

Algoritmica Grafurilor - Cursul 3

19 ottobre 2018

- 1 **Vocabularul teoriei grafurilor**
 - Matrici asociate
- 2 **Structuri de date**
- 3 **Probleme de drumuri în (di)grafuri**
 - Traversarea (di)grafurilor
 - Drumuri de cost minim
- 4 **Exerciții pentru seminarul de săptămâna viitoare**

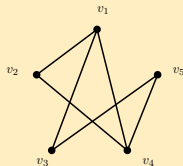
Matrici asociate - Matricea de adiacență

Fie G un graf cu $V(G) = \{v_1, \dots, v_n\}$. **Matricea de adiacență** a lui G este matricea $A = (a_{ij})_{1 \leq i, j \leq n} \in \mathcal{M}_{n \times n}(\{0, 1\})$, unde

$$a_{ij} = \begin{cases} 1, & \text{dacă } v_i \text{ și } v_j \text{ adiacente} \\ 0, & \text{altfel} \end{cases}.$$

Exemplu

Un graf și matricea sa de adiacență.



$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Matrici asociate - Matricea de incidență

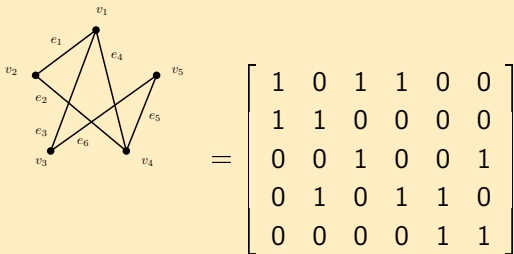
Fie G un graf cu $V(G) = \{v_1, \dots, v_n\}$ și $E(G) = \{e_1, \dots, e_m\}$.

Matricea de incidență a lui G este matricea $B = (b_{ij})_{1 \leq i, j \leq n} \in \mathcal{M}_{n \times m}(\{0, 1\})$, unde

$$b_{ij} = \begin{cases} 1, & \text{dacă } e_j \text{ este incidentă cu } v_i \\ 0, & \text{altfel} \end{cases}.$$

Exemplu

Un graf și matricea sa de incidență.



Pentru digrafuri, matrici similare pot fi definite cu elemente din $\{-1, 0, 1\}$ pentru a marca direcția arcelor.

Graph Algorithms " C. Croitoru - Graph Algorithms " C. Croitoru - Graph Algorithms " C. Croitoru

Valorile proprii, vectorii proprii și polinomul caracteristic ale matricei de adiacență sunt numite **valorile proprii**, **vectorii proprii**, și, respectiv, **polinomul caracteristic ale grafului** corespunzător. Acestea sunt obiectele de studiu ale **teoriei spectrale a grafurilor**.

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

Structuri de date pentru matricea de adiacență

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

Fie $G = (V, E)$ un (di)graf cu $V = \{1, 2, \dots, n\}$.

- Dacă $A = (a_{ij})_{1 \leq i, j \leq n}$ este matricea de adiacență a lui G atunci, reprezentând-o ca un tablou 2-dimensional, avem nevoie de $\mathcal{O}(n^2)$ timp pentru inițializare (în funcție de limbajul de programare).
- Astfel, orice algoritm care reprezintă G cu matrice de adiacență are complexitatea timp (și spațiu) $\Omega(n^2)$.
- Testarea dacă două noduri fixate sunt adiacente se face în $\mathcal{O}(1)$, dar parcurgerea întregii mulțimi de vecini $N_G(u)$ (sau $N_{G+(u)}$), pentru un anume nod $u \in V$, necesită $\Omega(n)$ - prea mult pentru grafuri rare de ordin mare.

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

Structuri de date pentru listele de adiacență

Fie $G = (V, E)$ un (di)graf cu $V = \{1, 2, \dots, n\}$ și $|E| = e$.

- Orice nod $u \in V$ are o listă, $A(u)$, a vecinilor săi din G :
 - ▶ când G este graf $A(u) = N_G(u)$;
 - ▶ dacă G este digraf, atunci $A(u) = N_G^+(u) = \{v \in V : uv \in E\}$;
- Dacă G este graf, atunci fiecare muchie $uv \in E$ generează două elemente în listele de adiacență: unul în $A(u)$ și unul în $A(v)$; spațiul necesar este $\mathcal{O}(n + 2e)$.
- Dacă G este digraf, atunci spațiul necesar este $\mathcal{O}(n + e)$.
- Listele de adiacență pot fi implementate folosind liste înlanțuite sau tablouri.
- Testarea dacă un nod u este adiacent cu un altul v în G necesită $\Omega(d_G(u))$ timp, dar parcurgerea întregii mulțimi de vecini $N_G(u)$ (sau $N_G^+(u)$), pentru un nod $u \in V$, se poate face în $\Omega(d_G(u))$ (și nu în $\mathcal{O}(n)$ ca în cazul matricei de adiacență).

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

Traversarea (sistematică a) grafurilor (graph search, graph traversal) este o paradigmă algoritmică care specifică o metodă sistematică de a parcurge mulțimea nodurilor care pot fi accesate (atinse) pe drumuri plecând dintr-un nod fixat al (di)grafului.

Dat un (di)graf $G = (\{1, \dots, n\}, E)$ și $s \in V(G)$

generează "eficient" mulțimea

$S = \{u \in V(G) : \text{există un drum de la } s \text{ la } u \text{ în } G\}.$

G va fi reprezentat cu liste de adiacență, deoarece, în timpul procesului de traversare, trebuie manipulată în mod eficient mulțimea de vecini a nodului curent.

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

Probleme de drumuri - Breadth-First Search (BFS)

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

```
for  $v \in V$  do
     $label(u) \leftarrow -1$ ;  $parent(u) \leftarrow -1$ ;
 $label(s) \leftarrow 0$ ;  $parent(s) \leftarrow 0$ ;
create queue  $Q$  containing  $s$ ;
while  $Q \neq \emptyset$  do
     $u \leftarrow pop(Q)$ ;
    for  $v \in A(u)$  do
        if  $label(v) < 0$  then
             $label(v) \leftarrow label(u) + 1$ ;
             $parent(v) \leftarrow u$ ;  $push(Q, v)$ ;
```

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

Proprietăți ale BFS. Nu este dificil de arătat că:

- $S = \{u \in V : \text{label}(u) \geq 0\}$;
- $\forall u \in V, \text{label}(u) = d_G(s, u)$ (distanța în G de la s la u);
- Variabila **parent** definește **arborele bfs** asociat parcurgerii din s : dacă G este graf atunci arborele bfs este un arbore parțial al componentei conexe conținând s ; dacă G este digraf atunci arborele bfs este o **arborescență** (arbore cu rădăcină, orientat - arcele sunt îndreptate către exteriorul rădăcinii s).
- Complexitatea timp a $BFS(s)$ este $\mathcal{O}(n_S + m_S)$, unde $n_S = |S| \leq |V| = n$, și $m_S = |E([S]_G)| \leq |E|$ (se observă că fiecare vecin din lista de adiacență a unui nod din S este accesat o singură dată).

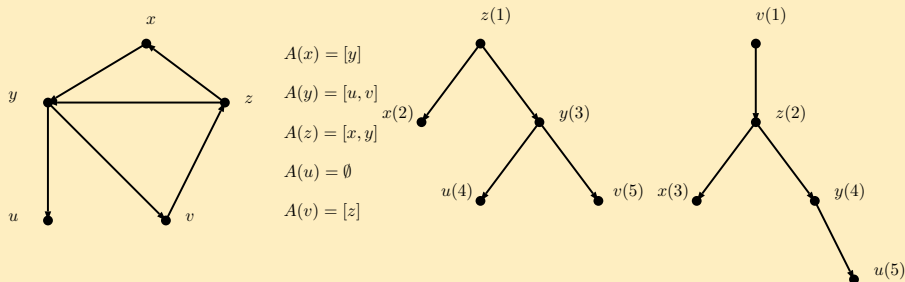
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Probleme de drumuri - Breadth-First Search (BFS)

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

Exemplu

Două parcurgeri BFS ale unui aceluiași graf (începând din z și din v):



(Am marcat în paranteze ordinea de vizitare.)

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

Probleme de drumuri - Depth-First Search (DFS)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph 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  $v \in V$  do
     $label(u) \leftarrow -1$ ;  $parent(u) \leftarrow -1$ ;
 $label(s) \leftarrow 0$ ;  $parent(s) \leftarrow 0$ ;
create stack  $S$  conținând  $s$ ;  $n_S \leftarrow 0$ ;
while  $S \neq \emptyset$  do
     $u \leftarrow top(S)$ ;
    if  $((v \leftarrow next[A(u)]) \neq NULL)$  then
        if  $label(v) < 0$  then
             $n_S ++$ ;  $label(v) \leftarrow n_S$ ;
             $parent(v) \leftarrow u$ ;  $push(S, v)$ ;
        else
             $delete(S, u)$ ;
```

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

Proprietăți ale DFS. Nu este dificil de arătat că:

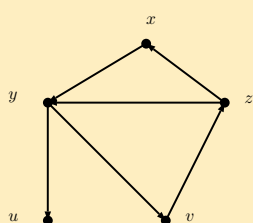
- $\{u \in V : \text{label}(u) \geq 0\}$ este exact mulțimea S de noduri accesibile pe drumuri din s ;
- $\forall u \in V, \text{label}(u) = \text{momentul vizitării lui } u$ (s este vizitat la momentul 0);
- Variabila **parent** definește **arborele dfs** asociat parcurgerii din s
- Complexitatea timp a $DFS(s)$ este $\mathcal{O}(n_S + m_S)$, unde $n_S = |S| \leq |V| = n$, și $m_S = |E([S]_G)| \leq |E|$ (se observă că fiecare vecin din lista de adiacență a unui nod din S este accesat o singură dată).

Probleme de drumuri - Depth-First Search (DFS)

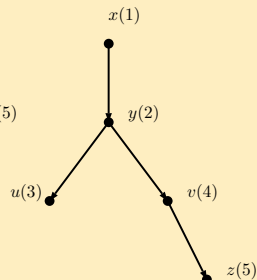
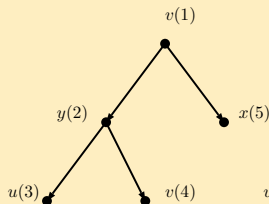
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

Exemplu

Două parcurgeri DFS ale unui același graf (începând din v și din x):



$A(x) = [y]$
 $A(y) = [u, v]$
 $A(z) = [x, y]$
 $A(u) = \emptyset$
 $A(v) = [z]$



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

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

Fie $G = (V, E)$ un digraf, cu $V = \{1, \dots, n\}$.

- Fiecare arc (muchie orientată) $e \in E$ are asociat un cost $a(e) \in \mathbb{R}$ (pondere, lungime etc).
- Dacă G este reprezentat cu liste de adiacență, atunci $a(ij)$ este un câmp al elementului din lista de adiacență a lui i (reprezentând un arc ij).
- Pentru ușurința notației vom folosi reprezentarea lui G cu o matrice de cost-adiacență $A = (a_{ij})_{1 \leq i, j \leq n}$, unde

$$a_{ij} = \begin{cases} a(ij), & \text{dacă } ij \in E \\ \infty, & \text{altfel} \end{cases}.$$

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

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

- Aici, ∞ reprezintă un număr real foarte mare relativ la costurile muchiilor (e.g., $\infty > n \cdot \max_{ij \in E} a(ij)$) și presupunem că $\infty \pm a = \infty$, $\infty + \infty = \infty$.
- Este de asemenea posibil să folosim ∞ ca un acces nereușit la structura de date utilizată pentru reprezentarea matricei A .
- Pentru $i, j \in V$, mulțimea tuturor drumurilor din G de la i la j este notată cu \mathcal{P}_{ij} :

$$\mathcal{P}_{ij} = \{P : P \text{ este un drum de la } i \text{ la } j\}.$$

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

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

- Dacă $P_{ij} \in \mathcal{P}_{ij}$, $P : (i =) v_0, v_0 v_1, v_1, \dots, v_{r-1}, v_{r-1} v_r, v_r (= j)$, atunci

$$V(P_{ij}) = \{v_0, v_1, \dots, v_r\}, E(P_{ij}) = \{v_0 v_1, \dots, v_{r-1} v_r\}.$$

- Costul lui $P_{ij} \in \mathcal{P}_{ij}$ este

$$a(P_{ij}) = 0 + \sum_{uv \in E(P_{ij})} a_{uv}.$$

- În particular $a(P_{ii}) = 0$.

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

Principalele probleme de drum de cost minim

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

Drumul de cost minim între două noduri.

P1 Date $G = (V, E)$ digraf; $a : E \rightarrow \mathbb{R}$; $s, t \in V, s \neq t$.

Găsiți $P_{st}^* \in \mathcal{P}_{st}$, așa încât $a(P_{st}^*) = \min \{a(P_{st}) : P_{st} \in \mathcal{P}_{st}\}$.

Drumurile de cost minim dintre un nod și toate celelalte noduri

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

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

Toate drumurile de cost minim.

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

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

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

Remarci 1

- Reprezentarea cu matrice de cost-adiacență a perechii (G, a) implică $P_{ij} \neq \emptyset, \forall i, j \in V(G)$: dacă $a(P_{ij}) < \infty$, atunci P_{ij} este un drum real în G , iar dacă $a(P_{ij}) = \infty$, atunci P_{ij} nu este un drum în G dar este un drum în digraful complet simetric obținut din G prin adăugarea tuturor arcelor lipsă (cu costuri ∞).
- Urmează că toate mulțimile din care se determină un element (drum) de cost minim în problemele **P1** - **P3** sunt nevide și finite și **drumurile minime cerute sunt bine definite**.
- Algoritmii pentru rezolvarea problemei **P1** se pot obține din cei pentru rezolvarea problemei **P2** prin adăugarea unei condiții (evidente) de oprire.
- Problema **P3** poate fi rezolvată prin iterarea oricărui algoritm pentru problema **P2**. Vom vedea că sunt și soluții mai eficiente.

1. Rețele de comunicație (Communication Networks). Digraful $G = (V, E)$ reprezintă o rețea de comunicații între nodurile din V , iar E modelează mulțimea **legăturilor orientate** dintre noduri.

- Dacă $a(e) \geq 0$ ($\forall e \in E$) reprezintă lungimea conexiunii directe dintre extremitățile lui e , atunci problemele **P1 - P3** sunt **probleme de drumuri de cost minim** naturale.
- Dacă $a(e) \geq 0$ ($\forall e \in E$) reprezintă timpul necesar pentru parcurgerea conexiunii directe dintre extremitățile lui e , atunci problemele **P1 - P3** **probleme pentru determinarea drumurilor celor mai rapide**.
- Dacă $a(e) \in (0, 1]$ ($\forall e \in E$) reprezintă probabilitatea funcționării corecte a conexiunii directe dintre extremitățile lui e și dacă presupunem că muchiile funcționează **independent** una de cealaltă, atunci problemele **P1 - P3** sunt **probleme pentru determinarea drumurilor celor mai sigure**:

Dacă $P_{ij} \in \mathcal{P}_{ij}$ pentru o pereche $i, j \in V$, atunci **probabilitatea** ca acest drum să funcționeze corect este (datorită independenței)

$$Prob(P_{ij}) = \prod_{e \in E(P_{ij})} a(e).$$

Luând $a'(e) = -\log a(e)$,

$$\log Prob(P_{ij}) = \log \left(\prod_{e \in E(P_{ij})} a(e) \right) = - \sum_{e \in E(P_{ij})} a'(e).$$

Funcția \log fiind monotonă, problemele P1 - P3 cu costurile a' , oferă **drumurile cele mai fiabile** în rețele de comunicare.

2. Rețele PERT - Metoda drumului critic (Critical path method - CPM). PERT (Pert Evaluation and Review Technique) este o metodă de a analiza (îndeosebi) îndeplinirea fiecărei sarcini dintr-un proiect mai complex.

- Fie $P = \{A_1, A_2, \dots, A_n\}$ activitățile atomice ale unui mare proiect P (n este foarte mare). $(P, <)$ este o mulțime parțial ordonată, unde $A_i < A_j$ dacă $i \neq j$ și activitatea A_i poate începe înainte ca activitatea A_j să se fi terminat.
- Pentru fiecare activitate A_i , timpul necesar finalizării t_i este cunoscut (sau doar estimat).

Găsiți o planificare a activităților acestui proiect care să minimizeze durata totală până la finalizare.

Asociem acestei problem un digraf aciclic astfel:

- pentru fiecare activitate A_p ($p \in \{1, \dots, n\}$) adăugăm un arc $i_p j_p$ de cost $a(i_p j_p) = t_p$;
- nodul i_p corespunde startului activității A_p iar nodul j_p este asociat finalizării ei;
- dacă activitatea A_k poate porni imediat după activitatea A_p adăugăm arcul $j_p i_k$ (o activitate fictivă - dummy activity).

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

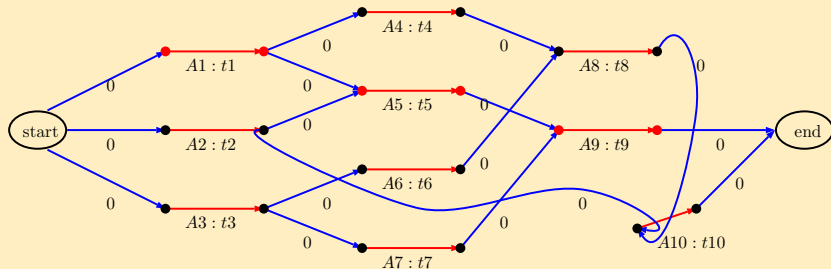
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

- Construcția digrafului este încheiată după adăugarea unui nod s corespunzând momentului inițial al proiectului, legat prin arce si_p pentru fiecare activitate A_p în care nu intră vreun arc și a unui nod t corespunzând finalizării proiectului, legat prin arce j_pt pentru fiecare activitate A_p din care nu iese vreun arc.
- În digraful obținut **costul maxim al unui drum de la s la t este egal cu timpul minim necesar îndeplinirii în întregime a proiectului.**
- Un drum de cost este numit a **drum critic** deoarece orice întârziere a unei activități de pe acest drum implică o întârziere a întregului proiect.

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 - Aplicații

Exemplu



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

3. Problema rucsacului (0 – 1). Avem un rucsac de dimensiune $b \in \mathbb{N}$, și n obiecte de dimensiuni $a_1, \dots, a_n \in \mathbb{N}$. Se cunoaște și profitul $p_i \in \mathbb{N}$ al adăugării obiectului i ($i \in \{1, \dots, n\}$) în rucsac. Se cere **să se găsească o completare a rucsacului care să maximizeze profitul total.**

Fie X_i , pentru $i \in \{1, \dots, n\}$, o variabilă booleană cu semnificația $x_i = 1$ dacă și numai dacă obiectul i este pus în rucsac. Problema rucsacului poate fi descrisă astfel

$$\max \left\{ \sum_{i=1}^n p_i x_i : \sum_{i=1}^n a_i x_i \leq b, x_i \in \{0, 1\}, \forall i = \overline{1, n} \right\}.$$

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

- Fie $G = (V, E)$ un digraf cu $V = \{s\} \cup V_1 \cup \dots \cup V_n \cup \{t\}$, unde $V_i = \{i^0, i^1, \dots, i^b\}$ este asociat obiectului i , $i = \overline{1, n}$.
- Arcele lui G și costurile sunt:
 - ▶ $s1^0$ și $s1^{a_1}$ cu $a(s1^0) = 0$, $a(s1^{a_1}) = p_1$ (obiectul 1 este adăugat rucsacului cu profitul p_1 și nivelul de umplere a_1 , sau nu este adăugat, cu profitul și nivelul de umplere 0).
 - ▶ $(i-1)^j i^j$ cu $a((i-1)^j i^j) = 0$, $\forall i = \overline{2, n}$, $\forall j = \overline{0, b}$ (obiectul i nu este adăugat rucsacului: după competarea cu primele $i-1$ obiecte și nivelul de umplere j se trece la umplerea cu primele i obiecte, fără obiectul i ; nivelul de umplere rămâne neschimbat j iar profitul adăugat este 0).
 - ▶ Dacă $j - a_i \geq 0$, atunci avem și arcul $(i-1)^{j-a_i} i^j$ cu $a((i-1)^{j-a_i} i^j) = p_i$ (se poate ajunge la nivelul de umplere j prin adăugarea obiectului i la o umplere cu primele $i-1$ obiecte, cu nivelul de umplere $j - a_i$).

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

► $n^j t$ cu $a(n^j t) = 0$, $\forall j = \overline{0, b}$.

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

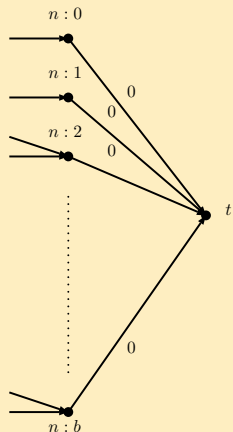
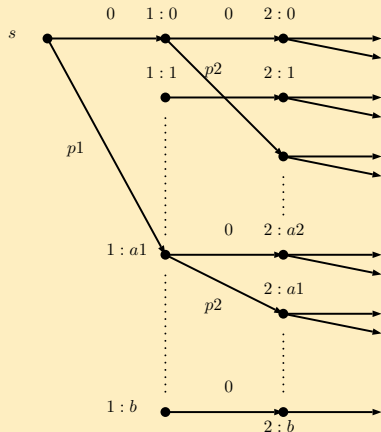
Remarci 2

- Fiecare drum de la s la t în G corespunde unei submulțimi de obiecte cu nivelul de umplere $\leq b$ și cu profitul total egal cu costul drumului. Deoarece, reciproc, fiecărei umpleri a rucsacului îi corespunde un drum de la s la t în G , urmează că problema rucsacului poate fi rezolvată prin determinarea unui drum de cost maxim în digraficul aciclic G .
- Descrierea statică dată mai sus pentru G poate fi transformată într-una procedurală, folosind programarea dinamică. Problema aceasta este NP-hard (ordinul lui G poate fi exponențial în dimensiunea problemei).

Drumuri de cost minim - Aplicații

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

Exemplu



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

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

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

Teorema 1

Fie G un digraf, $s \in V(G) = \{1, \dots, n\}$ și $a : E(G) \rightarrow \mathbb{R}$, a. î.

(I) $a(C) > 0$, pentru toate circuitele C din G .

Atunci (u_1, \dots, u_n) este o soluție a sistemului de ecuații

$$(B) \quad \begin{cases} u_s = 0 \\ u_i = \min_{j \neq i} (u_j + a_{ij}) \end{cases} \text{ dacă și numai dacă}$$

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

Drumuri de cost minim - Rezolvarea problemei P2

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

Proof:

" \Leftarrow ". Fie P_{si}^* o soluție optimă a problemei P2 și $u_i = a(P_{si}^*)$.

Ipoteza (I) implică $u_s = 0$, i.e., prima ecuație a sistemului (B) este satisfăcută. Pentru $i \neq s$, drumul P_{si}^* are penultimul nod j . Dacă P_{sj} este drumul de la s la j determinat pe P_{si}^* de j , avem

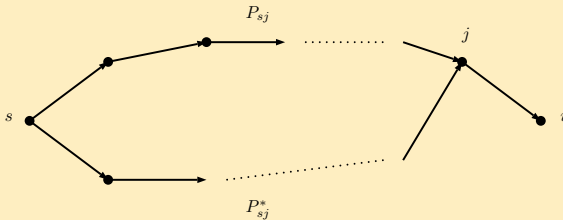
$$u_i = a(P_{si}^*) = a(P_{sj}) + a_{ji} \geq a(P_{sj}^*) + a_{ji} = u_j + a_{ji}.$$

Arătăm că $u_i = u_j + a_{ji}$. Presupunem că $u_i > u_j + a_{ji}$, i. e., $a(P_{sj}) > a(P_{sj}^*)$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 P2

Cazul 1. $i \notin V(P_{sj}^*)$. Atunci $P^1 = P_{sj}^* \circ (j, ji, i) \in \mathcal{P}_{si}$ și $a(P^1) = a(P_{sj}^*) + a_{ji} < a(P_{sj}) + a_{ji} = a(P_{si}^*)$, contradicție (P_{si}^* este un drum de cost minim).

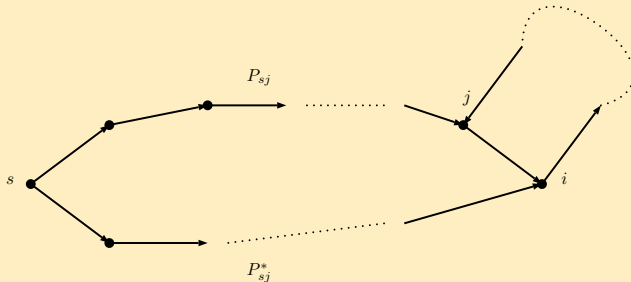


Drumuri de cost minim - Rezolvarea problemei P2

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

Cazul 2. $i \in V(P_{sj}^*)$. Fie $P_{sj}^* = P_{si} \circ P_{ij}$ cele două drumuri determinate de nodul i on P_{sj}^* . Atunci costul circuitului $C = P_{ij} \circ (j, ji, i)$ este $a(C) = a(P_{ij}) + a_{ji} = a(P_{sj}^*) - a(P_{si}) + a_{ji} = u_j + a_{ji} - a(P_{si})$ care este $\leq u_j + a_{ji} - a(P_{si}^*) = u_j + a_{ji} - u_i < 0$, contradicție (ipoteza (I) este încălcată).

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



Drumuri de cost minim - Rezolvarea problemei P2

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

Astfel partea " \Leftarrow " este demonstrată.

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

Remarcă 1

Am demonstrat mai sus că dacă j este nodul dinaintea lui i pe un drum de cost minim de la s la i , atunci drumul de la s la j determinat de j on acest drum este un drum de cost minim de la s la j . Inductiv, urmează:

Principiul de optimalitate al lui Bellman: Dacă P_{si}^* este un drum de cost minim de la s la i , atunci $\forall j \in V(P_{si}^*)$, dacă $P_{si}^* = P_{sj} \circ P_{ji}^*$, atunci P_{sj} (respectiv P_{ji}^*) este un drum de cost minim de la s la j (respectiv de la j la 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 *

Drumuri de cost minim - Rezolvarea problemei P2

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

" \Rightarrow ". Arătăm că dacă (u_1, \dots, u_n) este o soluție a sistemului (B), atunci

(a) $\exists P_{si} \in \mathcal{P}_{si}$ așa încât $u_i = a(P_{si}), \forall i \in V$.

(b) $\forall i \in V, u_i = \min \{a(P) : P \in \mathcal{P}_{si}\} (= a(P_{si}))$.

(a) Dacă $i = s$, atunci $u_s = 0$ și drumul P_{ss} satisface $a(P_{ss}) = 0 = u_s$.
Dacă $i \neq s$, considerăm următorul algoritm

$v \leftarrow i; k \leftarrow 0;$

while $v \neq s$ **do**

 find w a. î. $u_v = u_w + a_{wv}; // \exists w$ deoarece u_v stasface (B)

$i_k \leftarrow v; k++;$ $v \leftarrow w;$

Algoritmul determină drumul $P : (s =) i_{k+1}, i_{k+1} i_k, i_k, \dots, i_1, i_1 i_0, i_0 (= i)$ cu $P \in \mathcal{P}_{si}$ și

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

Drumuri de cost minim - Rezolvarea problemei P2

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

$$\begin{aligned}a(P) &= a(i_{k+1} i_k) + \cdots + a(i_1 i_0) = \\(u_{i_k} - u_{i_{k+1}}) &+ (u_{i_{k-1}} - u_{i_k}) + \cdots + (u_{i_0} - u_{i_1}) = \\u_{i_0} - u_{i_{k+1}} &= u_i - u_s = u_i,\end{aligned}$$

(altfel obținem un circuit de cost 0, în contradicție cu ipoteza (I)).

Cu notațiile din algoritmul de mai sus avem $u_i = u_{i_1} + a_{i_1 i}$.

(b) Fie $\bar{u}_i = a(P_{si}^*)$, $\forall i \in V$. Din demonstrația anterioară, \bar{u}_i , $i = \overline{1, n}$ satisface sistemul (B).

Presupunem că $u = (u_1, \dots, u_n) \neq (\bar{u}_1, \dots, \bar{u}_n) = \bar{u}$.

Deoarece $u_s = \bar{u}_s = 0$, urmează că există un $i \neq s$ așa încât $u_i \neq \bar{u}_i$ și $\forall j \in V(P_{si})$. $j \neq i$, $u_j = \bar{u}_j$, unde P_{si} este drumul construit la (a) pentru \bar{u}_i .

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

Drumuri de cost minim - Rezolvarea problemei P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 3

- Din demonstrația de mai sus urmează că pentru a rezolva P2 este suficient să se găsească o soluție a sistemului de ecuații (B). Drumurile de cost minim corespunzătoare pot fi obținute ca la (a) din demonstrația de mai sus: dacă avem $u_i = u_k + a_{ki}$ atunci k este nodul dinaintea lui i pe drumul de cost minim de la s la i (de cost u_i). În algoritmul care rezolva sistemul (B) menținem un tablou $before[1..n]$ cu elemente din $V \cup \{0\}$ cu semnificația finală $before[i] = \text{nodul dinaintea lui } i \text{ pe un drum de cost minim de la } s \text{ la } i$. Nodurile de pe acest drum pot fi determinate în $\mathcal{O}(n)$ prin construcția secvenței $i, before[i], before[before[i]], \dots, s$.

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 P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 5

- Dacă, în problemele **P1 - P3**, G este un graf și nu un digraf, putem folosi algoritmi pentru digrafuri înlocuind fiecare muchie a lui G cu o pereche simetrică de arce, fiecare cu același cost ca muchia. Această abordare funcționează doar pentru muchii de cost nenegativ (dacă o muchie are cost negativ, atunci 2-circuitul corespunzător format cu cele două arce simetrice are cost negativ, deci ipoteza (I') este încălcată).

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

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

Spunem că un graf $G = (V, E)$ este **rar** dacă $m \leq cn^2 / \log n$ ($n = |V|, m = |E|$). Motivul este acela că putem reprezenta matricea de adiacență A a lui G folosind doar $\mathcal{O}(n^2 / \log n)$ spațiu de memorie așa încât răspunsul la întrebarea " $a(i, j) = 1?$ " să poată fi dat în $\mathcal{O}(1)$.

Descrieți o astfel de reprezentare.

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

Exercițiul 2.

Arătați că nu există o ordonare e_1, e_2, \dots, e_{10} a muchiilor unui graf K_5 , așa încât: e_{10} și e_1 nu sunt adiacente și e_i și e_{i+1} nu sunt adiacente pentru orice $1 \leq i \leq 9$.

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 3. Fie $G = (V, E)$ un graf de ordin n și dimensiune m cu matricea de adiacență A . Din mulțimea celor 2^m orientări posibile ale tuturor muchiilor sale alegem una și considerăm matricea de incidență arc-nod $Q \in \mathcal{M}_{n \times n}(\{-1, 0, 1\})$.

$$q_{ve} = \begin{cases} -1, & \text{dacă } v \text{ este extremitatea inițială a arcului } e \\ 1, & \text{dacă } v \text{ este extremitatea finală a arcului } e \\ 0, & \text{dacă } e \text{ nu este incident cu } v. \end{cases}$$

Arătați că $A + QQ^T$ este matrice diagonală și aflați interpretarea combinatorială a elementelor sale diagonale.

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 cu $V = \{v_1, v_2, \dots, v_n\}$ și $E = \{e_1, e_2, \dots, e_m\}$. Fie $B = (b_{ij}) \in \mathcal{M}_{n \times m}(\{-1, 0, 1\})$ matricea de incidență a lui D , unde

$$b_{ij} = \begin{cases} 1, & \text{dacă } e_j \text{ este incident dinspre } v_i \\ -1, & \text{dacă } e_j \text{ este incident către } v_i \\ 0, & \text{altfel} \end{cases}.$$

Arătați că $\det(M) \in \{-1, 0, 1\}$ pentru orice submatrice pătratică, M , a lui B (i. e., B este o matrice total unimodulară).

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

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

Exercițiul 5. Fie $G = (S, T; E)$ un graf bipartit cu $V = S \cup T = \{v_1, v_2, \dots, v_n\}$ și $E = \{e_1, e_2, \dots, e_m\}$. Fie $B = (b_{ij}) \in \mathcal{M}_{n \times m}(\{0, 1\})$ matricea de incidență a lui G , unde

$$b_{ij} = \begin{cases} 1, & \text{dacă } e_j \text{ este incidentă cu } v_i \\ 0, & \text{altfel} \end{cases}.$$

Arătați că $\det(M) \in \{-1, 0, 1\}$ pentru orice submatrice pătratică a lui B (i. e., B este o matrice total unimodulară).

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

Exercițiul 6. Arătați că un digraf are o singură ordonare topologică dacă și numai dacă are un drum Hamiltonian iar toate celelate arce sunt orientate înainte relativ la direcția de parcurgere a acestui drum.

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 7. *Diametrul* unui graf G este cea mai mare distanță dintre any două noduri din G . Două noduri formează o **pereche diametrală de noduri** dacă distanța dintre ele este cât diametrul. Arătați că următorul algoritm determină pereche diametrală de noduri într-un arbore dat T :

- 1 plecând dintr-un nod al lui T , parcurge *bfs* (Breadth First Search) arborele T ; fie u ultimul nod vizitat.
- 2 mai parcurge o dată *bfs* arborele T plecând din nodul u ; fie v ultimul nod vizitat.
- 3 returnează perechea (u, v) .

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

Exercițiul 8. Arătați că o parcurgere DFS poate fi utilizată pentru a determina un circuit par într-un graf 3-regulat în $\mathcal{O}(n)$.

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