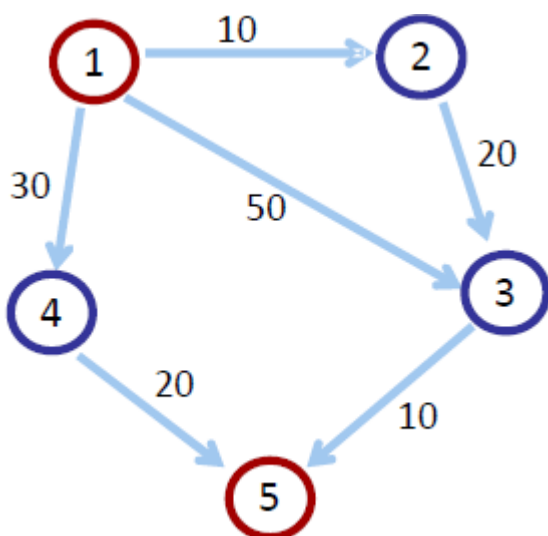
Fiind dat un graf orientat $G = (V, E)$, se considera funcția $w: E \rightarrow W$, numită funcție de cost, care asociază fiecărei muchii o valoare numerică. Domeniul funcției poate fi extins, pentru a include și perechile de noduri între care nu există muchie directă, caz în care valoarea este $+\infty$. Costul unui drum format din muchiile $p_1p_2 \ p_2p_3 \ \dots \ p_{n-1}p_n$, având costurile $w_{12}, w_{23}, \dots, w_{(n-1)n}$, este suma $w = w_{12} + w_{23} + \dots + w_{(n-1)n}$.

În exemplul alăturat, costul drumului de la nodul 1 la 5 este:

drumul 1: $w_{14} + w_{45} = 30 + 20 = 50$

drumul 2: $w_{12} + w_{23} + w_{35} = 10 + 20 + 10 = 40$

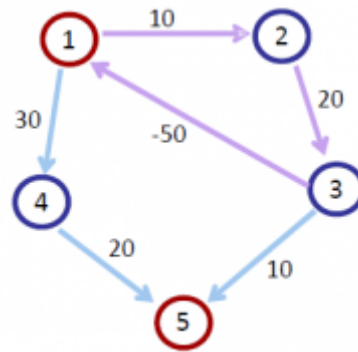
drumul 3: $w_{13} + w_{35} = 50 + 10 = 60$



Drumul de cost minim

Costul minim al drumului dintre două noduri este minimul dintre costurile drumurilor existente între cele două noduri. În exemplul de mai sus, drumul de cost minim de la nodul 1 la 5 este prin nodurile 2 și 3. Deși, în cele mai multe cazuri, costul este o funcție cu valori nenegative, există situații în care un graf cu muchii de cost negativ are relevanță

practica. O parte din algoritmi pot determina drumul corect de cost minim inclusiv pe astfel de grafuri. Totuși, nu are sens căutarea drumului minim în cazurile în care graful conține cicluri de cost negativ – un drum minim ar avea lungimea infinită, întrucât costul său s-ar reduce la fiecare reparcurgere a ciclului:



În exemplul alăturat, ciclul $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ are costul -20.

drumul 1: $w_{12} + w_{23} + w_{35} = 10 + 20 + 10 = 40$

drumul 2: $(w_{12} + w_{23} + w_{31}) + w_{12} + w_{23} + w_{35} = -20 + 10 + 20 + 10 = 20$

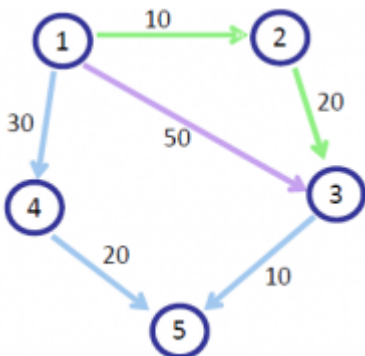
drumul 3: $(w_{12} + w_{23} + w_{31}) + (w_{12} + w_{23} + w_{31}) + w_{12} + w_{23} + w_{35} = -20 + (-20) + 10 + 20 + 10 = 0$

Relaxarea unei muchii

Relaxarea unei muchii $v_1 - v_2$ constă în a testa dacă se poate reduce costul ei, trecând printr-un nod intermediar u . Fie w_{12} costul inițial al muchiei de la v_1 la v_2 , w_{1u} costul muchiei de la v_1 la u , și w_{u2} costul muchiei de la u la v_2 . Dacă $w_{12} > w_{1u} + w_{u2}$, muchia directă este înlocuită cu succesiunea de muchii $v_1 - u, u - v_2$.

În exemplul alăturat, muchia de la 1 la 3, de cost $w_{13} = 50$, poate fi relaxată la costul 30, prin nodul intermediar $u = 2$, fiind înlocuită cu succesiunea w_{12}, w_{23} .

Toți algoritmi prezentați în continuare se bazează pe relaxare pentru a determina drumul minim.



Drumuri minime de sursă unică

Algoritmi din această secțiune determină drumul de cost minim de la un nod sursă, la restul nodurilor din graf, pe baza de relaxări repetate.

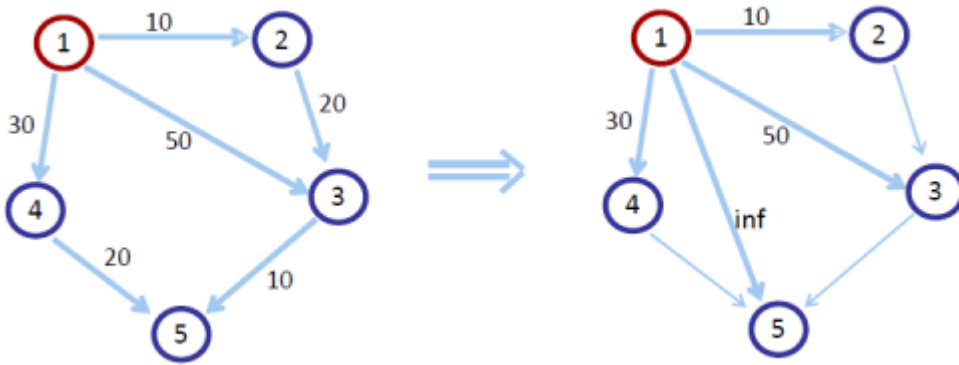
Algoritmul lui Dijkstra

Dijkstra poate fi folosit doar în grafuri care au toate muchiile nenegative.

Algoritmul este de tip Greedy:

optimul local căutat este reprezentat de costul drumului dintre nodul sursă s și un nod v . Pentru fiecare nod se reține un cost estimat $d[v]$, inițializat la început cu costul muchiei $s \rightarrow v$, sau cu $+\infty$, dacă nu există muchie.

În exemplul următor, sursa s este nodul 1. Inițializarea va fi:



Aceste drumuri sunt îmbunătățite la fiecare pas, pe baza celorlalte costuri estimate.

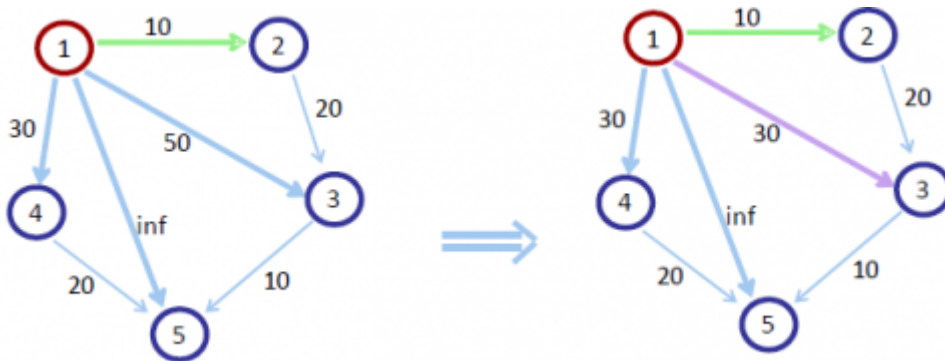
Algoritmul selectează, în mod repetat, nodul u care are, la momentul respectiv, costul estimat minim (fata de nodul sursa). În continuare, se încearcă să se relaxeze restul costurilor $d[v]$. Dacă $d[v] \geq d[u] + w_{uv}$, $d[v]$ ia valoarea $d[u] + w_{uv}$.

Pentru a ține evidența muchiilor care trebuie relaxate, se folosesc două structuri: S (mulțimea de vârfuri deja vizitate) și Q (o coadă cu priorități, în care nodurile se afla ordonate după distanța față de sursă) din care este mereu extras nodul aflat la distanța minimă. În S se afla inițial doar sursa, iar în Q doar nodurile spre care există muchie directă de la sursă, deci care au $d[\text{nod}] < +\infty$.

În exemplul de mai sus, vom inițializa $S = \{1\}$ și $Q = \{2, 4, 3\}$.

La primul pas este selectat nodul 2, care are $d[2] = 10$.

Singurul nod pentru care $d[\text{nod}]$ poate fi relaxat este 3 : $d[3] = 50 > d[2] + w_{23} = 10 + 20 = 30$



După primul pas, $S = \{1, 2\}$ și $Q = \{4, 3\}$.

La următorul pas este selectat nodul 4, care are $d[4] = 30$.

Pe baza lui, se poate modifica $d[5]$: $d[5] = +\infty > d[4] + w_{45} = 30 + 20 = 50$



După al doilea pas, $S = \{1, 2, 4\}$ și $Q = \{3, 5\}$.

La următorul pas este selectat nodul 3, care are $d[3] = 30$, și se modifică din nou $d[5]$: $d[5] = 50 > d[3] + w_{35} = 30 + 10 = 40$.

Algoritmul se încheie când coada Q devine vidă, sau când S conține toate nodurile. Pentru a putea determina și muchiile din care este alcătuit drumul minim căutat, nu doar costul sau final, este necesar să reținem un vector de

părinți P. Pentru nodurile care au muchie directă de la sursă, P[nod] este inițializat cu sursa, pentru restul cu null.

Pseudocodul pentru determinarea drumului minim de la o sursă către celelalte noduri utilizând algoritmul lui Dijkstra este:

```
Dijkstra(sursa, dest):
introdu sursa în Q
d[sursa] = 0
d[nod] = +∞ // pentru orice nod != sursa
P[nod] = null // pentru orice nod din V

// relaxari succesive
cat timp Q nu e vida
    u = extrage_min (Q)
    selectat(u) = true
    foreach nod in vecini[u] // (*)
        /* dacă drumul de la sursa la nod prin u este mai mic decât cel curent */
        dacă not selectat(nod) și d[nod] > d[u] + w[u, nod]
            // actualizează distanța și părinte
            d[nod] = d[u] + w[u, nod]
            P[nod] = u
            /* actualizează poziția nodului în coada prioritara */
            actualizează (Q,nod)

// găsirea drumului efectiv
Initializează Drum = {}
nod = P[dest]
cat timp nod != null
    inserează nod la începutul lui Drum
    nod = P[nod]
```

Reprezentarea grafului ca matrice de adiacență duce la o implementare ineficientă pentru orice graf care nu este complet, datorită parcurgerii vecinilor nodului u, din linia (*), care se va executa în $|V|$ pași pentru fiecare extragere din Q, iar pe întreg algoritmul vor rezulta $|V|^2$ pași. Este preferată reprezentarea grafului cu liste de adiacență, pentru care numărul total de operații cauzate de linia (*) va fi egal cu $|E|$. Complexitatea algoritmului este $O(|V|^2 + |E|)$ în cazul în care coada cu prioritate este implementată ca o căutare liniară. În acest caz funcția `extrage_min` se executa în timp $O(|V|)$, iar `actualizează(Q)` în timp $O(1)$.

O variantă mai eficientă este implementarea cozii ca heap binar. Funcția `extrage_min` se va executa în timp $O(\lg|V|)$; funcția `actualizează(Q)` se va executa tot în timp $O(\lg|V|)$, dar trebuie cunoscută poziția cheii nod în heap, adică heapul trebuie să fie indexat. Complexitatea obținută este $O(|E|\lg|V|)$ pentru un graf conex.

Cea mai eficientă implementare se obține folosind un heap Fibonacci pentru coada cu prioritate:

Aceasta este o structură de date complexă, dezvoltată în mod special pentru optimizarea algoritmului Dijkstra, caracterizată de un timp amortizat de $O(\lg|V|)$ pentru operația `extrage_min` și numai $O(1)$ pentru `actualizează(Q)`. Complexitatea obținută este $O(|V|\lg|V| + |E|)$, foarte bună pentru grafuri rare.

Algoritmul Bellman – Ford

Algoritmul Bellman Ford poate fi folosit și pentru grafuri ce conțin muchii de cost negativ, dar nu poate fi folosit pentru grafuri ce conțin cicluri de cost negativ (când căutarea unui drum minim nu are sens).

Cu ajutorul său putem afla dacă un graf conține cicluri. Algoritmul folosește același mecanism de relaxare ca și Dijkstra, dar, spre deosebire de acesta, nu optimizează o soluție folosind un criteriu de optim local, ci parcurge fiecare muchie de un număr de ori egal cu numărul de noduri și încearcă să o relaxeze de fiecare dată, pentru a îmbunătăți distanța până la nodul destinație al muchiei curente.

Motivul pentru care se face acest lucru este că drumul minim dintre sursă și orice nod destinație poate să treacă prin maximum $|V|$ noduri (adică toate nodurile grafului), respectiv $|V|-1$ muchii; prin urmare, relaxarea tuturor muchiilor de $|V|-1$ ori este suficientă pentru a propaga până la toate nodurile informația despre distanța minimă de la sursă.

Dacă, la sfârșitul acestor $|E| \cdot (|V|-1)$ relaxări, mai poate fi îmbunătățită o distanță, atunci graful are un ciclu de cost negativ și problema nu are soluție.

Menținând notațiile anterioare, pseudocodul algoritmului este:

```
BellmanFord(sursa):
d[sursa] = 0
d[nod] = +∞ // pentru orice nod != sursa
p[nod] = null // pentru orice nod din V
```

```

// relaxari succesive
// cum in initializare se face o relaxare (daca exista drum direct de la sursa la nod =>
// d[nod] = w[sursa, nod]) mai sunt necesare |V|-2 relaxari
for i = 1 to |V|-2
    foreach (u, v) in E // E = multimea muchiilor
        daca d[v] > d[u] + w(u,v)
            d[v] = d[u] + w(u,v)
            p[v] = u;

// daca se mai pot relaxa muchii
foreach (u, v) in E
    daca d[v] > d[u] + w(u,v)
        fail ("exista cicluri negativ")

```

Complexitatea algoritmului este $O(|E|*|V|)$.

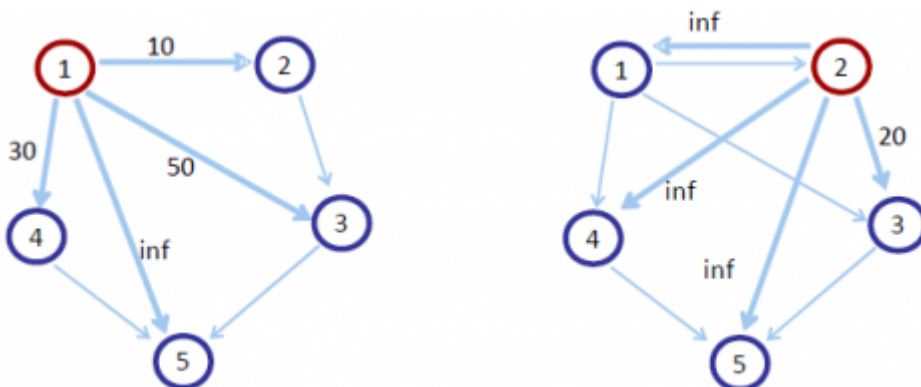
Drumuri minime intre oricare doua noduri

Floyd-Warshall

Algoritmii din aceasta secțiune determina drumul de cost minim dintre oricare doua noduri dintr-un graf. Pentru a rezolva aceasta problema s-ar putea aplica unul din algoritmii de mai sus, considerând ca sursa fiecare nod, pe rând, dar o astfel de abordare ar fi ineficienta.

Algoritmul Floyd-Warshall (intalnit si sub numele de Roy-Floyd) compara toate drumurile posibile din graf dintre fiecare 2 noduri, si poate fi utilizat si in grafuri cu muchii de cost negativ.

Estimarea drumului optim poate fi reținut într-o structura tridimensională $d[v1, v2, k]$, cu semnificația – costul minim al drumului de la $v1$ la $v2$, folosind ca noduri intermediare doar noduri pana la nodul k . Daca nodurile sunt numerotate de la 1, atunci $d[v1, v2, 0]$ reprezintă costul muchiei directe de la $v1$ la $v2$, considerând $+\infty$ daca aceasta nu exista. Exemplu, pentru $v1 = 1$, respectiv 2:



Pornind cu valori ale lui k de la 1 la $|V|$, ne interesează să găsim cea mai scurta cale de la fiecare $v1$ la fiecare $v2$ folosind doar noduri intermediare din mulțimea $\{1, \dots, k\}$. De fiecare data, comparăm costul deja estimat al drumului de la $v1$ la $v2$, deci $d[v1, v2, k-1]$ obținut la pasul anterior, cu costul drumurilor de la $v1$ la k si de la k la $v2$, adică $d[v1, k, k-1] + d[k, v2, k-1]$, obținute la pasul anterior. Atunci, $d[v1, v2, |V|]$ va conține costul drumului minim de la $v1$ la $v2$.

Pseudocodul acestui algoritm este:

```

FloydWarshall(G):
n = |V|
int d[n, n, n]
foreach (i, j) in (1..n, 1..n)
    d[i, j, 0] = w[i, j] // costul muchiei, sau infinit
for k = 1 to n
    foreach (i, j) in (1..n, 1..n)
        d[i, j, k] = min(d[i, j, k-1], d[i, k, k-1] + d[k, j, k-1])

```

Complexitatea temporală este $O(|V|^3)$, iar cea spațială este tot $O(|V|^3)$. O complexitate spațială cu un ordin mai mic se obține observând ca la un pas nu este nevoie decât de matricea de la pasul precedent $d[i, j, k-1]$ si cea de la pasul curent $d[i, j, k]$. O observație și mai bună este că, de la un pas $k-1$ la k , estimările lungimilor nu pot decât să scadă, deci putem să lucrăm pe o singură matrice. Deci, spațiul de memorie necesar este de dimensiune $|V|^2$.

Rescris, pseudocodul algoritmului arata astfel:

```
FloydWarshall(G):
n = |V|
int d[n, n]
foreach (i, j) in (1..n, 1..n)
    d[i, j] = w[i, j] // costul muchiei, sau infinit
for k = 1 to n
    foreach (i, j) in (1..n, 1..n)
        d[i, j] = min(d[i, j], d[i, k] + d[k, j])
```

Pentru a determina drumul efectiv, nu doar costul acestuia, avem doua variante:

1. Se retine o structura de părinți, similara cu cea de la Dijkstra, dar, bineînțeles, bidimensională.
2. Se folosește divide et impera astfel:

- se caută un pivot k astfel încât $\text{cost}[i][j] = \text{cost}[i][k] + \text{cost}[k][j]$
- se apelează funcția recursiv pentru ambele drumuri $\rightarrow (i, k), (k, j)$
- dacă pivotul nu poate fi găsit, afișăm i
- după terminarea funcției recursie afișăm extremitatea dreaptă a drumului

Cazuri speciale

1. Dacă avem un graf **neorientat, fara cicluri** (un arbore), exista un singur drum între oricare doua noduri, care poate fi aflat printr-o simplă parcurgere DFS. Folosind diferite preprocesări [8],[9], putem calcula distanța între oricare doua noduri în timp constant, $O(1)$.
2. Dacă avem un graf **orientat, fara cicluri** (un DAG [10]), putem să relaxăm muchiile nodurilor, parcurgându-le pe acestea în ordinea dată de sortarea topologică. $O(|V|+|E|)$
3. Dacă avem un graf unde toate muchiile au **cost egal**, putem afla distanța minimă de la un nod sursă la orice alt nod printr-o parcurgere BFS. (de asemenea, ținând cont de faptul că pot exista mai multe drumuri până la un anumit nod). $O(|V|+|E|)$
4. Pentru grafuri orientate, **rare** (relativ puține muchii), putem folosi algoritmul lui Johnson([11]) pentru calcularea distanței minime de la un nod, la oricare alt nod. $O(|V|^2 \log |V| + |V||E|)$

Concluzii

▪ Dijkstra*

- calculează drumurile minime de la o sursă către celelalte noduri
- nu poate fi folosit dacă există muchii de cost negativ
- complexitate minimă $O(|V| \lg |V| + |E|)$ utilizând heapuri Fibonacci;

▪ Bellman – Ford

- calculează drumurile minime de la o sursă către celelalte noduri
- detectează existența ciclurilor de cost negativ
- complexitate $O(|V| * |E|)$

▪ Floyd – Warshall

- calculează drumurile minime între oricare doua noduri din graf
- poate fi folosit în grafuri cu cicluri de cost negativ, dar nu le detectează
- complexitate $O(|V|^3)$

Referințe:

[1] http://en.wikipedia.org/wiki/Dijkstra's_algorithm [http://en.wikipedia.org/wiki/Dijkstra's_algorithm]

[2] http://en.wikipedia.org/wiki/Bellman-Ford_algorithm [http://en.wikipedia.org/wiki/Bellman-Ford_algorithm]

[3] http://www.algorithmist.com/index.php/Floyd-Warshall's_Algorithm [http://www.algorithmist.com/index.php/Floyd-Warshall's_Algorithm]

[4] http://en.wikipedia.org/wiki/Binary_heap [http://en.wikipedia.org/wiki/Binary_heap]

[5] http://en.wikipedia.org/wiki/Fibonacci_heap [http://en.wikipedia.org/wiki/Fibonacci_heap]

[6] T. Cormen, C. Leiserson, R. Rivest, C. Stein – Introducere în Algoritmi

[7] C. Giumale – Introducere în analiza algoritmilor

[8] http://en.wikipedia.org/wiki/Range_Minimum_Query [http://en.wikipedia.org/wiki/Range_Minimum_Query]

[9] http://en.wikipedia.org/wiki/Lowest_common_ancestor [http://en.wikipedia.org/wiki/Lowest_common_ancestor]

[10] http://en.wikipedia.org/wiki/Directed_acyclic_graph [http://en.wikipedia.org/wiki/Directed_acyclic_graph]

[11] http://en.wikipedia.org/wiki/Johnson%27s_algorithm [http://en.wikipedia.org/wiki/Johnson%27s_algorithm]

Resurse

- Varianta PDF a laboratorului

[http://cs.curs.pub.ro/2015/pluginfile.php/32415/mod_resource/content/1/Lab%208%20Drumuri%20minime.pdf]

Exercitii

În acest laborator vom folosi scheletul de laborator din arhiva skel-lab09.zip.

Înainte de a rezolva exercitiile, asigurați-vă că ați citit și înțeles toate precizările din secțiunea Precizări laboratoare 07-12 [https://ocw.cs.pub.ro/courses/pa/skel_graph].

Prin citirea acestor precizări vă asigurăm că:

- cunoașteți **convențiile** folosite
- evitați **buguri**
- evitați **depunctări** la lab/teme/test

Dijkstra

Se da un graf **orientat** cu **n** noduri și **m** arce. Graful are pe arce **costuri pozitive**.

Folosiți **Dijkstra** pentru a găsi **costul minim (lungimea minimă)** a unui drum de la o sursă dată (**source**) la toate celelalte $n - 1$ noduri din graf.

Costul / lungimea unui drum este suma costurilor/lungimilor arcelor care compun drumul.

Restricții și precizări:

- $n \leq 50.000$
- $m \leq 2.5 \cdot 10^5$
- $0 \leq c \leq 20.000$, unde c este costul/lungimea unui arc
- timp de execuție
 - C++: 1s
 - Java: 2s

Rezultatul se va returna sub forma unui vector **d** cu **n + 1** elemente.

Convenție:

- **d[node]** = costul minim / lungimea minimă a unui drum de la **source** la nodul **node**
- **d[source] = 0**
- **d[node] = -1**, dacă nu se poate ajunge de la **source** la **node**

d[0] nu este folosit, deci să fie inițializat cu 0! (am păstrat indexarea nodurilor de la 1)

Bellman-Ford

Se da un graf **orientat conex** cu **n** noduri si **m** arce. Graful are pe arce **costuri pozitive sau negative**.

Folositi **Bellman-Ford** pentru a gasi **costul minim (lungimea minima)** a unui drum de la o sursa data (**source**) la toate celelalte $n - 1$ noduri din graf. In caz ca se va detecta un ciclu de cost negativ, se va semnala acest lucru.

Costul / lungimea unui drum este suma costurilor/lungimilor arcelor care compun drumul.

Restrictii si precizari:

- $n \leq 50.000$
- $m \leq 2.5 * 10^5$
- $-1.000 \leq c \leq +1.000$, unde c este costul/lungimea unui arc
- timp de executie
 - C++: 1s
 - Java: 2s
- Pentru punctaj maxim, implementarea din laborator trebuie sa treaca toate testele, cu exceptia ultimelor 2, pe care va lua TLE. Pentru cei curiosi, exista si o implementare mai eficienta a algoritmului, oarecum similara cu cea de la Dijkstra (pentru mai multe detalii: <https://infoarena.ro/problema/bellmanford> [<https://infoarena.ro/problema/bellmanford>])

Rezultatul se va returna sub forma unui vector **d** cu **n + 1** elemente.

Conventie:

- **d[node]** = costul minim / lungimea minima a unui drum de la **source** la nodul **node**
- **d[source] = 0**
- **d[node] = -1**, daca nu se poate ajunge de la **source** la **node**

d[0] nu este folosit, deci ca fi initializat cu 0! (am pastrat indexarea nodurilor de la 1)

ATENTIE!!! Este posibil ca un astfel de graf sa aiba ciclu de cost negativ. In cazul detectarii unui ciclu de cost negativ, functia voastra va returna un vector gol! (**std::vector<int>()** / **{}** sau **new ArrayList<Integer>()**).

RoyFloyd

Se da un graf **orientat** cu **n** noduri. Graful are **costuri pozitive** pe arce.

Se da **matricea ponderilor**, se cere **matricea drumurilor minime**.

Restrictii si precizari:

- $n \leq 100$
- $0 \leq c \leq 1.000$, unde c este costul unui arc
- daca **nu exista muchie** intre o pereche de noduri x si y , distanta de la nodul x la nodul y din **matricea ponderilor** va fi 0
- daca dupa aplicarea algoritmului **nu se gaseste drum** pentru o pereche de noduri x si y , se va considera **distanta** dintre ele egala cu 0 (se stocheaza in **matricea distantelor** valoarea 0)
- drumul de la nodul i la nodul i are lungime 0 (prin conventie)
- timp de executie
 - C++: 1s
 - Java: 2s

Rezultatul se va **stoca** in matricea **d** declarata in schelet! Algoritmul vostru trebuie doar sa o populeze corect, tinand cont ca nodurile sunt indexate de la 1.

BONUS

Pentru exercitiul cu Dijkstra, **reconstituiti** drumul de lungime minima **source** la celelalte noduri din graf.

Folositi fisierele din skel-lab09.zip din folderul **bonus** ca sa va verificati.

Pentru fisierul mare, folositi comanda **diff** cu parametrul **-w** pentru a verifica usor ca outputul coincide.

Extra

Rezolvati problema rfinv [<https://infoarena.ro/problema/rfinv>] pe infoarena.

Rezolvati problema coach [<https://infoarena.ro/problema/coach>] pe infoarena.

Rezolvati problema rf [<https://infoarena.ro/problema/rf>] pe infoarena.

Rezolvati problema TODO [<https://infoarena.ro/problema/TODO>] pe infoarena.