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.

Algoritmica grafurilor - Cursul 4

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

26 octombrie 2018

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 - 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 *

Cuprins

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 - Solving all-pairs shortest drum problem **P3**
   - Înmulţirea (rapidă) a matricilor
2. Exerciţii 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 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

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

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

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

Lema 1

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

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

Proof: "$\Leftarrow$" Fie ord o ordonare topologică a lui G. Dacă $C = (u_1, u_1u_2, u_2, \ldots, u_k, u_ku_1, u_1)$ este un circuit în G, atunci, din proprietatea funcţiei ord, obţinem contradicţiea

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

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

let $v_0 \in V$;
while $(d^-_G(v_0) \neq 0)$ **do**
  take $u \in V$ aşa încât $uv_0 \in E$;
  $v_0 \leftarrow u$;
return v_0.

Evident, deoarece G nu are circuite şi V este finită, algoritmul se termină şi în nodul returnat, v_0, nu mai intră arce. Digraful $G - v_0$ nu are circuite şi 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 demonstraţia de mai sus obţinem următorul algoritm pentru recunoaşterea unui dag şi construcţia unei ordonări topologice în cazul instanţelor "yes":

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

Output: "yes" dacă G este dag şi 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 $d_G^-[u], \forall u \in V$; let $v \in V$ a. î. $d_G^-[v] = 0$;
$count \leftarrow 0$; $S \leftarrow \{u \in V : d_G^-[u] = 0\}$; // S este o coadă sau o stivă;
while $(S \neq \varnothing)$ **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

P2 Dat $G = (V, E)$ digraf; $a : E \to \mathbb{R}$; $s \in V$.
Determină $P^*_{si} \in \mathcal{P}_{si}, \forall i \in V$, a. î. $a(P^*_{si}) = \min\{a(P_{si}) : P_{si} \in \mathcal{P}$

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

P2 cu $G = (\{1, \ldots, n\}, E)$ dag, cu $ord[i] = i$, $\forall i \in V$, şi $s = 1$.
Condiţia (I) este satisfăcută şi sistemul (B) se rezolvă prin "substituţie".

$u_1 \leftarrow 0$; $before[1] \leftarrow 0$;
for $(i = \overline{2, n})$ **do**
  $u_i \leftarrow \infty$; $before[i] \leftarrow 0$;
  for $(j = \overline{1, i-1})$ **do**
    if $(u_i > u_j + a_{ji})$ **then**
      $u_i \leftarrow u_j + a_{ji}$; $before[i] \leftarrow j$;
// complexitatea timp $\mathcal{O}(n^2)$;

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

P2 Dat $G = (V, E)$ digraf; $a : E \to \mathbb{R}$; $s \in V$.
Determină $P^*_{si} \in \mathcal{P}_{si}, \forall i \in V$, a. î. $a(P^*_{si}) = \min\{a(P_{si}) : P_{si} \in \mathcal{P}$

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

P2 cu $a(e) \geqslant 0, \forall e \in E$.
Condiţia (I) este satisfăcută şi sistemul (B) se poate rezolva cu **algoritmul lui Dijkstra**, care are următorul invariant: $S \subseteq V$ şi

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

Iniţial $S = \{s\}$ şi în fiecare dintre cei $n-1$ paşi un nod nou este adăugat la S, obţinând $S = V$. 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 -

Algoritmul lui Dijkstra

$S \leftarrow \{s\}$; $before[s] \leftarrow 0$;
for $(i \in V \setminus \{s\})$ **do**
  $u_i \leftarrow a_{si}$; $before[i] \leftarrow s$; // (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**
      $u_j \leftarrow u_{j^*} + a_{j^*j}$; $before[j] \leftarrow j^*$;

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 *

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

Demonstraţia corectitudinii algoritmului lui Dijkstra

Deoarece (D) are loc după pasul de iniţializare, trebuie să demonstrăm că dacă (D) are loc înaintea iteraţiei while curente, atunci (D) are loc şi înaintea iteraţiei următoare.

Fie $S \subseteq V$ şi $u_1, \ldots u_n$ satisfăcând (D) înaintea iteraţiei 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 $\exists P^1_{sj^*} \in \mathcal{P}_{sj^*}$ aşa încât $a(P^1_{sj^*}) < u_{j^*}$. Deoarece S şi 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 $V(P^1_{sj^*}) \setminus S \neq \{j^*\}$; fie k primul nod de pe $P^1_{sj^*}$ (începând cu s) aşa încât $k \notin S$.

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

Demonstraţia corectitudinii algoritmului lui Dijkstra (cont.)

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

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

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

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

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

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

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

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

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

Am obţinut că singura posibilitate de a avea $\alpha_j < u_j$ este când $a(P^1_{j^*j}) = a_{j^*j}$, 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**).

□

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

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

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

- 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

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 $\mathcal{O}((n-1)+(n-2)+\cdots+1) = \mathcal{O}(n^2)$.
- Dacă valorile lui u_j pentru $j \in V \setminus S$ sunt ţinute într-o coadă cu priorităţi (e.g. heap) atunci extragerea fiecărui minim necesită $\mathcal{O}(1)$, dar timpul necsar execuţiei 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 întreţinerea heap-ului (Johnson,1977).
- Cea mai bună implementare se obţine folosind o grămadă (heap) Fibonacci cu o complexitate timp de $\mathcal{O}(m + n \log n)$ (Fredman & Tarjan, 1984).

- 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 1

- Pentru a rezolva problema **P1** folosind algoritmul lui Dijkstra, adăgăm un test de oprire a execuţiei când nodul t este introdus în S. în cazul cel mai nefavorabil complexitatea rămâne aceeaşi.
- O euristica interesantă care dirijează căutarea spre t se obţine cu ajutorul unui **estimator consistent**, i. e. o funcţie $g : V \to \mathbb{R}_+$ care satisface condiţiile:

  (i) $\forall i \in V,\ u_i + g(i) \leqslant \min\{a(P_{st}) : P_{st} \in \mathcal{P}_{st} \text{ şi } i \in V(P_{st})\}$
  (ii) $\forall ij \in E, g(i) \leqslant a_{ij} + g(j)$.

- 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 mulţime de puncte dintr-un spaţiu Euclidean, atunci, luând $g(i) =$ distanţa Euclideană de la i la t, obţinem un estimator consistent dacă sunt satisfăcute şi condiţiile 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 demonstraţiei 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 *

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.

Remarci 3

În descrierea algoritmului lui Dijkstra se foloseşte matricea de cost-adiacenţă. Pentru digrafuri foarte mari sau pentru digrafuri date printr-o funcţie care returnează lista de adiacenţă a unui nod dat, implementările folosind matricea de cost-adiacenţă sunt fie ineficiente fie imposibile.
O implementare care evită aceste lipsuri este dată pe slide-ul următor (Partition Shortest Path algoritm due to Glover, Klingman şi Philips (1985)).

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

Partition Shortest Path (PSP) algoritm

$u_s \leftarrow 0$; $before(s) \leftarrow 0$; $S \leftarrow \varnothing$; $NOW \leftarrow \{s\}$; $NEXT \leftarrow \varnothing$;
while ($NOW \cup NEXT \neq \varnothing$) **do**
  while ($NOW \neq \varnothing$) **do**
    extrge i from NOW; $S \leftarrow S \cup \{i\}$;
    $L \leftarrow N_G^+(i)$; //conţine adiacenţele şi 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 ($NEXT \neq \varnothing$) **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 -

Algoritmul Bellman-Ford-Moore

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

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

vom rezolva sistemul

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

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 *

G. Groiteru - Graph Algorithms * G. Groiteru - Graph Algorithms * G. Groiteru - Graph Algorithms

Algoritmul Bellman-Ford-Moore

Definim

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

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

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

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

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 $(m = \overline{1, n-2})$ **do**
  for $(i = \overline{1, n})$ **do**
    $u_i^{m+1} \leftarrow \min\left(u_i^m, \min_{j \neq i}\left(u_j^m + a_{ji}\right)\right);$

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

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

Algoritmul Bellman-Ford-Moore

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

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

$$A = \{P \ : \ P \in \mathcal{P}_{si}, length(P) \leqslant m+1\}$$
$$B = \{P \ : \ P \in \mathcal{P}_{si}, length(P) \leqslant m\}$$
$$C = \{P \ : \ P \in \mathcal{P}_{si}, length(P) = m+1\}$$

Atunci, $A = B \cup C$, şi

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

Deoarece u_i^m satisface (BM) avem

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

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

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 puţin două arce), atunci $a(P^0) = a(P^0_{sj}) + a_{ji} \geqslant u_j^m + a_{ji}$ (deoarece P^0_{sj} are m arce şi $\mathbf{u}^m$ satisface (BM). Astfel

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

adică, valoarea asignată lui u_i^{m+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 obţinute ca şi în algoritmul lui Dijkstra, dacă vectorul *before*[], iniţializat trivial, este actualizat corespunzător atunci când se determină minimul din cel de-al doilea for.

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

Remarci 4

Algoritmul Bellman-Ford-Moore

- Putem adăuga algoritmului următorul pas:

  if $(\exists i \in V$ aşa 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 obţine un test de $\mathcal{O}(n^3)$ dacă digraful G şi funcţia de cost a încalcă condiţia (I') (altfel, din demonstraţia corectitudinii, u_i^{n-1} nu poate fi scăzut). Un circuit de cost negativ poate fi găsit folosind vectorul $before[]$.
- Dacă există $k < n-1$ aşa 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; $a : E \to \mathbb{R}$.
Determină $P^*_{ij} \in \mathcal{P}_{ij}, \forall i, j \in V$, a. î. $a(P^*_{ij}) = \min\{a(P_{ij}) : P_{ij} \in$

- 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-adiacenţă A este dată.
- Fiecare drum de cost minim poate fi obţinut în $\mathcal{O}(n)$ dacă, în timpul construcţiei matricei U, întreţinem 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 condiţia (I'), putem rezolva **P3** aplicând algoritmul Bellman-Ford-Moore pentru $s \in \{1, \ldots, n\}$, în $\mathcal{O}(n^4)$. Există şi soluţii mai eficiente pe care le vom prezenta pe slide-urile următoare.

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

Iterarea algoritmului lui Dijkstra

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

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

Fie $\alpha : V \to \mathbb{R}$ aşa încât $\forall ij \in E$, $\alpha(i) + a_{ij} \geqslant \alpha(j)$.

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

Avem $\overline{a}_{ij} \geqslant 0$ şi nu este dificil de a vedea că pentru orice $P_{ij} \in \mathcal{P}_{ij}$,

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

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

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

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

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

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 $s \in V$ şi rezolvând **P2** cu algoritmul Bellman-Ford-Moore (care testează de asemeni 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 -

Iterarea algoritmului lui Dijkstra

Într-adevăr, dacă $(u_i, i \in V)$ este soluţie pentru **P2**, atunci $(u_i, i \in V)$ satisface sistemul (B), deci $u_j = \min\limits_{i \neq j} (u_i + a_{ij})$, adică, $\forall ij \in E$, avem $u_j \leqslant u_i + a_{ij}$. Astfel, $a_{ij} + u_i - u_j \geqslant 0$, $\forall ij \in E$, ceea ce arată că putem lua $\alpha(i) = u_i$, $\forall i \in V$ aşa încât condiţia (*) să aibă loc.

Algoritmul Floyd - Warshall

Fie

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

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

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 -

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,

$$u_{ij}^{m+1} = \min\{u_{ij}^m, u_{im}^m + u_{mj}^m\}, \forall i,j \in V, m = \overline{1,n}.$$

Aceasta rezultă prin inducţie după m. În pasul inductiv: un drum de cost minim de la i la j fără noduri interne $\geqslant m+1$ fie nu conţine nodul m şi costul său este u_{ij}^m, fie conţine nodul m şi atunci costul său este $u_{im}^m + u_{mj}^m$ (din principiul de optimalitate al lui Bellman şi ipoteza inductivă).

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

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

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 $(m = \overline{1,n})$ **do**
  for $(i = \overline{1,n})$ **do**
    for $(j = \overline{1,n})$ **do**
      if $(a_{ij} > a_{im} + a_{mj})$ **then**
        $a_{ij} \leftarrow a_{im} + a_{mj}$; $before(i,j) \leftarrow before(m,j)$;
        if $(i = j$ şi $a_{ij} < 0)$ **then**
          return "circuit negativ";

- 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

Această metodă pentru rezolvarea problemei **P3** este cunoscută drept **algoritmul Floyd - Warshal**:
Evident, complexitatea timp a acestui algoritm este $\mathcal{O}(n^3)$.

Remarcă

Dacă ştim că digraful nu are circuite de cost negative, atunci dacă elementele diagonale ale lui A sunt iniţializate cu ∞, valoarea finală a fiecărui element diagonal este costul minim al unui circuit care trece prin nodul corespunzător.

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

Înmulţirea (rapidă) a matricilor

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

$$u_{ij}^m = \min\{a(P_{ij}) \ : \ P_{ij} \in \mathcal{P}_{ij}, P_{ij} \text{ are cel mult } m \text{ arce}\}$$

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

Notăm cu $U^m = (u_{ij}^m)_{1 \leqslant i,j \leqslant n}$ pentru $m \in \{0, 1, 2, \ldots, n-1\}$, unde U^0 are toate elementele ∞, mai puţin cele de pe diagonală care sunt 0. Atunci, iterarea algoritmului Bellman-Ford-Moore poate fi descrisă într-o formă matriceală:

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

Înmulţirea (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 $(m = \overline{0, n-2}\ )$ **do**
  for $(i, j \in V)$ **do**
    $u_{ij}^{m+1} = \min\limits_{k \in V}(u_{ik}^{m} + a_{kj});$

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

Înmulţirea (rapidă) a matricilor

Considerăm următorul "produs de matrici"

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

Operaţia $\otimes$ este asociativă şi este similară înmulţirii uzuale a matricilor. Putem scrie $U^{m+1} = U^m \otimes A$ şi, prin inducţie, obţinem

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

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

În ipoteza (I') avem: $A^{(2^k)} = A^{(n-1)}$, $\forall k$ cu $2^k \geqslant n-1$.
Astfel, calculând succesiv, $A, A^{(2)}, A^{(4)} = A^{(2)} \otimes A^{(2)}, \ldots$,obţinem algoritm cu $\mathcal{O}(n^3 \log n)$ complexitate timp pentru 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 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 *

Remarcă 1

Dacă operaţia $\otimes$ este implementată cu algoritmi mai rapizi, n^3 de mai sus devine $n^{\log_2 7} = n^{2.81}$ (Strassen 1969) sau $n^{2.3728639}$ (Cooppersmith & Winograd 1987, Le Gall 2014).

Algorithms * C. Croitoru - Graph Algorithms *

Exerciţii pentru seminarul de săptămâna următare

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 -

Exerciţiul 1. Un graf G este dat funcţional: pentru fiecare nod $v \in V(G)$ putem obţine $N_G(v)$ pentru 1\$; alternativ, după o preprocesare care costă T\$ ($T >> 1$), putem obţine, $N_G(v)$ şi also $N_G(w)$, pentru toate nodurile $w \in V(G)$. În G se construieşte un drum P plecând dintr-un nod arbitrar şi 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, $Offline(P)$. Descrieţi o strategie de plată (de accesare a mulţimilor de vecini) aşa încât

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

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 *

Exerciţii pentru seminarul de săptămâna următare

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 -

Exerciţiul 2. Fie D un digraf şi $a : E(D) \to \mathbb{R}_+$, $b : E(D) \to \mathbb{R}_+^*$. Descrieţi algoritm eficient pentru determinarea unui circuit C^* of D, aşa 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 *

Exerciţiul 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 funcţie de cost $c : E \to \{0, 1, \ldots, C\}$ 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 *

Exerciţii pentru seminarul de săptămâna următare

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 -

Exerciţiul 4. Fie $D = (V, E)$ un digraf tare conex de ordin n şi $a : E \to \mathbb{R}$ o funcţie 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 lungime lui X (numărul de arce), şi $a_{avg}(X)$, 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 $s \in V$, $k \in \mathbb{N}^*$, şi $A_k(v)$ costul minim al unui mers de la s la v (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 *

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

Exerciţiul 4. (cont'd)

a) Dacă D nu conţine circuite de cost negativ, dar există un circuit C cu $A(C) = 0$, arătaţi că există un nod $v \in V$ aşa încât

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

b) Arătaţi că dacă $a^*_{avg} = 0$, atunci

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

c) Dacă $a^*_{avg} \neq 0$, transformaţi funcţia a aşa încât noua funcţie a' să satisfacă ipoteza de la **b)** şi din (MMC) (care are loc pentru a') arătaţi că (MMC) are loc pentru orice funcţie a.

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

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

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

Exerciţii pentru seminarul de săptămâna următare

Exerciţiul 6. Daţi un exemplu de digraf care să aibă şi costuri negative pe arce şi pe care algoritmul lui Dijkstra să eşueze.

Exerciţiul 7. Fie G un graf conex. Arătaţi că

a) orice două drumuri de lungime maximă ale lui G au intersecţie nevidă.

b) dacă G este un arbore, atunci toate drumurile de lungime maximă din G au intersecţia nevidă.

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

a) Arătaţi că există o mulţime stabilă, S, aşa încât graful parţial bipartit $H = (S, V \setminus S; E')$ să fie conex, unde $E' = E \setminus \binom{V \setminus S}{2}$.

(b) Arătaţi că $\alpha(G) \geqslant \dfrac{|G| - 1}{\Delta(G)}$.

Exerciţii pentru seminarul de săptămâna următare

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 -

Exerciţiul 9. Arătaţi că orice arbore, T, are cel puţin $\Delta(T)$ noduri pendante (frunze).

Exerciţiul 10. Fie G un graf cu $n \geqslant 2$ noduri şi m muchii.

a) Arătaţi că G conţine cel puţin două noduri de acelaşi grad.

b) Fie $r(G)$ numărul maxim de noduri având acelaşi grad în G. Dacă notăm cu $d_{med} = 2m/n$, demonstraţi că

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

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 *

Exerciţii pentru seminarul de săptămâna următare

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

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

Arătaţi că $m \leqslant n$;

Exerciţiul 12. Fie $G = (V, E)$ un digraf; definim o funcţie $f_G : \mathcal{P}(V) \to \mathbb{N}$ prin $f_G(\varnothing) = 0$ şi $f_G(S) = |\{v : v \text{ este accesibil din } S\}|$, pentru $\varnothing \neq S \subseteq V$.

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

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

(b) Demonstraţi că (a) este echivalentă cu

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