

Algoritmica grafurilor - Cursul 4

25 ottobre 2019

1 Probleme de drumuri în digrafuri

- Drumuri de cost minim - Problema P2 pentru dags: sortarea topologică
- Drumuri de cost minim - Problema P2 pentru costuri nenegative
- Drumuri de cost minim - Problema P2 pentru costuri reale
- Drumuri de cost minim - Rezolvarea problem P3
- Înmulțirea (rapidă) a matricilor

2 Exerciții pentru seminarul de săptămâna următoare

Un **digraf aciclic** (**dag**) este un digraf fără circuite.

O **ordonare topologică** a (nodurilor) unui digraf $G = (V, E)$, cu $|G| = n$, este o funcție injectivă $ord : V \rightarrow \{1, 2, \dots, n\}$ ($ord[u] =$ numărul de ordine al nodului u , $\forall u \in V$) așa încât

$$uv \in E \Rightarrow \text{ord}[u] < \text{ord}[v], \forall uv \in E.$$

Lema 1

$G = (V, E)$ este un digraf fără circuite dacă și numai dacă admite o ordonare topologică.

Proof: " \Leftarrow " Fie ord o ordonare topologică a lui G . Dacă $C = (u_1, u_1 u_2, u_2, \dots, u_k, u_k u_1, u_1)$ este un circuit în G , atunci, din proprietatea funcției ord , obținem contradicția

$$ord[u_1] < ord[u_2] < \dots < ord[u_k] < ord[u_1].$$

" \Rightarrow " Fie $G = (V, E)$ un digraf de ordin n fără circuite. Arătăm prin inducție după n că G are o ordonare topologică. Pasul induc-tiv:

```
let  $v_0 \in V$ ;  
while ( $d_G^-(v_0) \neq 0$ ) do  
  take  $u \in V$  așa încât  $uv_0 \in E$ ;  
   $v_0 \leftarrow u$ ;  
return  $v_0$ .
```

Evident, deoarece G nu are circuite și V este finită, algoritmul se termină și în nodul returnat, v_0 , nu mai intră arce. Digraful $G - v_0$ nu are circuite și din ipoteza inductivă are o ordonare topologică ord' . Ordonarea topologică a lui G este

$$ord[v] = \begin{cases} 1, & \text{dacă } v = v_0 \\ ord'[v] + 1, & \text{dacă } v \in V \setminus \{v_0\}. \end{cases}$$



Din demonstrația de mai sus obținem următorul algoritm pentru recunoașterea unui dag și construcția unei ordonări topologice în cazul instanțelor "yes":

Input: $G = (\{1, \dots, n\}, E)$ digraf cu $|E| = m$.

Output: "yes" dacă G este dag și o ordonare topologică ord ; "no" altfel.

Drumuri de cost minim - Problema P2 pentru dags

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

```
construct the array  $d_G^-[u], \forall u \in V$ ;  
 $count \leftarrow 0$ ;  $S \leftarrow \{u \in V : d_G^-[u] = 0\}$ ; //  $S$  este o coadă sau o stivă;  
while ( $S \neq \emptyset$ ) do  
     $v \leftarrow pop(S)$ ;  $count++$ ;  $ord[v] \leftarrow count$ ;  
    for ( $w \in A[v]$ ) do  
         $d_G^-[w]--$ ;  
        if ( $d_G^-[w] = 0$ ) then  
            push( $S, w$ );  
// complexitatea timp  $\mathcal{O}(n + m)$ ;  
if ( $count = n$ ) then  
    return "yes" ord;  
return "no";
```

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Problema P2 pentru dags

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

P2 Dat $G = (V, E)$ digraf; $a : E \rightarrow \mathbb{R}$; $s \in V$.

Determină $P_{si}^* \in \mathcal{P}_{si}, \forall i \in V$, a. î. $a(P_{si}^*) = \min \{a(P_{si}) : P_{si} \in \mathcal{P}_{si}\}$.

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

P2 cu $G = (\{1, \dots, n\}, E)$ dag, cu $\text{ord}[i] = i, \forall i \in V$, și $s = 1$.

Condiția (I) este satisfăcută și sistemul (B) se rezolvă prin "substituție".

$u_1 \leftarrow 0; \text{before}[1] \leftarrow 0;$

for ($i = \overline{2, n}$) **do**

$u_i \leftarrow \infty; \text{before}[i] \leftarrow 0;$

for ($j = \overline{1, i-1}$) **do**

if ($u_i > u_j + a_{ji}$) **then**

$u_i \leftarrow u_j + a_{ji}; \text{before}[i] \leftarrow j;$

// complexitatea timp $\mathcal{O}(n^2)$;

Drumuri de cost minim - Problema P2 pentru costuri nenegative

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

P2 Dat $G = (V, E)$ digraf; $a : E \rightarrow \mathbb{R}$; $s \in V$.

Determină $P_{si}^* \in \mathcal{P}_{si}$, $\forall i \in V$, a. î. $a(P_{si}^*) = \min \{a(P_{si}) : P_{si} \in \mathcal{P}_{si}\}$

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

P2 cu $a(e) \geq 0, \forall e \in E$.

Condiția (I) este satisfăcută și sistemul (B) se poate rezolva cu **algoritmul lui Dijkstra**, care are următorul invariant: $S \subseteq V$ și

$$(D) \quad \begin{cases} \forall i \in S & u_i = \min \{a(P_{si}) : P_{si} \in \mathcal{P}_{si}\} \\ \forall i \in V \setminus S & u_i = \min \{a(P_{si}) : P_{si} \in \mathcal{P}_{si}, V(P_{si}) \setminus S = \{i\}\} \end{cases}$$

Inițial $S = \{s\}$ și în fiecare dintre cei $n - 1$ pași un nod nou este adăugat la S , obținând $S = V$. Astfel, datorită invariantului (D) de mai sus, **P2** este rezolvată.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Algoritmul lui Dijkstra

```
 $S \leftarrow \{s\}; \text{before}[s] \leftarrow 0; u_s \leftarrow 0;$   
for ( $i \in V \setminus \{s\}$ ) do  
     $u_i \leftarrow a_{si}; \text{before}[i] \leftarrow s; //$  (are loc D)  
while ( $S \neq V$ ) do  
    find  $j^* \in V \setminus S$  s. t.  $u_{j^*} = \min \{u_j : j \in V \setminus S\};$   
     $S \leftarrow S \cup \{j^*\};$   
    for ( $j \in V \setminus S$ ) do  
        if ( $u_j > u_{j^*} + a_{j^*j}$ ) then  
             $u_j \leftarrow u_{j^*} + a_{j^*j}; \text{before}[j] \leftarrow j^*;$ 
```

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Demonstrația corectitudinii algoritmului lui Dijkstra

Deoarece (D) are loc după pasul de inițializare, trebuie să demonstrăm că dacă (D) are loc înaintea iterației while curente, atunci (D) are loc și înaintea iterației următoare.

Fie $S \subseteq V$ și u_1, \dots, u_n satisfăcând (D) înaintea iterației while curente. Arătăm a mai întâi că dacă j^* este astfel încât $u_{j^*} = \min \{u_j : j \in V \setminus S\}$, atunci

$$u_{j^*} = \min \{a(P_{sj^*}) : P_{sj^*} \in \mathcal{P}_{sj^*}\}.$$

Să presupunem că $\exists P_{sj^*}^1 \in \mathcal{P}_{sj^*}$ așa încât $a(P_{sj^*}^1) < u_{j^*}$. Deoarece S și u_i satisfac (D), avem

$$u_{j^*} = \min \{a(P_{sj^*}) : P_{sj^*} \in \mathcal{P}_{sj^*}, V(P_{sj^*}) \setminus S = \{j^*\}\}.$$

Urmează că $V(P_{sj^*}^1) \setminus S \neq \{j^*\}$; fie k primul nod de pe $P_{sj^*}^1$ (începând cu s) așa încât $k \notin S$.

Demonstrația corectitudinii algoritmului lui Dijkstra (cont.)

Atunci $a(P_{sj}^1) = a(P_{sk}^1) + a(P_{kj}^1)$. Din modul de alegere a lui k , avem $V(P_{sk}^1) \setminus S = \{k\}$ și, deoarece (D) este satisfăcută, avem $a(P_{sk}^1) \geq u_k$. Obținem că $u_{j^*} > a(P_{sj}^1) \geq u_k + a(P_{kj}^1) \geq u_k$ (costurile sunt ≥ 0 , deci $a(P_{kj}^1) \geq 0$). Dar aceasta contrazice alegerea lui j^* .

Urmează că în iterația curentă, după asignarea $S \leftarrow S \cup \{j^*\}$, prima parte din (D) are loc.

Bucloa for de după această asignare este necesară pentru a asigura partea a doua din (D) după iterația while: $\forall j \in V \setminus (S \cup \{j^*\})$

$$\min \{a(P_{sj}) : P_{sj} \in \mathcal{P}_{sj}, V(P_{sj}) \setminus (S \cup \{j^*\}) = \{j\}\} =$$

$$\min \{\min \{a(P_{sj}) : P_{sj} \in \mathcal{P}_{sj}, V(P_{sj}) \setminus S = \{j\}\} (= u_j),$$

$$\min \{a(P_{sj}) : P_{sj} \in \mathcal{P}_{sj}, V(P_{sj}) \setminus S = \{j, j^*\}\} (= \alpha_j)\}.$$

Demonstrația corectitudinii algoritmului lui Dijkstra (cont.)

Primul argument din minimul de mai sus este u_j (vechea valoare dinaintea buclei **for**); fie α_j cel de-al doilea argument.

Fie P_{sj}^1 un drum pentru care $\alpha_j = a(P_{sj}^1)$ cu $j^* \in V(P_{sj}^1)$ și $V(P_{sj}^1 \setminus (S \cup \{j^*\})) = \{j\}$; avem $\alpha_j = a(P_{sj^*}^1) + a(P_{j^*j}^1)$. Deoarece am demonstrat că $S \cup \{j^*\}$ satisface prima parte din (D), urmează că $a(P_{sj^*}^1) = u_{j^*}$ și deci $\alpha_j = u_{j^*} + a(P_{j^*j}^1)$.

Dacă $a(P_{j^*j}^1) \neq a_{j^*j}$, urmează că există $i \in V(P_{j^*j}^1) \cap S$, $i \neq j^*$. De unde $u_j \leq a(P_{si}^1) + a(P_{ij}^1) = u_i + a(P_{ij}^1) \leq a(P_{si}^1) + a(P_{ij}^1) = a(P_{sj}^1) = \alpha_j$.

Am obținut că singura posibilitate de a avea $\alpha_j < u_j$ este când $a(P_{j^*j}^1) = a_{j^*j}$, în acest caz $\alpha_j = u_{j^*} + a_{j^*j} < u_j$, care este testul din bucla **for** a algoritmului (u_j este vechea valoare dinaintea buclei **for**).



Drumuri de cost minim - Problema P2 pentru costuri nenegative

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Complexitatea timp a algoritmului lui Dijkstra

Deoarece bucla **for** din cel de-al doilea pas poate fi înlocuită echivalent cu

```
for ( $j \in N_G^+(j^*)$ ) do  
  if ( $u_j > u_{j^*} + a_{j^*j}$ ) then  
     $u_j \leftarrow u_{j^*} + a_{j^*j}$ ; before[ $j$ ]  $\leftarrow j^*$ ;
```

timpul general necesar algoritmului pentru a actualiza valorile u_j este

$$\mathcal{O}\left(\sum_{j^* \in V \setminus \{s\}} d_G^+(j^*)\right) = \mathcal{O}(m).$$

Urmează că complexitatea timp este dominată de secvența de determinări a minimelor u_{j^*} .

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Complexitatea timp a algoritmului lui Dijkstra

- Dacă alegerea minimului u_j^* este făcută prin parcurgerea lui $(u_j)_{j \in V \setminus S}$, atunci algoritmul se execută în $\mathcal{O}((n-1) + (n-2) + \dots + 1) = \mathcal{O}(n^2)$.
- Dacă valorile lui u_j pentru $j \in V \setminus S$ sunt ținute într-o coadă cu priorități (e.g. heap) atunci extragerea fiecărui minim necesită $\mathcal{O}(1)$, dar timpul necesar execuției tuturor reducerilor u_j este în cazul cel mai nefavorabil $\mathcal{O}(m \log n)$ - sunt $\mathcal{O}(m)$ reduceri posibile, fiecare necesitând $\mathcal{O}(\log n)$ pentru întreținerea heap-ului (Johnson, 1977).
- Cea mai bună implementare se obține folosind o grămadă (heap) Fibonacci cu o complexitate timp de $\mathcal{O}(m + n \log n)$ (Fredman & Tarjan, 1984).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Remarci 1

- Pentru a rezolva problema P1 folosind algoritmul lui Dijkstra, adăgăm un test de oprire a execuției când nodul t este introdus în S . în cazul cel mai nefavorabil complexitatea rămâne aceeași.
- O euristică interesantă care dirijează căutarea spre t se obține cu ajutorul unui **estimator consistent**, i. e. o funcție $g : V \rightarrow \mathbb{R}_+$ care satisface condițiile:
 - (i) $\forall i \in V, u_i + g(i) \leq \min \{a(P_{st}) : P_{st} \in \mathcal{P}_{st} \text{ și } i \in V(P_{st})\}$
 - (ii) $\forall ij \in E, g(i) \leq a_{ij} + g(j)$.
- Evident, $g(i) = 0, \forall i$ este un estimator consistent trivial.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Remarci 2

- Dacă $V(G)$ este a mulțime de puncte dintr-un spațiu Euclidean, atunci, luând $g(i) =$ distanța Euclideană de la i la t , obținem un estimator consistent dacă sunt satisfăcute și condițiile din (ii).
- Dacă g este un estimator consistent, atunci alegerea lui j^* în algoritmul lui Dijkstra se face astfel

$$u_{j^*} + g(j^*) = \min \{u_j + g(j) : j \in V \setminus S\}.$$

Corectitudinea demonstrației este similară cu cea dată pentru $g(i) = 0, \forall i$.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Partition Shortest Path (PSP) algorithm

```
 $u_s \leftarrow 0$ ;  $before(s) \leftarrow 0$ ;  $S \leftarrow \emptyset$ ;  $NOW \leftarrow \{s\}$ ;  $NEXT \leftarrow \emptyset$ ;  
while (  $NOW \cup NEXT \neq \emptyset$  ) do  
  while (  $NOW \neq \emptyset$  ) do  
    extrage  $i$  from  $NOW$ ;  $S \leftarrow S \cup \{i\}$ ;  
     $L \leftarrow N_G^+(i)$ ; //conține adiacențele și costurile arcelor care pleacă din  $i$ .  
    for ( $j \in L$ ) do  
      if ( $j \notin NOW \cup NEXT$ ) then  
         $u_j \leftarrow u_i + a_{ij}$ ;  $before(j) \leftarrow i$ ; insert  $j$  into  $NEXT$ ;  
      else if ( $u_j > u_i + a_{ij}$ ) then  
         $u_j \leftarrow u_i + a_{ij}$ ;  $before(j) \leftarrow i$ ;  
  if ( $NEXT \neq \emptyset$ ) then  
    find  $d = \min_{i \in NEXT} u_i$ ; move to  $NOW$  each  $i \in NEXT$  cu  $u_i = d$ ;
```

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Algoritmul Bellman-Ford-Moore

Dacă există un arc $ij \in E$ așa încât $a_{ij} < 0$ atunci algoritmul lui Dijkstra poate eșua (strategia "best first" nu mai funcționează). Presupunând că

$$(I') \quad a(C) \geq 0, \forall C \text{ circuit în } G,$$

vom rezolva sistemul

$$(B) \quad \begin{cases} u_s = 0 \\ u_i = \min_{j \neq i} (u_j + a_{ji}), \forall i \neq s \end{cases}$$

prin aproximări succesive.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul Bellman-Ford-Moore

Definim

$$(BM) \quad u_i^k = \min \{a(P) : P \in \mathcal{P}_{si}, |E(P)| \leq k\}, \forall i \in V, k = \overline{1, n-1}$$

Deoarece lungimea (numărul de arce) ale fiecărui drum din G este cel mult $n - 1$, urmează că dacă vom construi

$$\begin{aligned} \mathbf{u}^1 &= (u_1^1, \dots, u_n^1), \\ \mathbf{u}^2 &= (u_1^2, \dots, u_n^2), \\ &\vdots \\ \mathbf{u}^{n-1} &= (u_1^{n-1}, \dots, u_n^{n-1}), \end{aligned}$$

atunci \mathbf{u}^{n-1} este o soluție a sistemului (B). Deoarece valorile lui \mathbf{u}^1 sunt evidente, atunci, dacă vom indica o regulă de a trece de la \mathbf{u}^k la \mathbf{u}^{k+1} , vom obține următorul algoritm pentru a rezolva (B):

Drumuri de cost minim - Problema P2 pentru costuri reale

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Algoritmul Bellman-Ford-Moore

```
 $u_s^1 \leftarrow 0;$   
for ( $i \in V \setminus \{s\}$ ) do  
   $u_i^1 \leftarrow a_{si};$  //evident (BM) are loc.  
for ( $k = \overline{1, n-2}$ ) do  
  for ( $i = \overline{1, n}$ ) do  
     $u_i^{k+1} \leftarrow \min(u_i^k, \min_{j \neq i} (u_j^k + a_{ji}));$ 
```

Pentru a dovedi corectitudinea acestui algoritm, vom arăta că dacă u^k satisface (BM) atunci u^{k+1} satisface (BM), pentru $k = \overline{1, n-2}$.

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul Bellman-Ford-Moore

$$(B) \quad \begin{cases} u_s = 0 \\ u_i = \min_{j \neq i} (u_j + a_{ji}), \forall i \neq s \end{cases}$$

Pentru $i \in V$, considerăm următoarele mulțimi de drumuri:

$$A = \{P : P \in \mathcal{P}_{si}, \text{length}(P) \leq k + 1\}$$

$$B = \{P : P \in \mathcal{P}_{si}, \text{length}(P) \leq k\}$$

$$C = \{P : P \in \mathcal{P}_{si}, \text{length}(P) = k + 1\}$$

Atunci, $A = B \cup C$, și

$$\min \{a(P) : P \in A\} = \min (\min \{a(P) : P \in B\}, \min \{a(P) : P \in C\})$$

Deoarece u_i^k satisface (BM) avem

$$\min \{a(P) : P \in A\} = \min (u_i^k, \min \{a(P) : P \in C\})$$

Algoritmul Bellman-Ford-Moore

Fie $\min \{a(P) : P \in C\} = a(P^0)$, pentru $P^0 \in C$. Atunci j este nodul dinaintea lui $i \in P^0$ (există un astfel de j deoarece P^0 are cel puțin două arce), atunci $a(P^0) = a(P_{sj}^0) + a_{ji} \geq u_j^k + a_{ji}$ (deoarece P_{sj}^0 are k arce și u^k satisface (BM)). Astfel

$$\min \{a(P) : P \in A\} = \min(u_i^k, \min_{j \neq i}(u_j^k + a_{ji})),$$

adică, valoarea asignată lui u_i^{k+1} în algoritm.

Complexitatea timp este $\mathcal{O}(n^3)$ dacă minimul din al doilea for necesită $\mathcal{O}(n)$.

Drumuri de cost minim pot fi obținute ca și în algoritmul lui Dijkstra, dacă vectorul *before*[], inițializat trivial, este actualizat corespunzător atunci când se determină minimul din cel de-al doilea for.

Remarci 4

Algoritmul Bellman-Ford-Moore

- Putem adăuga algoritmului următorul pas:

if ($\exists i \in V$ așa încât $u_i^{n-1} > \min_{j \neq i} (u_j^{n-1} + a_{ji})$) then
return "există un circuit de cost negativ";

În acest fel se obține un test de $\mathcal{O}(n^3)$ dacă digraful G și funcția de cost a încalcă condiția (I') (altfel, din demonstrația corectitudinii, u_i^{n-1} nu poate fi scăzut). Un circuit de cost negativ poate fi găsit folosind vectorul *before*[].

- Dacă există $k < n - 1$ așa încât $\mathbf{u}^k = \mathbf{u}^{k+1}$, atunci algoritmul poate fi oprit. Folosind această idee, este posibil să implementăm algoritmul în $\mathcal{O}(nm)$, memorând nodurile i pentru care valoarea u_i se modifică în coadă.

P3 Dat $G = (V, E)$ digraf; $a : E \rightarrow \mathbb{R}$.

Determină $P_{ij}^* \in \mathcal{P}_{ij}, \forall i, j \in V$, a. î. $a(P_{ij}^*) = \min \{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$.

- Fie $u_{ij} = \min \{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$. Astfel avem de determinat matricea $U = (u_{ij})_{n \times n}$, când matricea de cost-adiacență A este dată.
- Fiecare drum de cost minim poate fi obținut în $\mathcal{O}(n)$ dacă, în timpul construcției matricei U , întreținem matricea *Before* = $(before_{ij})_{n \times n}$, unde *before*_{ij} = nodul dinaintea lui j pe drumul de cost minim de la i la j din G .
- Dacă perechea (G, a) satisface condiția (I'), putem rezolva P3 aplicând algoritmul Bellman-Ford-Moore pentru $s \in \{1, \dots, n\}$, în $\mathcal{O}(n^4)$. Există și soluții mai eficiente pe care le vom prezenta pe slide-urile următoare.

Iterarea algoritmului lui Dijkstra

Dacă toate costurile sunt nenegative, putem rezolva P3 aplicând algoritmul lui Dijkstra pentru $s \in \{1, \dots, n\}$, în $\mathcal{O}(n^3)$.

Iterarea algoritmului lui Dijkstra este de asemeni posibilă și atunci când avem costuri negative, dacă condiția (I') are loc, după o pre-procesare interesantă.

Fie $\alpha : V \rightarrow \mathbb{R}$ așa încât $\forall ij \in E, \alpha(i) + a_{ij} \geq \alpha(j)$.

Fie $\bar{a} : E \rightarrow \mathbb{R}_+$ dată prin $\bar{a}_{ij} = a_{ij} + \alpha(i) - \alpha(j), \forall ij \in E$.

Avem $\bar{a}_{ij} \geq 0$ și nu este dificil de a vedea că pentru orice $P_{ij} \in \mathcal{P}_{ij}$,

$$(*) \quad \bar{a}(P_{ij}) = a(P_{ij}) + [\alpha(i) - \alpha(j)].$$

Astfel, putem itera algoritmul Dijkstra pentru a determina drumurile de cost minim relativ la \bar{a} .

Drumuri de cost minim - Rezolvarea problemei P3

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Iterarea algoritmului lui Dijkstra

Relația (*) arată că un drum este de cost minim relativ la costul \bar{a} dacă și numai dacă este minim relativ la costul a ($\bar{a}(P_{ij}) - a(P_{ij})$ este o constantă care nu depinde de P). Astfel, avem următorul algoritm:

- 1: find α and construct matrice \bar{A} ;
- 2: solve P3 for \bar{A} , returning \bar{U} and \overline{Before} ;
- 3: find U ($u_{ij} = \bar{u}_{ij} - \alpha(i) + \alpha(j)$);

Pasul 2 necesită $\mathcal{O}(n^3)$ din iterarea algoritmului lui Dijkstra. Pasul 1 poate fi implementat în $\mathcal{O}(n^3)$, alegând un nod $s \in V$ și rezolvând P2 cu algoritmul Bellman-Ford-Moore (care testează de asemenea dacă (I') este îndeplinită).

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Iterarea algoritmului lui Dijkstra

Într-adevăr, dacă $(u_i, i \in V)$ este soluție pentru P2, atunci $(u_i, i \in V)$ satisface sistemul (B), deci $u_j = \min_{i \neq j} (u_i + a_{ij})$, adică, $\forall ij \in E$, avem $u_j \leq u_i + a_{ij}$. Astfel, $a_{ij} + u_i - u_j \geq 0$, $\forall ij \in E$, ceea ce arată că putem lua $\alpha(i) = u_i$, $\forall i \in V$ așa încât condiția (*) să aibă loc.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Algoritmul Floyd - Warshall

Fie

$$u_{ij}^k = \min \{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}, V(P_{ij}) \setminus \{i, j\} \subseteq \{1, 2, \dots, k-1\}\}$$

$$\forall i, j \in V, k = \overline{1, n+1}.$$

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Algoritmul Floyd - Warshall

Evident, $u_{ij}^1 = a_{ij}$, $\forall i, j \in V$ (presupunem că $a_{ii} = 0$, $\forall i \in V$). Mai mult,

$$u_{ij}^{k+1} = \min \{u_{ij}^k, u_{ik}^k + u_{kj}^k\}, \forall i, j \in V, k = \overline{1, n}.$$

Aceasta rezultă prin inducție după k . În pasul inductiv: un drum de cost minim de la i la j fără noduri interne $\geq k+1$ fie nu conține nodul k și costul său este u_{ij}^k , fie conține nodul k și atunci costul său este $u_{ik}^k + u_{kj}^k$ (din principiul de optimalitate al lui Bellman și ipoteza inductivă).

Evident, dacă obținem $u_{ii}^k < 0$, atunci există un circuit de cost negativ C care trece prin nodul i cu $V(C) \setminus \{i\} \subseteq \{1, \dots, k-1\}$.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Rezolvarea problemei P3

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

```
for ( $i = \overline{1, n}$ ) do
  for ( $j = \overline{1, n}$ ) do
     $before(i, j) \leftarrow i$ ;
    if ( $i = j$ ) then
       $a_{ii} \leftarrow 0$ ;  $before(i, i) \leftarrow 0$ ;
  for ( $k = \overline{1, n}$ ) do
    for ( $i = \overline{1, n}$ ) do
      for ( $j = \overline{1, n}$ ) do
        if ( $a_{ij} > a_{ik} + a_{kj}$ ) then
           $a_{ij} \leftarrow a_{ik} + a_{kj}$ ;  $before(i, j) \leftarrow before(k, j)$ ;
        if ( $i = j$  și  $a_{ij} < 0$ ) then
          return "circuit negativ";
```

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Să presupunem că (I') este îndeplinită și în matricea de cost-adiacență elementele diagonale sunt 0. Fie

$$\forall i, j \in V, k = \overline{1, n-1}.$$

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Înmulțirea (rapidă) a matricilor

```
for  $(i, j \in V)$  do
  if  $(i \neq j)$  then
     $u_{ij}^0 \leftarrow \infty$ ;
  else
     $u_{ij}^0 \leftarrow 0$ ;
for  $(k = 0, n - 2)$  do
  for  $(i, j \in V)$  do
     $u_{ij}^{k+1} = \min_{h \in V} (u_{ih}^k + a_{hj})$ ;
```

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Înmulțirea (rapidă) a matricilor

Considerăm următorul “produs de matrici”

$$\forall B, C \in \mathcal{M}_{n \times n}, B \otimes C = P = (p_{ij}), \text{ where } p_{ij} = \min_{k=1, n} (a_{ik} + b_{kj}).$$

Operația \otimes este asociativă și este similară înmulțirii uzuale a matricilor. Putem scrie $U^{m+1} = U^m \otimes A$ și, prin inducție, obținem

$$U^1 = A, U^2 = A^{(2)}, \dots, U^{n-1} = A^{(n-1)},$$

$$\text{where } A^{(k)} = A^{(k-1)} \otimes A \text{ și } A^{(1)} = A.$$

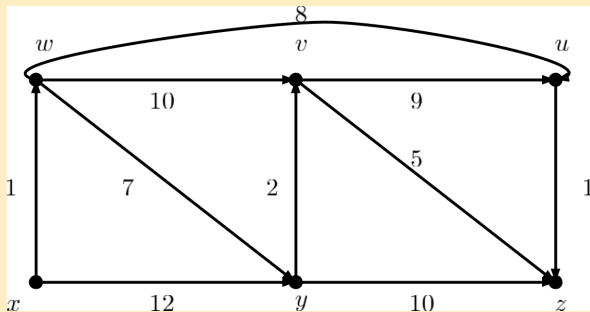
În ipoteza (I') avem: $A^{(2^p)} = A^{(n-1)}$, $\forall p$ cu $2^p \geq n - 1$.

Astfel, calculând succesiv, $A, A^{(2)}, A^{(4)} = A^{(2)} \otimes A^{(2)}, \dots$, obținem algoritmul cu $\mathcal{O}(n^3 \log n)$ complexitate timp pentru rezolvarea problemei P3.

Exerciții pentru seminarul de săptămâna următoare

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Exercițiul 1'. Rulați algoritmul lui Dijkstra pe digraful următor.



C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul de săptămâna următoare

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiul 1. Un graf G este dat funcțional: pentru fiecare nod $v \in V(G)$ putem obține $N_G(v)$ pentru 1\$; alternativ, după o preprocesare care costă T \$ ($T \gg 1$), putem obține, $N_G(v)$ și also $N_G(w)$, pentru toate nodurile $w \in V(G)$. În G se construiește un drum P plecând dintr-un nod arbitrar și alegând un vecin nevizitat încă atâta vreme cât este posibil. După ce drumul este construit putem să îi comparăm costul, $Online(P)$, cu cel mai mic cost posibil, $Offline(P)$. Descrieți o strategie de plată (de accesare a mulțimilor de vecini) așa încât

$$Online(P) \leq \left(2 - \frac{1}{T}\right) \cdot Offline(P).$$

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul de săptămâna următoare

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiul 2. Fie D un digraf și $a : E(D) \rightarrow \mathbb{R}_+$, $b : E(D) \rightarrow \mathbb{R}_+^*$.
Descrieți algoritm eficient pentru determinarea unui circuit C^* of D ,
așa încât

$$\frac{a(C^*)}{b(C^*)} = \min \left\{ \frac{a(C)}{b(C)} : C \text{ circuit în } D \right\}.$$

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercițiul 3. În problema determinării drumurilor de cost minim de
la un nod dat s la toate celelalte noduri dintr-un digraf $G = (V, E)$,
avem o funcție de cost $c : E \rightarrow \{0, 1, \dots, C\}$ unde $C \in \mathbb{N}$ nu depinde de
 $n = |V|$ sau de $m = |E|$. Cum se poate modifica algoritmul lui Dijkstra
pentru a reduce complexitatea timp la $\mathcal{O}(n + m)$?

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiul 4. Fie $D = (V, E)$ un digraf tare conex de ordin n și $a : E \rightarrow \mathbb{R}$ o funcție de cost pe arcele sale. Dacă X este un mers, un drum sau un circuit în D , atunci $a(X)$, costul lui X , este suma costurilor de pe arcele sale, $len(X)$, este lungimea lui X (numărul de arce), și $a_{avg}(X)$, costul mediu al arcelor sale, este $a_{avg}(X) = \frac{a(X)}{len(X)}$.
Fie

$$a_{avg}^* = \min_{C \text{ circuit în } D} a_{avg}(C).$$

Fie $s \in V$, $k \in \mathbb{N}^*$, și $A_k(v)$ costul minim al unui mers de la s la v de lungime k (dacă există un astfel de mers, altfel $A_k(v) = +\infty$).

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercițiul 4. (cont'd)

- a) Dacă D nu conține circuite de cost negativ, dar există un circuit C cu $a(C) = 0$, arătați că există un nod $v \in V$ așa încât

$$A_n(v) = \min \{a(P) : P \text{ este un } sv\text{-drum de la } s \text{ la } v \text{ în } D\}.$$

- b) Arătați că dacă $a_{avg}^* = 0$, atunci

$$(MMC) \quad a_{avg}^* = \min_{v \in V} \max_{0 \leq k \leq n-1} \frac{A_n(v) - A_k(v)}{n - k}.$$

- c) Dacă $a_{avg}^* \neq 0$, transformați funcția a așa încât noua funcție a' să satisfacă ipoteza de la b) și din (MMC) (care are loc pentru a') arătați că (MMC) are loc pentru orice funcție a .

Exercițiul 5. În numeroase aplicații, pentru un digraf dat, $G = (V, E)$ cu $a : E \rightarrow \mathbb{R}_+$, trebuie să răspundem consecvent la o întrebare de tipul următor: Care este drumul de cost minim dintre s și t ? ($s, t \in V$, $s \neq t$). Pentru un digraf foarte mare, G , se propune următorul algoritm Dijkstra modificat (bidirecțional):

- construiește inversul lui G , G' , cu funcția de cost $a' : E(G') \rightarrow \mathbb{R}_+$, dată prin $a'_{ij} = a_{ji}$, $\forall ij \in E(G')$;
- aplică succesiv un pas din algoritmul lui Dijkstra lui G și a (plecând din s) și lui G' și a' (plecând din t);
- când un nod u este introdus în S (mulțimea nodurilor etichetate de algoritmul lui Dijkstra) de amândouă instanțele algoritmului ne oprim;
- returnează drumul de la s la u în G reunit cu inversul drumului de la t la u în G' .

Arătați că această procedură nu este corectă, oferind un exemplu care

Exercițiul 6. Dați un exemplu de digraf care să aibă și costuri negative pe arce și pe care algoritmul lui Dijkstra să eșueze.

Exercițiul 7. Fie G un graf conex. Arătați că

- a) orice două drumuri de lungime maximă ale lui G au intersecție nevidă.
- b) dacă G este un arbore, atunci toate drumurile de lungime maximă din G au intersecția nevidă.

Exercițiul 8. Fie $G = (V, E)$ un graf conex.

- a) Arătați că există o mulțime stabilă, S , așa încât graful parțial bipartit $H = (S, V \setminus S; E')$ să fie conex, unde $E' = E \setminus \binom{V \setminus S}{2}$.
- (b) Arătați că $\alpha(G) \geq \frac{|G| - 1}{\Delta(G)}$.

Exerciții pentru seminarul de săptămâna următoare

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiul 9. Arătați că orice arbore, T , are cel puțin $\Delta(T)$ noduri pendante (frunze).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiul 10. Fie G un graf cu $n \geq 2$ noduri și m muchii.

- a) Arătați că G conține cel puțin două noduri de același grad.
- b) Fie $r(G)$ numărul maxim de noduri având același grad în G . Dacă notăm cu $d_{med} = 2m/n$, demonstrați că

$$r(G) \geq \left\lceil \frac{n}{2d_{med} - 2\delta(G) + 1} \right\rceil.$$

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exercițiul 11. Fie $G = (S, T; E)$ un graf bipartit cu următoarele proprietăți:

- $|S| = n, |T| = m$ ($n, m \in \mathbb{N}^*$);
- $\forall t \in T, |N_G(t)| > k > 0$ (pentru un $k < n$ fixat);
- $\forall t_1, t_2 \in T$, dacă $t_1 \neq t_2$, atunci $|N_G(t_1) \cap N_G(t_2)| = k$;

Arătați că $m \leq n$;

Exercițiul 12. Fie $G = (V, E)$ un digraf; definim o funcție $f_G : \mathcal{P}(V) \rightarrow \mathbb{N}$ prin $f_G(\emptyset) = 0$ și $f_G(S) = |\{v : v \text{ este accesibil din } S\}|$, pentru $\emptyset \neq S \subseteq V$.

(a) Arătați că, pentru orice digraf G , avem

$$f_G(S) + f_G(T) \geq f_G(S \cup T) + f_G(S \cap T), \forall S, T \subseteq V.$$

(b) Demonstrați că (a) este echivalentă cu

$$f_G(X \cup \{v\}) - f_G(X) \geq f_G(Y \cup \{v\}) - f_G(Y), \forall X \subseteq Y \subseteq V, \forall v \in V \setminus Y$$