



## 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 4 (săptămâna 20-24 octombrie)

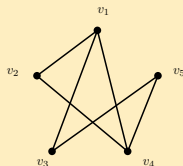
## Matrici asociate - Matricea de adiacență

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.



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

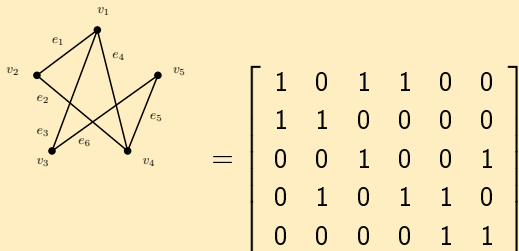
## Matrici asociate - Matricea de incidență

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.



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**.

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

Fie  $G$  un digraf cu  $V(G) = \{v_1, \dots, v_n\}$  și  $E(G) = \{e_1, \dots, e_m\}$ . **Matricea de incidență nod-arc** a lui  $G$  este matricea  $B = (b_{ij})_{1 \leq i, j \leq n} \in \mathcal{M}_{n \times m}(\{-1, 0, 1\})$ , unde

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



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

- 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  $\mathcal{O}(n + 2m)$ .
- Dacă  $G$  este digraf, atunci spațiul necesar este  $\mathcal{O}(n + m)$ .
- Listele de adiacență pot fi implementate folosind liste înlănțuite sau tablouri.
- Testarea dacă un nod  $u$  este adiacent cu un altul  $v$  în  $G$  necesită  $\mathcal{O}(d_G(u))$  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).

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.



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

**Proprietăți ale BFS.** Nu este dificil de arătat că:

- $S = \{u \in V : label(u) \geq 0\}$ ;
- $\forall u \in V, label(u) = d_G(s, u)$  (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 componente 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).

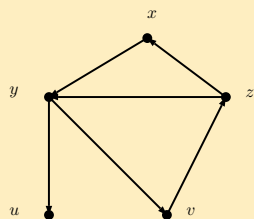
Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru  
- Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \*

# 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 -

## Exemplu

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



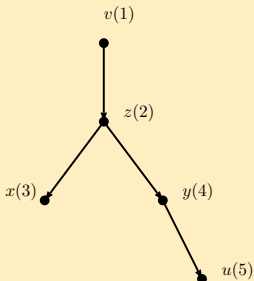
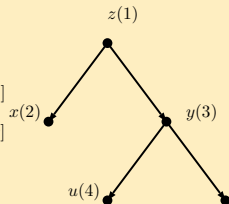
$$A(x) = [y]$$

$$A(y) = [u, v]$$

$$A(z) = [x, y]$$

$$A(u) = \emptyset$$

$$A(v) = [z]$$



(Am marcat în paranteze ordinea de vizitare.)

Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru -  
- 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
     $label(u) \leftarrow -1$ ;  $parent(u) \leftarrow -1$ ;
 $label(s) \leftarrow 0$ ;  $parent(s) \leftarrow 0$ ;
crează o stivă  $S$  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 \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \*





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





# 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 \* C. Croitoru  
- 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  $\infty$ ).
- 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 comunicare (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)

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

Luând  $a'(e) = -\log a(e)$ ,

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

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

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

2. **Rețele PERT - Metoda drumului critic (Critical path method - CPM).** **PERT (Path 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  $A_j$  poate începe doar după ce activitatea  $A_i$  s-a 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.

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  $s_i p$  pentru fiecare activitate  $A_p$  (în  $s$  nu intră vreun arc) și a unui nod  $t$  corespunzând finalizării proiectului, legat prin arce  $j_p t$  pentru fiecare activitate  $A_p$  (din  $t$  nu iese vreun arc). Toate aceste arce au cost 0.
- 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 maxim 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 \*





- 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  $j - a_i$ ).

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

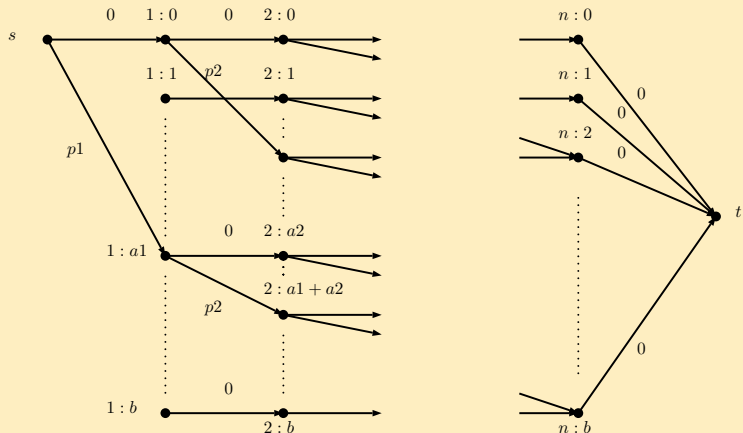
### Remarci 2

- 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 \*  
- 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

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

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

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

$\forall i \in V(G), \exists P_{si}^* \in \mathcal{P}_{si}$  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  $i \neq s$ , fie  $j$  penultimul nod de pe drumul  $P_{si}^*$ . 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 \*

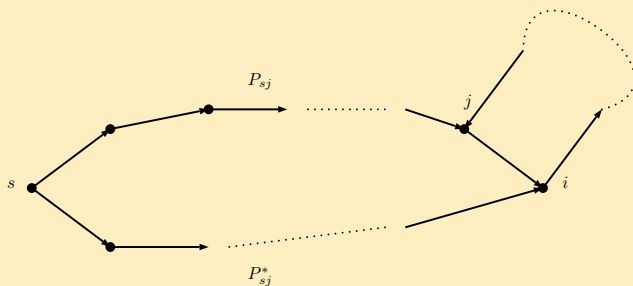


## Drumuri de cost minim - Rezolvarea problemei P2

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. Croitoru - 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 -

Astfel partea " $\Leftarrow$ " este demonstrată.

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

### Remarca 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  $j$  de pe 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.  $u_v = u_w + a_{vw}; // \exists w$  deoarece  $u_v$  satisface (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

$$\begin{aligned}a(P) &= a(i_{k+1}i_k) + \cdots + a(i_1i_0) = \\& (u_{i_k} - u_{i_{k+1}}) + (u_{i_{k-1}} - u_{i_k}) + \cdots + (u_{i_0} - u_{i_1}) = \\& u_{i_0} - u_{i_{k+1}} = u_i - u_s = u_i,\end{aligned}$$

În fiecare iterație **while**  $w \notin \{i_0, \dots, i_{k-1}\}$  (altfel obținem un circuit de cost 0, în contradicție cu ipoteza (I)). (Cu notațiile din algoritmul de mai sus avem  $u_i = u_{i_1} + a_{i_1i}$ .)

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

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

Rulăm algoritmul de la (a) pentru  $\bar{u}_i$ ,  $i = \overline{1, n}$  și fie  $\bar{P}_{s_i}$ ,  $i = \overline{1, n}$ , drumurile care rezultă.

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





### Remarca 3

Dacă algoritmi care rezolvă sistemul de ecuații (B) ocolesc (prin întreținerea tabloului *before*) circuitele de cost 0, atunci problema P2 este rezolvată, chiar dacă se pierde unicitatea soluției. Astfel, acești algoritmi vor rezolva P2 în ipoteza

(I')  $a(C) \geq 0$ , pentru toate circuitele  $C$  din  $G$ .

### Remarca 4

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).



## Exerciții pentru seminarul 4

C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - 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  $1 \leq i \leq 9$ .

Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* 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 3.** Fie  $E = \{e_1, e_2, \dots, e_m\}$  o ordonare a muchiilor unui graf conex  $G = (V, E)$  de ordin  $n$ . Pentru orice submulțime  $A \subseteq E$  definim  $x^A \in GF^m$  vectorul caracteristic  $m$ -dimensional al lui  $A$ :  $x_i^A = 1 \Leftrightarrow e_i \in A$ . ( $GF^m$  este spațiul vectorial  $m$ -dimensional 0 - 1 peste  $\mathbb{Z}_2$ .)  
Arătați că:

- (a) submulțimea vectorilor caracteristici ai tuturor tăieturilor din  $G$  la care se adaugă vectorul nul este un subspațiu  $X$  al lui  $GF^m$ ;
- (b) submulțimea vectorilor caracteristici ai tuturor circuitelor din  $G$  generează un subspațiu,  $U$ , al lui  $GF^m$  care este ortogonal pe  $X$ ;
- (c)  $\dim(X) \geq n - 1$ ;
- (d)  $\dim(U) \geq m - n + 1$ ;
- (e) inegalitățile de mai sus sunt, de fapt, egalități.

## Exerciții pentru seminarul 4

C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms  
\* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - 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  $G = (V, E)$  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 nod-arc  $Q \in \mathcal{M}_{n \times m}(\{-1, 0, 1\})$ .

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

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 \*



**Exercițiul 6.** 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).

**Exercițiul 7.** Fie  $G$  un graf cu  $n$  noduri și  $m$  muchii.

- Arătați că  $G$  este bipartit dacă și numai dacă  $G$  nu conține circuite (induse) impare.
- Descrieți un algoritm de complexitate timp  $\mathcal{O}(n + m)$  care să decidă dacă un graf dat este bipartit. (Un algoritm pentru recunoașterea grafurilor bipartite.)

## Exerciții pentru seminarul 4

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

**Exercițiul 8.** Fie  $M_G$  matricea de incidență muchie-nod a unui graf dat  $G = (V, E)$ , adică  $M_G = (m_{ij})_{\substack{1 \leq i \leq m, \\ 1 \leq j \leq n}}$ , unde

$$V = \{v_1, v_2, \dots, v_n\}, E = \{e_1, e_2, \dots, e_m\}.$$

$$m_{ij} = \begin{cases} 1 & \text{dacă } e_i \text{ este incidentă cu } v_j \\ 0 & \text{altfel} \end{cases}$$

- (a) Arătați că dacă  $T$  este un arbore, atunci prin îndepărtarea din  $M_T$  a unei coloane corespunzătoare unui nod dat se obține o matrice pătratică nesingulară.
- (b) Arătați că dacă  $C$  este un circuit, atunci  $M_C$  este matrice nesingulară dacă și numai dacă  $C$  este impar.

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

## Exerciții pentru seminarul 4

C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph 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 9.** Fie  $G = (V, E)$  un graf cu  $n$  noduri și  $m \geq 1$  muchii. Considerăm următorul algoritm:

```
 $G' \leftarrow G;$   
while  $(\exists u \in V(G') \text{ a. } \hat{\text{i.}} d_{G'}(u) < m/n)$  do  
     $G' \leftarrow G' - u;$   
return  $G';$ 
```

- Determinați complexitatea timp a unei implementări eficiente a acestui algoritm.
- Arătați că graful returnat,  $G'$ , este nenul (are și noduri, dar și muchii).
- Arătați că orice graf conține un drum de lungime cel puțin  $m/n$ .

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 4

C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - 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 10. Diametrul** unui graf  $G$  este cea mai mare distanță dintre orice două noduri din  $G$ . Două noduri formează o **pereche de noduri diametral opuse** dacă distanța dintre ele este cât diametrul. Arătați că următorul algoritm determină pereche de noduri diametral opuse într-un arbore dat  $T$ :

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

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 \*

## Exerciții pentru seminarul 4

C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms  
\* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - 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 11.** 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 -

**Exercițiul 12.**

- (a) Arătați că pentru un graf bipartit cu  $n$  noduri și  $m$  muchii avem  $4m \leq n^2$ .
- (b) Descrieți un algoritm de complexitate timp  $\mathcal{O}(n + m)$  care să decidă dacă un graf dat (cu  $n$  noduri și  $m$  muchii) este complementul unui graf bipartit.

\* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph  
Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* 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 4

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

**Exercițiul 13.** Demonstrați că un graf  $G$  este bipartit dacă și numai dacă orice subgraf indus  $H$  al lui  $G$  satisface inegalitatea:  $2\alpha(H) \geq |H|$ .

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

**Exercițiul 14.** Fie  $G = (S, T; E)$  un graf bipartit și  $X \in \{S, T\}$ .  $G$  se numește  **$X$ -lanț** dacă nodurile lui  $X$  pot fi ordonate:  $x_1, x_2, \dots, x_k$  ( $|X| = k$ ) astfel ca

$$N_G(x_1) \supseteq N_G(x_2) \supseteq \dots \supseteq N_G(x_k)$$

- (a) Arătați că  $G$  este  $S$ -lanț dacă și numai dacă este  $T$ -lanț.
- (b) Să presupunem că  $G$  (care este bipartit) are ordinul  $n$ , dimensiunea  $m$  și este reprezentat folosind liste de adiacență. Descrieți un algoritm de recunoaștere a unui  $S$ -lanț de complexitate timp  $\mathcal{O}(n + m)$ .

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

## Exerciții pentru seminarul 4

C. Croitoru - Graph Algorithms \* C. Croitoru - Graph Algorithms \* C. Croitoru - 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 15.** Fie  $G$  un graf; notăm cu  $b(G)$  graful obținut din  $G$  prin inserarea simultană a câte unui nou nod pe fiecare muchie a lui  $G$ .

- Arătați că  $b(G)$  este graf bipartit.
- Demonstrați că  $G \simeq H$  dacă și numai dacă  $b(G) \simeq b(H)$ . De aici deduceți că testarea izomorfismului între două grafuri poate fi redusă în timp polinomial la testarea izomorfismului între două grafuri bipartite.

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

**Exercițiul 16.** Fie  $G$  un graf bipartit; arătați că  $G$  este conex dacă și numai dacă admite o singură bipartiție cu mulțimi stabile.

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