

Grafuri Teorie

ianuarie 2021

Cuprins

1 CURS 1: Descrierea cursului. Vocabularul Teoriei grafurilor.	3
2 CURS 2: Vocabularul Teoriei grafurilor.	47
3 CURS 3: Vocabularul Teoriei grafurilor. Probleme de drum minim in (di)grafuri.	88
4 CURS 4: Probleme de drum minim in (di)grafuri.	146
5 CURS 5: Probleme de conexiune in (di)grafuri (teoremele lui Menger, Konig si Hall). Arbori partiali.	195
6 CURS 6: Arbori partiali de cost minim. Cuplaje maxime si acoperiri minime.	227
7 CURS 7: Problema cuplajului maxim. Teoremele lui Berge si Tutte. Fluxuri in retele.	258
8 CURS 8: Fluxuri in retele. Sectiuni, drumuri de crestere. Teorema flux maxim - sectiune minima. Algoritmii F&F si E&K.	305
9 CURS 9: Prefluxuri. Algoritmul Ahuja&Orlin. Aplicatii combinatoriale ale fluxurilor in retele.	334
10 CURS 10: Fluxuri de cost minim. Reduceri poliomiale pentru probleme de decizie pe grafuri.	367
11 CURS 11: Reduceri polinomiale pentru probleme de decizie pe grafuri. Abordari ale problemelor NP-hard pe grafuri.	402
12 CURS 12: Grafuri planare.	430
13 CURS 13: Tree decompositions.	448

14 Diverse cursuri explicate poate mai bine	460
15 ALGORITMI	562
15.1 BFS	562
15.2 BFS - liste alocate dinamic	563
15.3 BFS - liste alocate static	564
15.4 BFS -matrice de adiacenta	565
15.5 DFS - nerecursiv - matrice de adiacenta	566
15.6 DFS - recursiv - matrice de adiacenta	567
15.7 Algoritmul lui Dijkstra	567
15.8 Algoritmul Ford-Fulkerson	568
15.9 Algoritmul de colorare greedy	568
15.10 Algoritmul Bellman-Ford	568
15.11 Metoda generala MST - Algoritmul lui Prim	569
15.12 Algoritmul lui Kruskal	570
15.13 Algoritmul lui Hopcroft si Karp - pentru gasirea cuplajului maxim	571
15.14 Algoritmul de Unificare	573
15.15 Algoritmul lui Lee	574
15.16 Grafuri bipartite	575
15.17 Sortare topologica	576
15.18 Drum minim	577
15.19 Verificare graf conex	578
15.20 Determinarea si afisarea componentelor conexe	579
15.21 Determinarea si afisarea componentelor conexe - optimizat - numarul maxim de varfuri de grad par	580
15.22 Lungime minima intre 2 varfuri	581
15.23 Determinarea si afisarea unui ciclu eulerian	582
15.24 Determinarea si afisarea unui ciclu hamiltonian	583
15.25 Clase de echivalenta	584
16 .	584

1 CURS 1: Descrierea cursului. Vocabularul Teoriei grafurilor.

Fie n = noduri si m = muchii.

- graf complet de ordin n : $m = \frac{n(n-1)}{2}$
- numarul grafurilor partiale: 2^m
- numarul subgrafurilor: $2^m - 1$
- graf neorientat: $2m = \text{suma gradelor}$
- graf neorientat: $\exists x$ numar par de varfuri de grad impar
- numarul grafurilor neorientate cu n varfuri: $2^{C_n^2}$

\times multime: $|X| = \text{card } X$

$|X| = k \Rightarrow X \text{ este } \star k\text{-mult.}$

G graf $G = (V, E)$ MUCHIE
 \uparrow NOD

$|G| = |V(G)|$ - ord G

$|E(G)| = \underline{\text{dimu}} G$

$NG(u) = \{v \in V(G) : uv \in E(G)\}$ - vecinătatea nod u

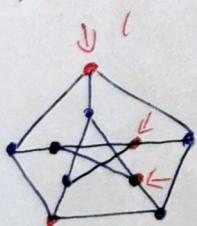
$d_G(u) = |NG(u)|$ - gradul unui nod

nod izolat $\longleftrightarrow d_G(v) = 0$

pendant $\longleftrightarrow d_G(v) = 1$

$$\sum_{v \in V} d_G(v) = 2|E|$$

Mult. stabilit (mult independentă de noduri)



$S \subseteq V$ astfel că $\binom{|S|}{2} \cap E = \emptyset$ (nu există nicio muchie între nodurile lui S)

Nr de stabilitate

$\alpha(G)$ - card max. al unei mult. stabile

Cuplaj - M-dacă oricare 2 mulțimi ale sale NU au noduri în comun

Nr de cuplaj - $v(G)$ sau $\sigma(G)$ (să mor de înțeleg)
 card max al unei cuplaj

p-colorare - \Rightarrow fct. $c: V \rightarrow \{1..p\}$ at. $c(e) \neq c(f)$
pt fiecare urecă

Clasă de colorare - mult. nodurilor de aceeași culoare

Nr. cromatic - cea mai mică val a lui p ai
G are o p-colorare. $\chi(G) \leq |G|$

p-mulțime-colorare - \Rightarrow fct. $c: E \rightarrow \{1..p\}$ at. $c(e) \neq c(f)$
pt $e, f \in E$ cu
 $|e \cap f| = 1$
Indexul cromatic - cea mai $<$ val. a lui p ai
G are o p-mulțime colorare $\chi'(G) \leq |E(G)|$

2 gr. sunt izomorfe dacă \exists o bijectie între
mult. lor de noduri care induce o bijectie
între mult. lor de mulțimi. WTF?? $G_1 \cong G_2$

PE scurt: ai un graf, poti rearanja
acel graf păstrând următoarele și nodurile
ai. să obtii alt graf

Algoritmica grafurilor - Cursul 1

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 **Descrierea cursului**
 - **Informații legate de curs**
- 2 **Aplicații ale teoriei grafurilor**
- 3 **Vocabularul teoriei grafurilor**
 - **Definiția grafului**
- 4 **Exerciții pentru seminarul din săptămâna următoare**
- 5 **Exerciții rezolvate (parțial)**

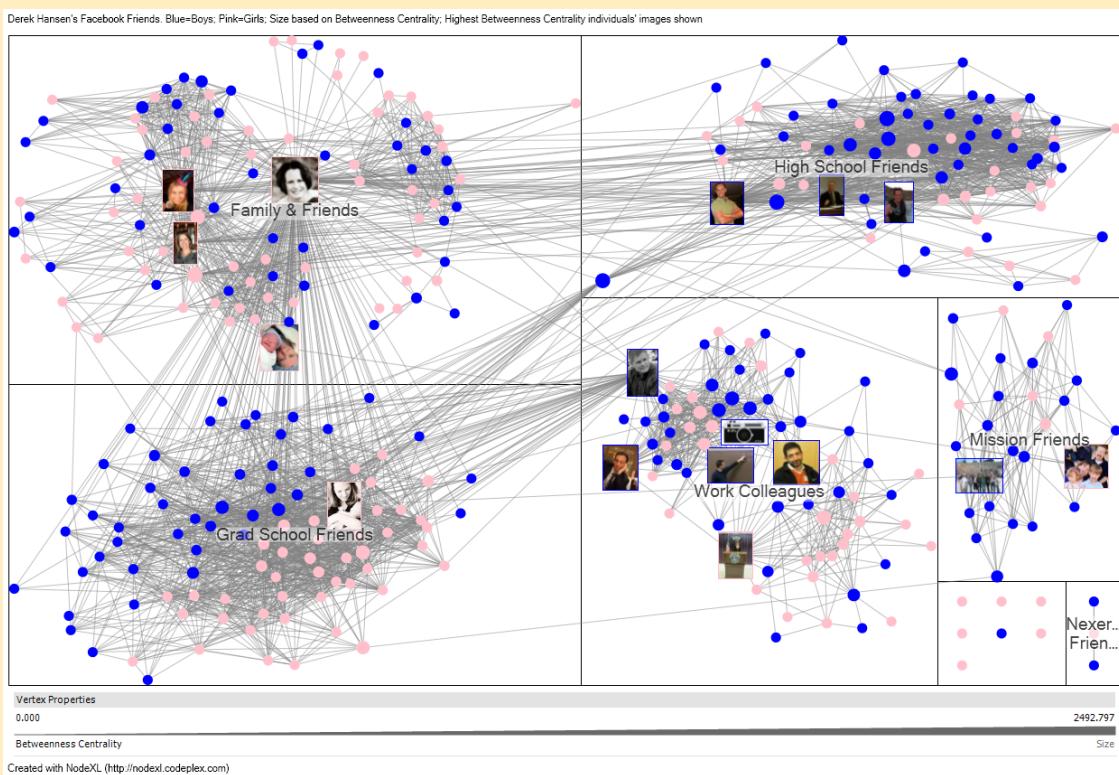


Figure: Facebook/Twitter

Aplicații ale teoriei grafurilor

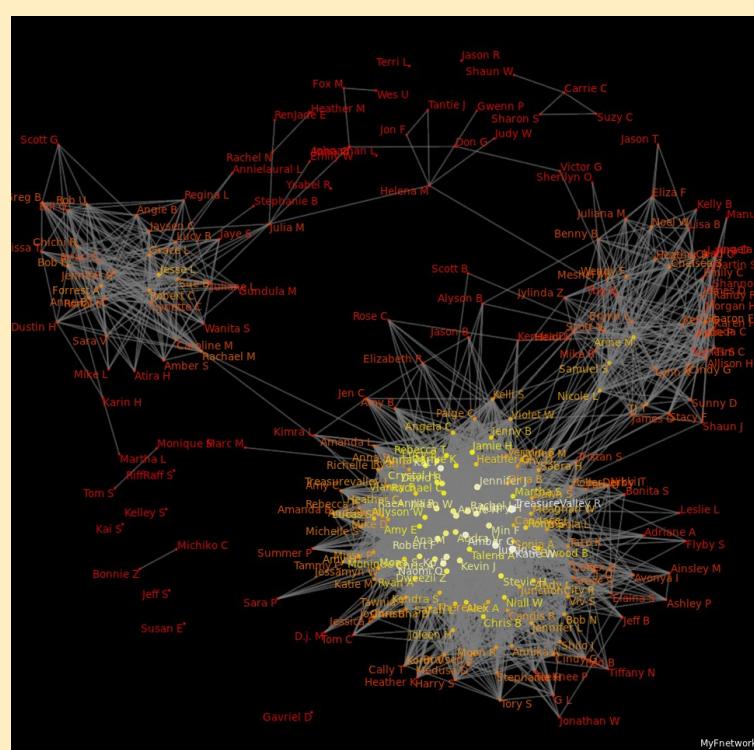


Figure: A Facebook network of a few users

Grafurile sunt folosite pentru **analiza rețelelor** sociale/de știri (precum Facebook sau Twitter) pentru a determina caracteristici cum ar fi:

- nivelul de **conexiune**, densitatea;
 - influența utilizatorilor asupra rețelei (**centralitatea**, **potențialul rețelei sociale** (**social networking potential**));
 - nivelul de **segmentare**: măsura clusterizării;
 - robustețea sau stabilitatea structurală a rețelei.

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

- data mining și agregare, marketing;
 - analiza comportamentului și predicție în rețea;
 - colectarea de informații și analiza securității (supravegherea terorismului și a crimei organizate) etc.

Aplicații ale teoriei grafurilor

În afara studiului rețelelor sociale (un subiect la modă astăzi) există multe alte aplicații ale teoriei grafurilor:

- **Arbore parțiali de cost minim:** pentru conectarea eficientă a unor puncte de comunicație (e. g. în IT&C);
 - **Drumuri și circuite Euleriene:** **Problema poștasului chinez** - să se determine un drum/circuit de cost minim care trece prin fiecare muchie a unui graf o singură dată (pentru salubrizarea străzilor, expedierea poștei sau a unor servicii, colectarea deșeurilor etc.);
 - **Drumuri și circuite Hamiltoniene:** vizitarea eficientă a unui număr de puncte (dintr-un oraș, dintr-o țară etc); **Problema comis-voiajorului, Problema rutării vehiculelor;**

- **Colorarea grafurilor:** colorarea hărților (a fețelor unui graf planar), planificarea cursurilor/seminariilor (problema orarului), planificarea unei sesiuni, alocarea frecvențelor radio mobile, alocarea reședinților de memorie.

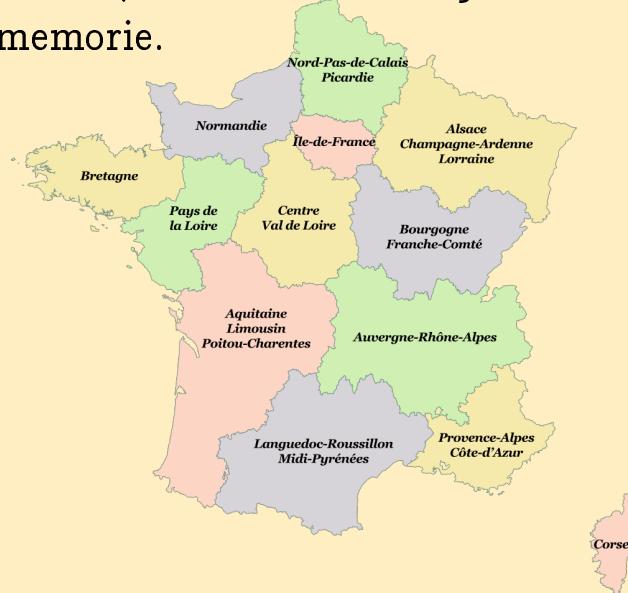


Figure: Regiunile Franței începând cu 2016

Aplicații ale teoriei grafurilor

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

- **Cuplaje:** probleme de asignare, în studii de chimie computațională și de chimie matematică asupra materialelor organice.

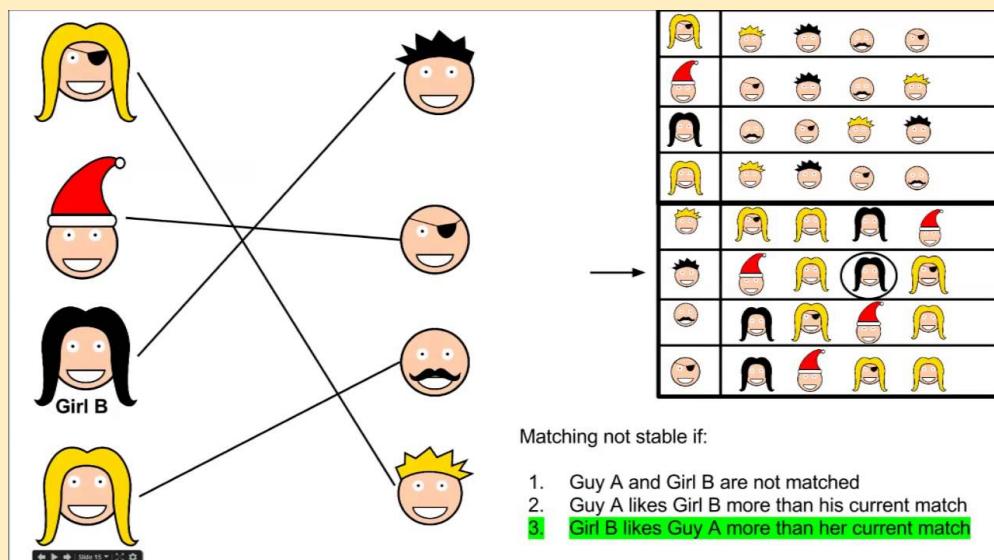


Figure: Gale Shapley algorithm

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

- **Cuplaje stabile:** repartizarea rezidenților în spitale, proceduri de admitere în învățământul superior, identificarea repartizării optime a organelor donate pentru transplant (e. g. rinichi), problema alocării proiectelor către studenți etc.
- **Fluxuri de valoare maximă:** determinarea nivelului de încărcare în rețelele de transport pentru îmbunătățirea condițiilor de trafic, reconstrucția imaginilor din proiecția razelor X (în tomografie), planificarea procesoarelor paralele etc.

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

Aplicații ale teoriei grafurilor

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

Aplicații în informatică

- În managementul bazelor de date, **bazele de date de tip graf** folosesc structurile de date tip graf pentru stocare și interogare.
- **Graph rewriting systems** folosite în **verificarea sistemelor software**.
- **Quantum computation**.
- **Modelarea documentelor web drept grafuri** și clusterizarea acestora.
- Aproximarea și compresia datelor.
- Modelarea rețelelor de senzori cu grafuri (folosind de exemplu diagrame Voronoi).
- Si multe altele.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo10ms * C. Croitoru - Graph Algorithms *

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

Notății

Pentru o mulțime finită X :

- $|X| = \text{card}(X) \in \mathbb{N}$ este **cardinalul** lui X ;
- Dacă $|X| = k$, atunci X este o **k -mulțime**;
- $2^X = \mathcal{P}(X)$ este **mulțimea părților** lui X : $2^X = \{Y : Y \subseteq X\}$,
 $|2^X| = 2^{|X|}$;
- $\binom{X}{k} = \{Y : Y \subseteq X, |Y| = k\}$, $\left|\binom{X}{k}\right| = \binom{|X|}{k}$.

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

Definiția grafului

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

Definiția 1

Un **graf** este o pereche $G = (V, E)$, unde:

- $V = V(G)$ o mulțime finită, nevidă; este **mulțimea nodurilor (vârfurilor)** lui G ;
- $E = E(G)$ este o submulțime a lui $\binom{V(G)}{2}$; este **mulțimea muchiilor** lui G .

$|G| = |V(G)|$ este **ordinul grafului G** , iar $|E(G)|$ este **dimensiunea sa**.

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

Definiția 2

Dacă $G = (V, E)$ și $e = uv = vu = \{u, v\} \in E$ este o muchie a lui G , spunem că

- e conectează (sau leagă) nodurile u și v ;
- nodurile u și v sunt adiacente sau u și v sunt vecine;
- e este incidentă cu u și v ;
- u și v sunt capetele (extremitățile) lui e .

Vecinătatea nodului u este $N_G(u) = \{v \in V(G) : uv \in E(G)\}$.

Două muchii e și f sunt adiacente dacă au un capăt în comun: $|e \cap f| = 1$.

Definiția grafului

Fie $G = (V, E)$ un graf și $v \in V$.

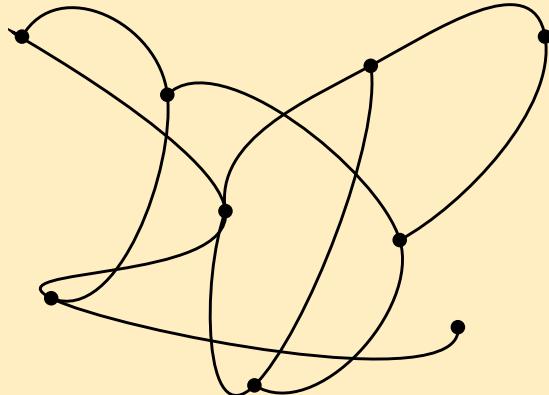
- Gradul unui nod v : $d_G(v) = |N_G(v)|$.
- v este un nod izolat dacă $d_G(v) = 0$ și pendant (sau frunză) dacă $d_G(v) = 1$.
- o proprietate utilă:

$$\sum_{v \in V} d_G(v) = 2|E|.$$

Un graf poate fi reprezentat în plan ca o figură constând dintr-o mulțime de puncte (forme geometrice mici: puncte, cercuri, pătrate etc) corespunzând vârfurilor sale și curbe care conectează vârfurile corespunzătoare muchiilor din graf.

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

Un exemplu:

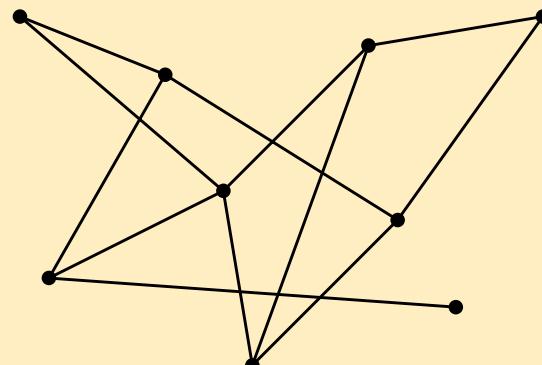


C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

Definiția grafului

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

Același graf din nou:



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

Definiția grafului

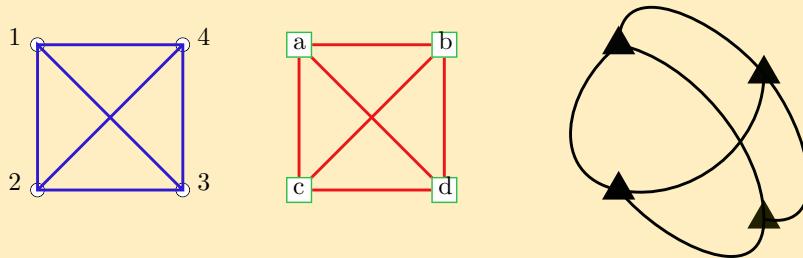
[Jump to Table of contents](#)

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

Putem adăuga etichete (nume, numere etc) și culori nodurilor și muchiilor obținând reprezentări vizuale mai bune.

Mai jos sunt trei reprezentări vizuale ale aceluiași graf:

$$G = (\{1, 2, 3, 4\}, \{12, 13, 14, 23, 24, 34\})$$



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

Definiția grafului

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

Definiția 3

Mulțime stabilă (sau **mulțime independentă de noduri**) în $G = (V, E)$:

$S \subseteq V$ astfel încât $\binom{S}{2} \cap E = \emptyset$.

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

Cu alte cuvinte $S \subseteq V$ este o mulțime stabilă a lui G dacă nu există nicio muchie între nodurile sale.

Notație

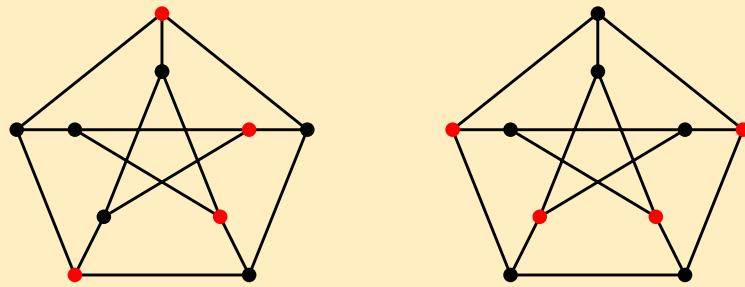
Cardinalul maxim al unei mulțimi stable a lui G este **numărul de stabilitate** (sau **numărul de independentă**) al lui G și se notează cu $\alpha(G)$.

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

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

În următorul graf (al lui Petersen) avem două mulțimi stabile de cardinal maxim (de ce?):



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

Definiția grafului

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

Mulțimi stabile: Problemă de optimizare

P1 Input: G graf.

Output: $\alpha(G)$ și o mulțime stabilă S cu $|S| = \alpha(G)$.

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

Mulțimi stabile: Problemă de decizie

SM Instanță: G graf, $k \in \mathbb{N}$.

Întrebare: Există o mulțime stabilă S în G , astfel încât $|S| \geq k$?

NP-completă (Karp, 1972).

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

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

Definiția 4

Cuplaj (multime independentă de muchii) în $G = (V, E)$: $M \subseteq E$ astfel
încât pentru orice $e, f \in M$, dacă $e \neq f$, atunci $e \cap f = \emptyset$.

Cu alte cuvinte, $M \subseteq E$ este un cuplaj dacă oricare două muchii ale sale
nu noduri în comun.

Notație

Cardinalul maxim al unui cuplaj în G este numit **numărul de cuplaj**
(**numărul de muchie-independentă**) al lui G și se notează cu $\nu(G)$.

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

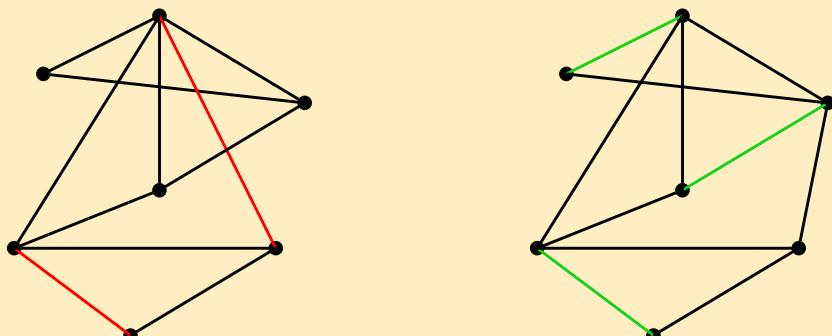
Definiția grafului

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

Pentru următorul graf sunt marcate două cuplaje cu roșu și verde (al
doilea fiind de cardinal maxim - de ce?):

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



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

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

Cuplaj maxim: Problemă de optimizare

P2 Input: G graf.

Output: $\nu(G)$ și un cuplaj M cu $|M| = \nu(G)$.

Edmonds (1965) a arătat că $P2 \in P$.

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

Notă

Problemele $P1$ și $P2$ sunt similare: în amândouă se cere să se determine un membru de cardinal maxim al unei familii de mulțimi relativ la un graf dat. Ce face ca ele să fie diferite?

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

Definiția grafului

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

Definiția 5

Pentru $p \in \mathbb{N}$, o **p -colorare** a grafului $G = (V, E)$ este o funcție $c : V \rightarrow \{1, \dots, p\}$ astfel încât $c(u) \neq c(v)$ pentru fiecare $uv \in E$.

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

Merită notat că multimea tuturor nodurilor cu aceeași culoare este o mulțime stabilă (se mai numește **clasă de colorare**). Deoarece noduri adiacente au culori diferite, o p -colorare corespunde unei partiții a lui V cu cel mult p mulțimi stabile (sau **clase de colorare**).

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

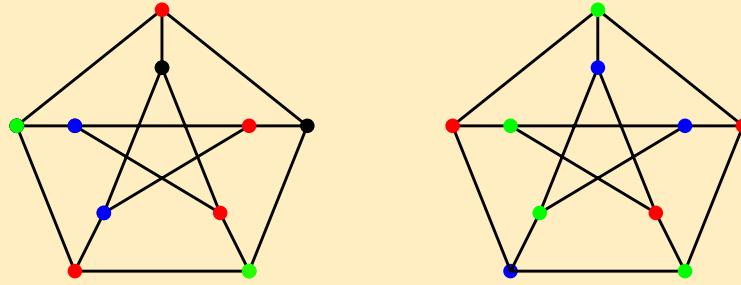
Notație

Numărul cromatic al grafului G este cea mai mică valoare a lui p astfel încât G are o p -colorare. Acest parametru se notează cu $\chi(G)$. ($\chi(G) \leq |G|$ - de ce?)

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

Exemplu

Pentru următorul graf avem marcate două colorări ($\chi(G) = 3!$)



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

Definiția grafului

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

Colorarea nodurilor: Problemă de optimizare

P3 Input: G graf.
Output: $\chi(G)$ și o $\chi(G)$ -colorare.

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

Colorarea nodurilor: Problemă de decizie

COL Instanță: G graf, $p \in \mathbb{N}$.
Întrebare: Există o p -colorare a lui G ?

NP-completă pentru $p \geq 3$ (Karp, 1972).

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

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

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

Definiția 6

Pentru $p \in \mathbb{N}$, o **p -muchie colorare** a grafului $G = (V, E)$ este o funcție $c : E \rightarrow \{1, \dots, p\}$ astfel încât $c(e) \neq c(f)$ pentru orice $e, f \in E$ cu $|e \cap f| = 1$.

Se poate observa că, dată o p -muchie colorare, o mulțime de muchii cu aceeași culoare este un cuplaj. Astfel, o p -muchie colorare corespunde unei partitii a lui E cu cel mult p cuplaje.

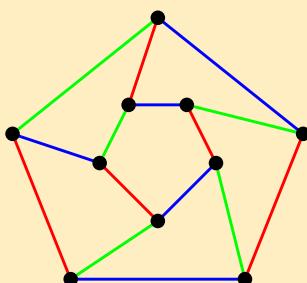
Notatie

Indexul cromatic al grafului G : cea mai mică valoare a lui p astfel încât G are o p -muchie colorare. Acest parametru este notat cu $\chi'(G)$. ($\chi'(G) \leq |E(G)|$ - de ce?)

Definiția grafului

Exemplu

Pentru următorul graf am marcat o colorare a muchiilor ($\chi'(G) = 3!$)



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

Colorarea muchiilor: Problemă de optimizare

P3 Input: G graf.
Output: $\chi'(G)$ și o $\chi'(G)$ -colorare.

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

Colorarea muchiilor: Problemă de decizie

COL Instanță: G graf, $p \in \mathbb{N}$.
Întrebare: Există o p -muchie colorare of G ?

NP-completă pentru $p \geq 3$ (Holyer, 1984).

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

Definiția grafului

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

Definiția 7

Două grafuri $G_1 = (V_1, E_1)$ și $G_2 = (V_2, E_2)$ sunt **izomorfe** dacă există o bijecție $\varphi : V_1 \rightarrow V_2$ astfel încât pentru orice două noduri $u_1, v_1 \in V_1$, u_1 și v_1 sunt adiacente în G_1 (i. e., $u_1 v_1 \in E_1$) dacă și numai dacă $\varphi(u_1)$ și $\varphi(v_2)$ sunt adiacente in G_2 (i. e., $\varphi(u_1)\varphi(v_2) \in E_2$).

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

Cu alte cuvinte, două grafuri sunt izomorfe dacă există o bijecție între mulțimile lor de noduri care induce o bijecție între mulțimile lor de muchii.

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

Notație

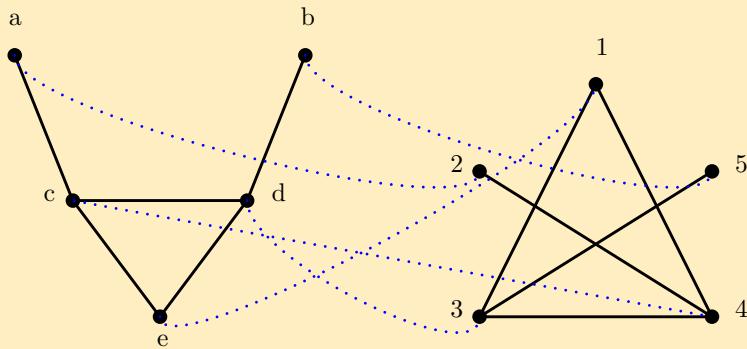
$G_1 \cong G_2$.

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

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

Exemplu

Grafurile de mai jos sunt izomorfe.



Definiția grafului

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

Testarea izomorfismului: **Problemă de optimizare**

ISO Input: grafurile G_1 și G_2 .
Output: Sunt G_1 și G_2 izomorfe?

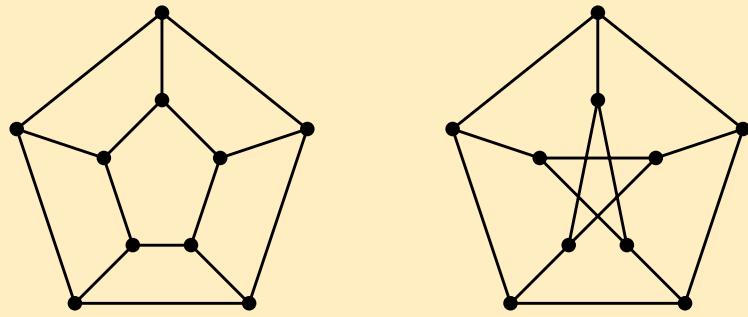
Nu se știe dacă este în P sau dacă este NP-completă.

Există un algoritm cu timp de execuție quasipolinomial (i. e., $2^{\mathcal{O}((\log n)^c)}$ pentru un $c > 0$, Babai, 2015).

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

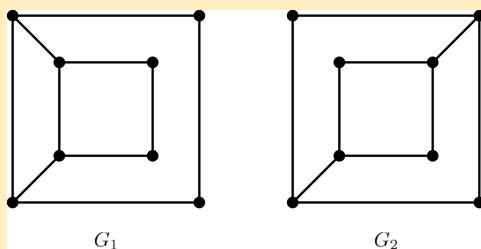
Exemplu

Următoarele două grafuri au aceleași ordine, dimensiuni și secvențe ale gradelor, dar nu sunt izomorfe (de ce?).

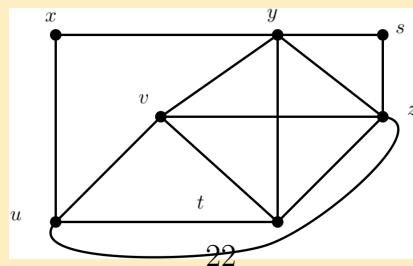


Exerciții pentru seminarul din săptămâna următoare

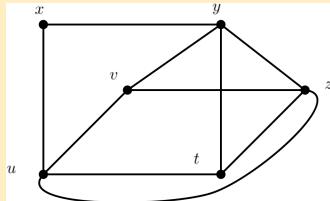
Exercițiul 1. Grafurile de mai jos sunt izomorfe?



Exercițiul 1'. Determinați numărul minim de culori cu care poate fi colorate nodurile grafului de mai jos:



Exercițiu 1". Determinați numărul de stabilitate al grafului de mai jos:



Exercițiu 2.

Fie $P(n) =$ "În orice graf cu cel puțin n noduri există trei noduri distincte care sunt două căte două adiacente sau două căte două neadiacente." Arătați că 6 este cea mai mică valoare a lui $n \in \mathbb{N}$, $n \geq 3$ pentru care $P(n)$ este adevărată.

Exercițiu 3.

- (a) Există un graf cu gradele 3, 3, 3, 3, 5, 6, 6, 6, 6, 6?
 - (b) Există un graf bipartit cu gradele 3, 3, 3, 3, 3, 5, 6, 6, 6, 6, 6, 6?
 - (c) Există un graf cu gradele 1, 1, 3, 3, 3, 3, 5, 6, 8, 9?

Exercitii pentru seminarul din săptămâna următoare

Exercițiul 4.

Doi studenți, L(azy) și T(hinky), trebuie să găsească un drum particular între două noduri fixate într-un graf rar G : $|E(G)| = O(|V(G)|)$. L consideră că, deoarece graful este rar, numărul de drumuri dintre cele două noduri trebuie să fie mic, și o soluție lazy este să genereze (cu backtracking) toate aceste drumuri și apoi să rețină drumul dorit. T nu este de acord și dă următorul exemplu: fie $H = K_2 \times P_{n-1}$ ($n \geq 3$); adăugăm la H două noduri noi x și y fiecare unite prin câte două muchii cu câte una dintre cele două perechi de noduri adiacente de grad 2 din H .

Graful obținut, G , este rar dar numărul de drumuri dintre x și y este foarte mare. Explicați lui L acest exemplu desenând G , arătând că este rar și determinând numărul de drumuri dintre x și y .

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

Exercițiu 5. O sesiune de examene trebuie planificată folosind următoarele specificații: mulțimea examenelor este cunoscută; fiecare student trimită o listă a examenelor la care dorește să fie examinat; fiecare examen are loc cu toți studenții înscriși la examen (care este scris); fiecare student poate participa la cel mult un examen într-o aceeași zi.

Construiți un graf cu ajutorul căruia să raspundeți la următoarele întrebări (prin determinarea unor parametri corespunzători):

- (a) Care este numărul maxim de examene care pot fi organizate într-o același zi?
 - (b) Care este numărul minim de zile necesare organizării unei sesiuni?

Exercitii pentru seminarul din săptămâna următoare

Exercițiu 6. (exercițiu 5 - continuare)

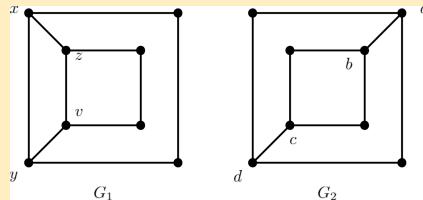
Un programator isteț și îndemânatic se întreabă dacă cele două probleme NP-hard din exercițiul anterior nu ar putea fi rezolvate în timp polinomial deoarece graful construit pare să aparțină unei clase speciale de grafuri.

- (a) Arătați că pentru orice graf dat, G , există un input pentru problema de planificare anterioară astfel încât graful construit (ca mai sus) să fie tocmai G .

Programatorul sugerează următoarea "abordare greedy" pentru a răspunde la a doua întrebare din exercițiul 4: începând cu prima zi, se planifică zilnic un număr maxim de examene (din mulțimea examenelor neplanificate încă), până când toate examenele vor fi planificate.

- b) Arătați că această abordare greedy este greșită printr-un contraexemplu.

Exercițiul 1. Grafurile de mai jos sunt izomorfe?



Indicație: Dacă grafurile sunt izomorfe atunci ele trebuie să aibă același număr de: noduri, muchii, componente conexe, circuite (induse) de o anumită lungime fixată etc.

Soluția 1.

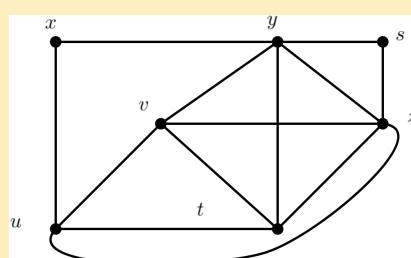
- Nodurile de grad 3 din graful G_1 formează (induc) în G_1 un circuit de lungime patru (fără corzi).
 - Ce formează nodurile de grad 3 din graful G_2 ?

Soluția 2.

- Câte circuite de lungime patru fără corzi există în graful G_1 și câte în graful G_2 ?

Exercitii rezolvate (partial)

Exercitiul 1. Să se determine numărul cromatic al grafului de mai jos:

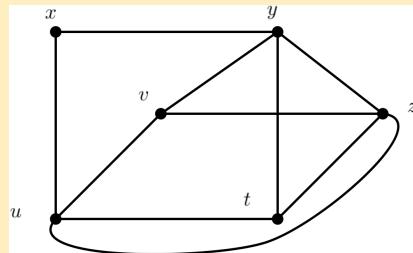


Soluție (și metodă).

- Se caută în graf un subgraf complet (o clică) de cardinal maxim (în cazul nostru $\{y, v, z, t\}$).
 - Se colorează nodurile acestui subgraf cu atâtea culori câte noduri conține ($c(y) = 1$, $c(v) = 2$, $c(z) = 3$, $c(t) = 4$).
 - Apoi se încearcă extinderea acestei colorări adăugând culori numai dacă este neapărată nevoie: $c(x)$ poate fi 2 etc

*C. Crișan - Graph Algorithms * C. Crișan - Graph Algorithms * C. Crișan - Graph Algorithms*

Exercițiu 1". Să se determine numărul de stabilitate al grafului de mai jos:



Soluție (și metodă).

- Există în graf multimi stabile de cardinal 2? Da: $\{u, y\}$ deci $\alpha(G) \geq 2$.
- Există în graf multimi stabile de cardinal 3? ... dacă răspunsul este afirmativ, atunci $\alpha(G) \geq 3$.
- Etc.

*C. Crișan - Graph Algorithms * C. Crișan - Graph Algorithms * C. Crișan - Graph Algorithms*

Exercițiu 2.

Fie $P(n) =$ "În orice graf cu cel puțin n noduri există trei noduri distincte care sunt două căte două adiacente sau două căte două neadiacente." Arătați că 6 este cea mai mică valoare a lui $n \in \mathbb{N}$, $n \geq$ pentru care $P(n)$ este adevărată.

Soluție.

- Pentru $3 \leq n \leq 5$ avem contraexemplul: P_3 (drumul cu 3 noduri, fără corzi) C_4 (circuitul cu 4 noduri, fără corzi) și, respectiv, C_5 (circuitul cu 5 noduri, fără corzi).
- Pentru $n \geq 6$, fie $v_0 \in V(G)$; $|N_G(v_0)| \geq 3$ or $|N_{\overline{G}}(v_0)| \geq 3$.
- Putem presupune că v_0 are cel puțin vecini în G (de ce?), v_1, v_2, v_3 , atunci acești vecini sunt fie doi căte doi ne-adiacenți (formează o mulțime stabilă) sau există o muchie $v_i v_j \in E(G)$ (v_0, v_i, v_j va fi o clică).

Exercițiu 3.

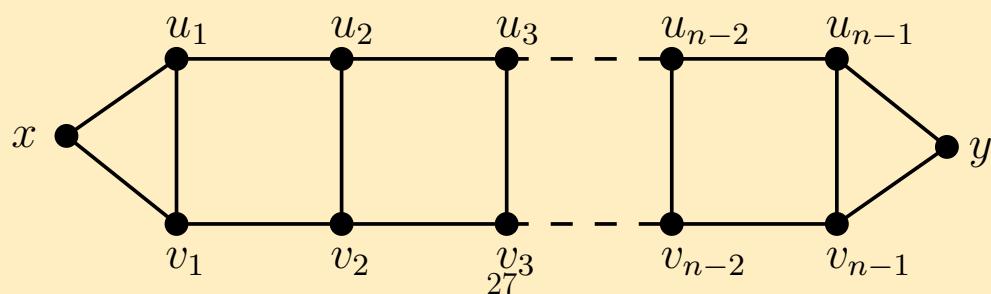
(c) Există un graf cu gradele 1, 1, 3, 3, 3, 3, 3, 5, 6, 8, 9?

Soluție.

- Se verifică mai întâi dacă suma gradelor este pară (de ce?). (Suma este pară.)
- Dacă ar exista un graf cu aceste grade ar avea 10 noduri, deci nodul de grad 9 este adiacent cu toate celelalte.
- Putem șterge din (ipoteticul) graf nodul de grad 9 și subgraful rămas ar avea gradele 0, 0, 2, 2, 2, 2, 4, 5, 7.
- Nodurile de grad 0 pot fi ignorate (de ce?). **Există un graf cu gradele 2, 2, 2, 2, 4, 5, 7?**

Exerciții rezolvate (partial)**Exercițiu 4. Soluție.**

- H este un graf de ordin $(2n - 2)$ (ladder graph - vezi figura 5) și $|G| = 2n$ și $|E(G)| = 3n - 1$, deci G este un graf rar: $|E(G)| = \mathcal{O}(|G|)$.
- Se poate demonstra prin inducție după n că numărul de xy -drumuri în G este 2^n (Cum?).
- Acest număr este exponențial în ordinul lui G (metoda backtracking nu este prea eficientă în acest caz).



Figure

Exercise 5. Solution

- Fie $G = (V, E)$ un graf, unde V e mulțimea examenelor. Pentru fiecare examen, v , se construiește lista tuturor studenților care susțin acest examen; atunci $uv \in E$ dacă și numai dacă listele lui u și v au studenți în comun.
- Presupunem că $S \subseteq V$ este mulțimea examenelor dintr-o anumită zi; pentru orice $u \neq v \in S$ nu putem avea $uv \in E$, deci S e o mulțime stabilă (independentă) de noduri din G . **Răspunsul la întrebarea (a) este ...**
- În acest fel, o sesiune corespunde unei partiții a mulțimii V în mulțimi stable. Numărul minim de clase într-o asemenea partiție este răspunsul la cea de-a doua întrebare**acest număr este ...**

Exerciții rezolvate (partial)

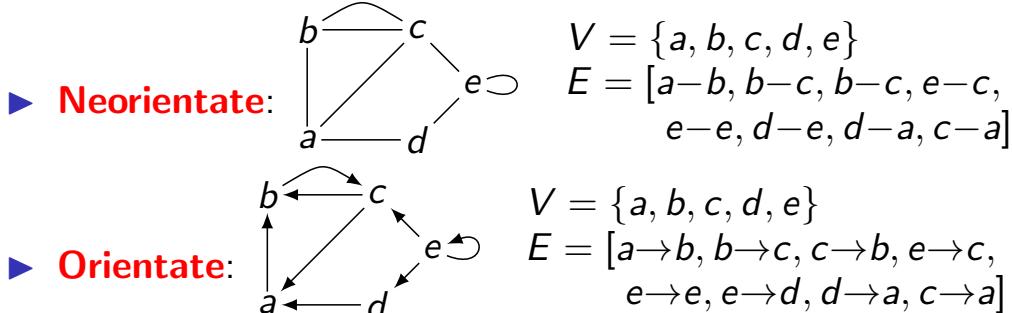
Exercise 6. Solution

- (a) Fie $G = (V, E)$ un graf; definim V ca fiind mulțimea examenelor iar E mulțimea studenților: studentul $uv \in E$ va fi "înscris" doar la examenele u și v .
- (b) Fie $G = P_6$, $V(G) = \{1, 2, \dots, 6\}$ cu muchiile $(i, i + 1) \pmod 6$ - drumul cu 6 noduri fără corzi, $\alpha(P_6) = 3$, $\chi(P_6) = 2$ - este un graf bipartit.

Dacă primei zile îi asociem mulțimea stabilă $\{1, 4, 6\}$, atunci G se va colora cu 3 culori.

Graf = structură de date pentru modelarea relațiilor binare dintre componentele unui sistem.

- $G = (V, E)$ unde
 - V : mulțime de **noduri** = componentele sistemului
 - E : listă de **muchii** = legături/relații dintre noduri
- Tipuri de grafuri:



Muchiile orientate se numesc și **arce** (singular: **arc**).

OBSERVAȚIE: Dacă $a \neq b$ atunci:

- În grafuri neorientate: $a-b = b-a$ (orientarea nu contează)
- În grafuri orientate: $a \rightarrow b \neq b \rightarrow a$ (orientarea contează!)

Vocabularul teoriei grafurilor

Terminologie

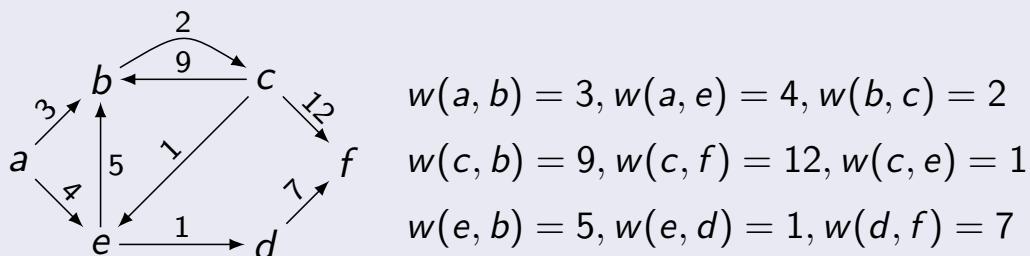
- Orice muchie este **incidentă** la 2 noduri (capetele muchiei).
- Un arc $a \rightarrow b$ are **sursa** a și **destinația** b .
- Muchiile de forma $a-a$ sau $a \rightarrow a$ se numesc **bucle**.
- Aparițiile multiple ale unei muchii în E se numesc **muchii paralele**. Exemplu: $b-c$ în
 $E = [a-b, b-c, b-c, e-c, e-e, d-e, d-a, c-a]$
- Un graf orientat se numește și **digraf** (engl. *directed graph*).
- Un **pseudograf** este un graf cu bucle și muchii paralele.
- Un **multigraf** este un graf fără bucle și cu muchii paralele.
- Un **graf simplu** este un graf fără bucle și fără muchii paralele.
 - ⇒ dacă $G = (V, E)$ este graf simplu atunci $E \subseteq V \times V$ și scriem $a \rightarrow b$ în loc de $\langle a, b \rangle \in E$.
 - ⇒ **cel mai adesea, vom studia grafuri simple.**

Graf ponderat (G, w) unde

- $G = (V, E)$ este un graf
- $w : E \rightarrow \mathbb{R}$
unde $w(e)$ este **ponderea** sau **greutatea** muchiei $e \in E$.

Dacă G este graf simplu iar $e = x-y$ sau $e = x \rightarrow y$, atunci scriem $w(x, y)$ în loc de $w(e)$.

Exemplu



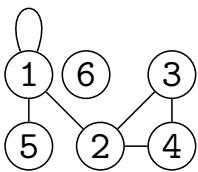
Vocabularul teoriei grafurilor

- **Ordinul** unui graf = numărul de noduri
- **Mărimea** unui graf = numărul de muchii
- **Mulțimea de vecini** a unui nod $x \in V$:
 - $V(x) = \{y \mid x-y \in E\}$ dacă graful este neorientat
 - $V(x) = \{y \mid x \rightarrow y \in E\}$ dacă graful este orientat
 Dacă $S \subseteq V$ atunci $V(S) = \bigcup_{x \in S} V(x)$.
- $V[x] = V(x) \cup \{x\}$
- x este **nod terminal** dacă are un singur vecin: $|V(x)| = 1$
- x este **nod izolat** dacă nu are vecini: $|V(x)| = 0$
- Dacă G este graf neorientat:
 - **gradul** lui x este $\deg(x) = |\{y \mid x-y \in E\}| + |\{y \mid y-x \in E\}|$
 - **secvența de grade** a lui G este lista gradelor nodurilor din G , în ordine descrescătoare.
- Dacă G este digraf:
 - **gradul interior** al lui x este $\deg^-(x) = |\{y \mid y \rightarrow x \in E\}|$
 - **gradul exterior** al lui x este $\deg^+(x) = |\{y \mid x \rightarrow y \in E\}|$

Vocabularul teoriei grafurilor

Exemplu

[Jump to Table of contents](#)



Mărimea: 6

Ordinul: 6

v	$V(v)$	$V[v]$	$\deg(v)$
1	{1, 2, 5}	{1, 2, 5}	4
2	{1, 3, 4}	{1, 2, 3, 4}	3
3	{2, 4}	{2, 3, 4}	2
4	{2, 3}	{2, 3, 4}	2
5	{1}	{1, 5}	1
6	\emptyset	{6}	0

Secvența de grade este [4, 3, 2, 2, 1, 0].

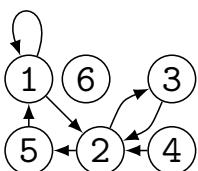
Nodul 5 este terminal. Nodul 6 este izolat.



Cursul 8

Vocabularul teoriei grafurilor

Exemplu



Mărimea: 6

Ordinul: 7

v	$V(v)$	$V[v]$	$\deg^-(v)$	$\deg^+(v)$	$\deg(v)$
1	{1, 2, 5}	{1, 2, 5}	2	2	4
2	{1, 3, 4, 5}	{1, 2, 3, 4, 5}	3	2	5
3	{2, 4}	{2, 3, 4}	1	1	2
4	{2, 3}	{2, 3, 4}	0	1	1
5	{1, 3}	{1, 3, 5}	1	1	2
6	\emptyset	{6}	0	0	0

Secvența de grade este lista [5, 4, 2, 2, 1, 0].



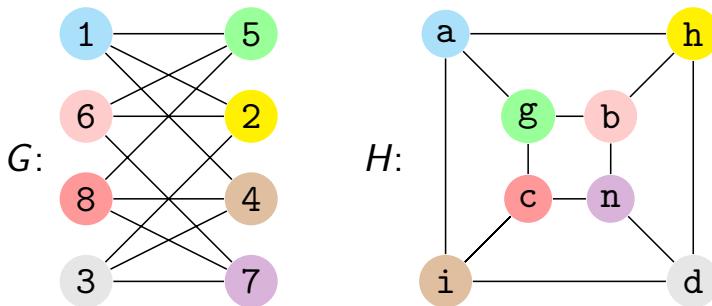
Cursul 8

Grafurile simple $G = (V, E)$ și $H = (V', E')$ sunt **izomorfe**, și scriem $G \simeq H$, dacă există o funcție bijectivă $f : V \rightarrow V'$ astfel încât

- $x-y \in E$ dacă și numai dacă $f(x)-f(y) \in E'$
 - $x \rightarrow y \in E$ dacă și numai dacă $f(x) \rightarrow f(y) \in E'$

REMARĂ. Două grafuri sunt izomorfe dacă putem redenumi nodurile primului graf cu numele nodurilor celui de-al doilea graf în aşa fel încât muchiile dintre noduri să rămână aceleasi.

De exemplu $G \cong H$ unde



Vocabularul teoriei grafurilor

Conectivitate (1)

Dacă $G = (V, E)$ este graf neorientat atunci

- Un **drum** sau **cale** de la x la y în G este o listă de noduri $\pi = [x_1, x_2, \dots, x_n]$ astfel încât $n \geq 2$, $x_1 = x$, $x_n = y$ și $x_i - x_{i+1} \in E$ pentru $1 \leq i < n$.
 - Lungimea drumului este $n - 1$.

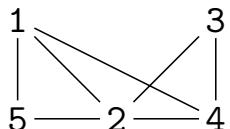
Dacă $G = (V, E)$ este graf orientat atunci

- Un **drum** sau **cale** de la x la y în G este o listă de noduri $\pi = [x_1, x_2, \dots, x_n]$ astfel încât $n \geq 2$, $x_1 = x$, $x_n = y$ și $x_i \rightarrow x_{i+1} \in E$ pentru $1 \leq i < n$.
 - Lungimea drumului este $n - 1$.

- Scriem $x \rightsquigarrow y$ dacă există drum de la x la y , și $x \xrightarrow{\pi} y$ dacă π este un drum de la x la y .
- Un drum este
 - elementar** dacă nu conține de mai multe ori același nod,
 - hamiltonian** dacă este elementar și conține toate nodurile grafului.
 - simplu** dacă nu conține de mai multe ori aceeași muchie,
 - eulerian** dacă este simplu și conține toate muchiile grafului.
- Un **ciclu** este un drum simplu de la un nod x la x .
- Un ciclu $[x_1, \dots, x_n, x_1]$ este
 - hamiltonian** dacă $[x_1, \dots, x_n]$ este drum eulerian.
 - eulerian** dacă este drum eulerian.

Vocabularul teoriei grafurilor

Conecțivitate



$[5, 1, 2, 3, 4, 2]$ este un drum simplu dar neelementar de lungime 5.

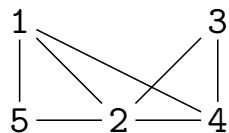
$[5, 1, 2, 4]$ și $[5, 1, 2, 3]$ sunt drumuri elementare de lungime 3.

$[2, 1, 5, 2, 3, 4, 2]$ este un ciclu neelementar de lungime 6.

$[2, 3, 4, 2]$ este un ciclu elementar de lungime 3.

$[1, 2, 5, 1, 4, 3, 2, 4]$ este un drum eulerian.

$[2, 5, 1, 4, 3, 2]$ este un ciclu hamiltonian.



[5, 1, 2, 3, 4, 2] este un drum simplu dar neelementar de lungime 5.

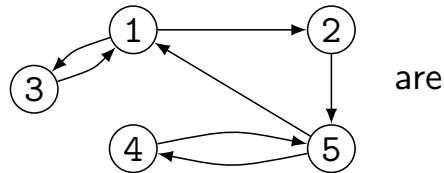
$[5, 1, 2, 4]$ și $[5, 1, 2, 3]$ sunt drumuri elementare de lungime 3.

[2, 1, 5, 2, 3, 4, 2] este un ciclu neelementar de lungime 6.

$[2, 3, 4, 2]$ este un ciclu elementar de lungime 3.

[1, 2, 5, 1, 4, 3, 2, 4] este un drum eulerian.

[2, 5, 1, 4, 3, 2] este un ciclu hamiltonian.



drumul hamiltonian [3, 1, 2, 5, 4] și ciclul eulerian [5, 4, 5, 1, 3, 1, 2, 5].



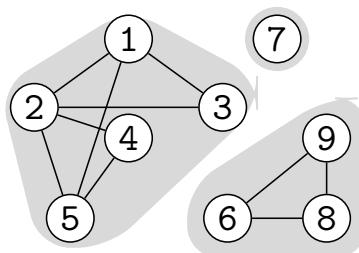
Vocabularul teoriei grafurilor

Componente conexa

Componentele conexe sunt definite pentru grafuri neorientate $G = (V, E)$.

- Relația de conectivitate “ \rightsquigarrow ” este o relație de echivalență
 - clasele de echivalență ale lui \rightsquigarrow se numesc **componente conexe** ale lui G

EXEMPLU.



are 3 componente conexe: $\{1, 2, 3, 4, 5\}$, $\{6, 8, 9\}$ și $\{7\}$.



Componentele tare conexe, sau tari, sunt definite pentru grafuri neorientate $G = (V, E)$.

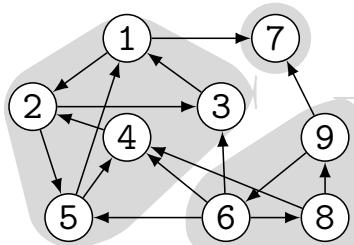
- Conectivitatea tare

$$x \sim_{ct} y \Leftrightarrow x \rightsquigarrow y \text{ și } y \rightsquigarrow x$$

este o relație de echivalentă. Clasele de echivalentă ale lui \sim_{ct} se numesc **componente tari** ale lui G .

- G este **tare conex** dacă are o singură componentă tare, adică dacă oricare două noduri din V sunt conectate tare.

EXEMPLU.



are 3 componente tari: $\{1, 2, 3, 4, 5\}$, $\{6, 8, 9\}$ și $\{7\}$.

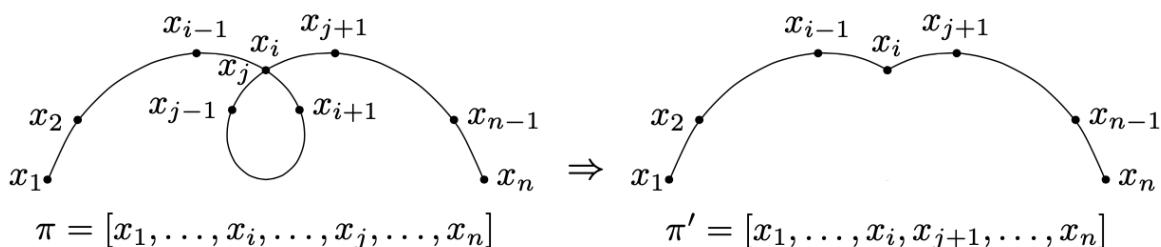
Teoreme fundamentale ale Teoriei Grafurilor

Teorema 1. Într-un graf, suma gradelor nodurilor este egală cu dublul numărului de muchii.

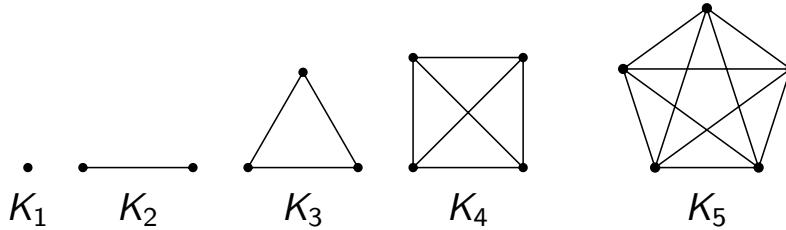
$$\sum_{x \in V(G)} \deg(x) = 2 \cdot |E(G)|.$$

Teorema 2. Orice drum de la x la y conține un drum elementar de la x la y .

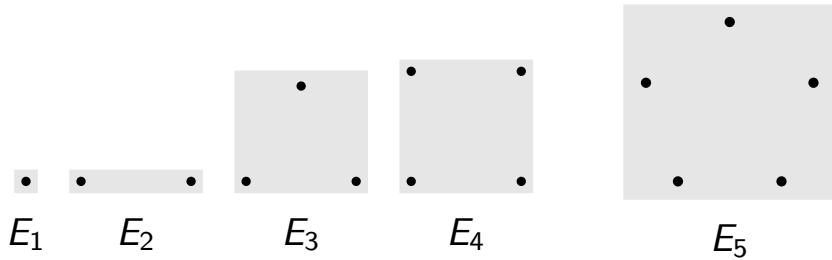
SCHITĂ DE DEMONSTRATIE:



- **Graful complet** K_n ($n \geq 1$) are $V = \{1, 2, \dots, n\}$ și $E = \{i-j \mid 1 \leq i < j \leq n\}$.

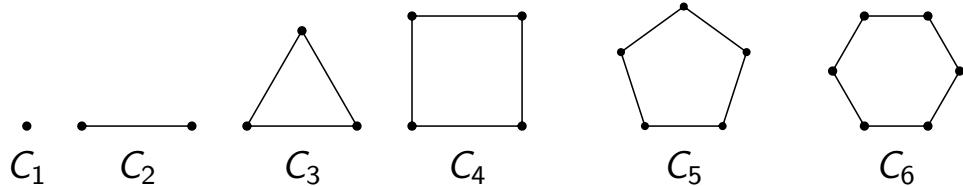


- **Graful nul** E_n ($n \geq 1$) are $V = \{1, 2, \dots, n\}$ și $E = \emptyset$.

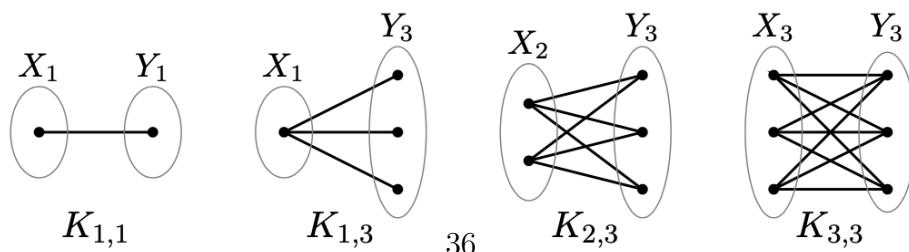


Clase de grafuri

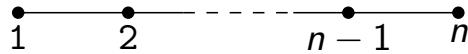
- **Graful ciclic** C_n ($n \geq 1$) are $V = \{1, 2, \dots, n\}$ și $E = \{i-j \mid 1 \leq i \leq n, j = 1 + (i \text{ mod } n)\}$.



- Pentru $m, n > 0$, **graful bipartit complet** $K_{m,n}$ are $V = \{x_i \mid 1 \leq i \leq m\} \cup \{y_j \mid 1 \leq j \leq n\}$ și $E = \{x_i-y_j \mid 1 \leq i \leq m, 1 \leq j \leq n\}$



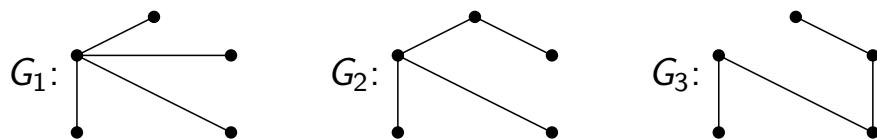
- **Calea** P_n este graful cu $V = \{1, 2, \dots, n\}$ și $E = \{i-j \mid 1 \leq i < n, j = i + 1\}$.



- Un **arbore de ordinul n** este un graf simplu conex cu n noduri și fără cicluri. Multimea acestor arbori se notează cu T_n .

REMARCA. Toți arborii din T_n au n noduri și $n - 1$ muchii.

EXEMPLU. Arboi din clasa T_5 :



Subgraf, subgraf parțial

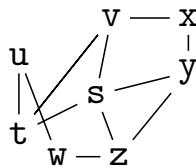
Fie $G = (V, E)$ un graf și $S \subseteq V$.

- **Subgraful induș** de S în G este graful $G' = (S, E')$ unde $E' = \{e \in E \mid \text{capetele lui } e \text{ sunt în } S\}$.
- G' este **subgraf** al lui G dacă $G' = G[S]$ pentru un $S \subseteq V$. În acest caz, mai spunem și că G' este **conținut** în G .
- $G' = (V', E')$ este **subgraf parțial** al lui G dacă $V' \subseteq V$ și $E' \subseteq E$.

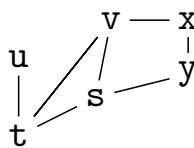
Fie $G = (V, E)$ un graf și $S \subseteq V$.

- **Subgraful induș** de S în G este graful $G' = (S, E')$ unde $E' = \{e \in E \mid \text{capetele lui } e \text{ sunt în } S\}$.
- G' este **subgraf** al lui G dacă $G' = G[S]$ pentru un $S \subseteq V$. În acest caz, mai spunem și că G' este **conținut** în G .
- $G' = (V', E')$ este **subgraf parțial** al lui G dacă $V' \subseteq V$ și $E' \subseteq E$.

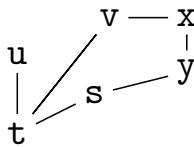
EXEMPLU. Fie G graful



- Subgraf:



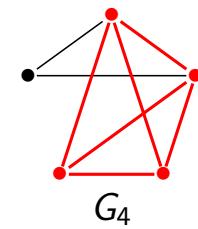
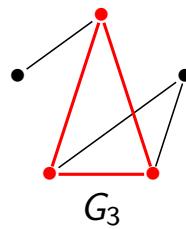
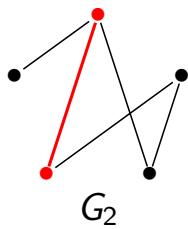
- Subgraf parțial:



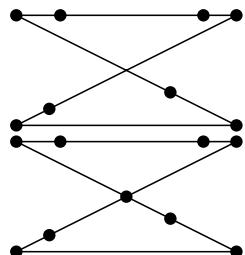
Vocabularul teoriei grafurilor

Clică, subdiviziune

- O **n-clică** (sau doar **clică**) a unui graf neorientat G este un subgraf al lui G izomorf cu K_n . De exemplu:



- O **subdiviziune** a unui graf neorientat G este un graf obținut prin una sau mai multe inserări succesive de noduri noi pe muchiile lui G . De exemplu:



este subdiviziune a lui $K_{2,2}$.

nu este subdiviziune a lui $K_{2,2}$.

Fie $G = (V, E)$ un graf, $x \in V$, $S \subseteq V$ și $T \subseteq E$.

[Jump to Table of contents](#)

- $G - S$ este subgraful induș $G[V - S]$
- $G - T$ este subgraful parțial $(V, E - T)$
- $G \setminus S$ este graful (V', E') obținut din contracția nodurilor din S în un singur nod nou x_S :
 - $V' = (V - S) \cup \{x_S\}$
 - Dacă G este neorientat atunci

$$E' = \{x-y \mid x, y \in V - S \text{ și } x-y \in E\} \cup \\ \{x-x_S \mid \text{există } x-y \in E \text{ cu } x \in V - S \text{ și } y \in S\}.$$

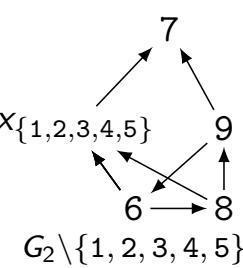
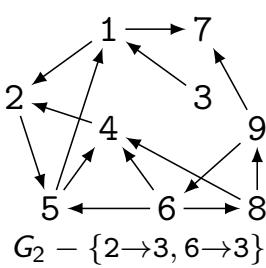
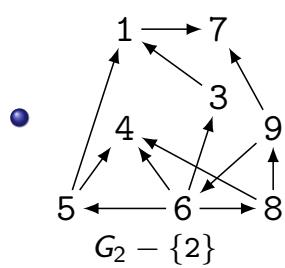
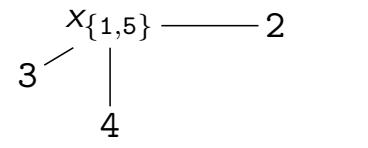
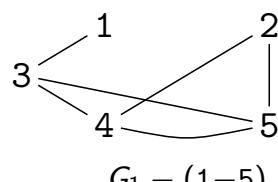
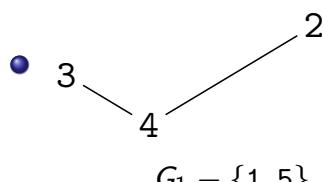
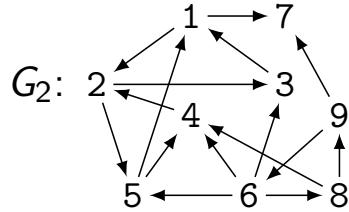
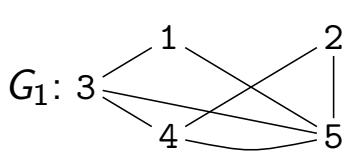
Dacă G este digraf atunci

$$E' = \{x \rightarrow y \mid x, y \in V(G) - S \text{ și } x \rightarrow y \in E\} \cup \\ \{x \rightarrow x_S \mid \text{există } x \rightarrow y \in E \text{ cu } x \in V - S \text{ și } y \in S\} \cup \\ \{x_S \rightarrow y \mid \text{există } x \rightarrow y \in E \text{ cu } x \in S \text{ și } y \in V - S\}.$$

- $G \setminus e$ este graful $G \setminus S$ unde S sunt capetele muchiei e .

Operații cu grafuri

Exemple

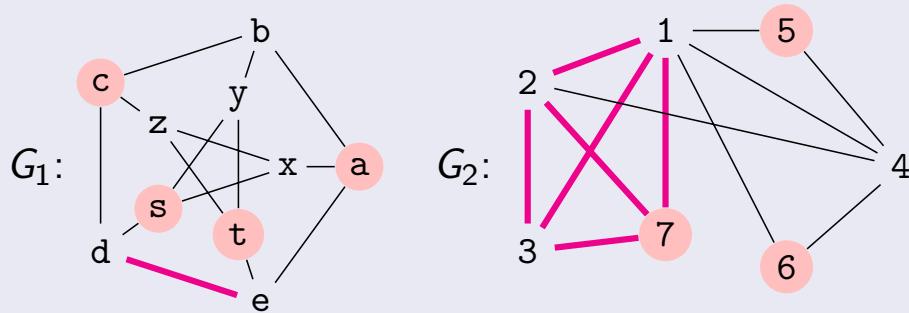


Fie $G = (V, E)$ un graf neorientat și $S \subseteq V$.

- **Gradul minim** al lui G este $\delta(G) = \min\{\deg(x) \mid x \in V\}$
- **Gradul maxim** al lui G este $\Delta(G) = \max\{\deg(x) \mid x \in V\}$
- Multimea de noduri S este **stabilă** sau **independentă** dacă nu există nici o muchie între noduri din S .
- **Numărul de independentă** al lui G este $\alpha(G) = \max\{|S| \mid S \text{ este multime independentă în } G\}$.
- **Numărul de clică** $\omega(G)$ al lui G este ordinul maxim al unei cíci din G , adică $\omega(G) = \max\{n \mid H \text{ este } n\text{-clică a lui } G\}$.

Vocabularul teoriei grafurilor

Exemple



- $\delta(G_1) = \Delta(G_1) = 3$, $\alpha(G_1) = 4$, $\omega(G_1) = 2$.
- $\delta(G_2) = 2$, $\Delta(G_2) = 6$, $\alpha(G_2) = 3$, $\omega(G_2) = 4$.

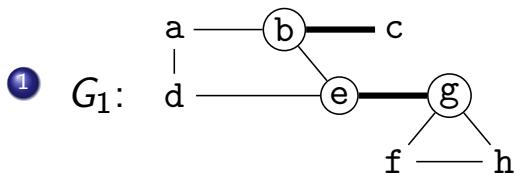
Fie $G = (V, E)$ un graf neorientat conex.

- $x \in V(G)$ este **nod de articulație** (engl. **cut vertex**) dacă $G - \{x\}$ nu este conex. Altfel spus, ștergerea lui x distrugе conectivitatea lui G .
- $e \in E(G)$ este o **punte** (engl. **bridge**) dacă $G - e$ nu este conex. Altfel spus, ștergerea muchiei e distrugе conectivitatea lui G .
- $S \subsetneq V(G)$ este o **mulțime de articulație** (engl. **vertex cut set**) a lui G dacă graful $G - S$ nu este conex. Altfel spus, ștergerea nodurilor din S distrugе conectivitatea lui G .

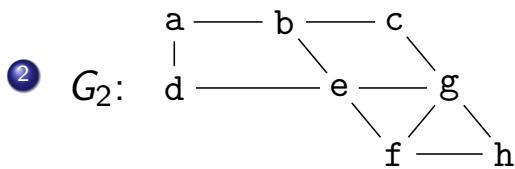
Vocabularul teoriei grafurilor

grad de conectivitate, distanță, excentricitate, centru, periferie, rază, diametru

- **Gradul de conectivitate** $\kappa(G)$ al unui graf incomplet G este numărul minim de noduri ce trebuie eliminate din G pentru a-l deconecta, adică $\kappa(G) = \min\{|S| \mid S \text{ este mulțime de articulație a lui } G\}$. Dacă k este un întreg strict pozitiv, spunem că G este **k -conex** dacă $k \leq \kappa(G)$.
- **Distanța** $d(x, y)$ de la x la y este cea mai mică lungime a unui drum de la x la y .
- **Excentricitatea** $e(x)$ a nodului x este distanța cea mai mare de la x la un nod în G , adică $e(x) = \max\{d(x, y) \mid y \in V\}$.
- **Centrul** lui G este mulțimea nodurilor cu excentricitate minimă.
- **Periferia** lui G este mulțimea nodurilor cu excentricitate maximă.
- **Raza** lui G este excentricitatea unui nod din centrul lui G , adică $radius(G) = \min\{e(x) \mid x \in V\}$.
- **Diametrul** lui G este excentricitatea unui nod de la periferia lui G , adică $diam(G) = \max\{e(x) \mid x \in V\}$.

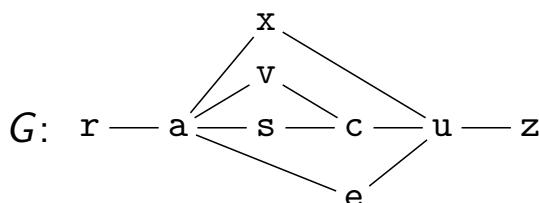


are 3 puncte de articulație (cele încercuite) și 2 punți (cele îngroșate).



nu are puncte de articulație și nici punți, dar are multimi de articulație cu 2 noduri, de exemplu $\{c, e\}$, $\{f, g\}$ sau $\{b, e\}$. Deci $\kappa(G_2) = 2$.

Vocabularul teoriei grafurilor



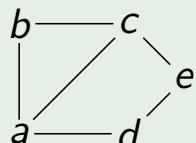
- $d(x, c) = 2$, $d(r, z) = 4$,
- $e(x) = e(e) = 2$,
- $e(z) = e(z) = 4$,
- $radius(G) = 2$,
- $diam(G) = 4$,
- G are centrul $\{x, e\}$,
- G are periferia $\{r, z\}$.

- 1 Listă de noduri + listă de muchii
- 2 Liste de adiacență
- 3 Matrice de adiacență
- 4 Matrice de incidentă
- 5 Matrice de ponderi

Reprezentarea grafurilor

1. Cu listă de noduri + listă de muchii

Exemplu

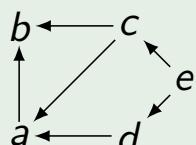


Listă de noduri $V = [a, b, c, d, e]$

Listă de muchii $E = [a-b, a-d, b-c, c-e, d-e]$

Observații: $a-b = b-a$, $a-c = c-a$, etc.

muchia $a-b \leftrightarrow$ multimea $\{a, b\}$



Listă de noduri $V = [a, b, c, d, e]$

Listă de arce $E = [a \rightarrow b, c \rightarrow a, c \rightarrow b, d \rightarrow a, e \rightarrow c, e \rightarrow d]$

Observații: $a \rightarrow b \neq b \rightarrow a$, $a \rightarrow c \neq c \rightarrow a$, etc.

arcul $a \rightarrow b \leftrightarrow$ perechea $\langle a, b \rangle$

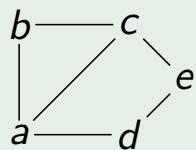
Remarcă

Dacă nu există noduri izolate, nu este necesar să fie reținută lista de noduri V :

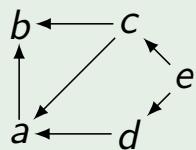
- ▶ V se poate calcula din E

Pentru fiecare nod $u \in V$ se reține lista de vecini a lui u .

Exemplu



$$\begin{aligned}\text{adj}[a] &= [b, c, d] & \text{adj}[d] &= [a, e] \\ \text{adj}[b] &= [a, c] & \text{adj}[e] &= [c, d] \\ \text{adj}[c] &= [a, b, e]\end{aligned}$$



$$\begin{aligned}\text{adj}[a] &= [b] & \text{adj}[d] &= [a] \\ \text{adj}[b] &= [] & \text{adj}[e] &= [c, d] \\ \text{adj}[c] &= [a, b]\end{aligned}$$

Reprezentarea grafurilor

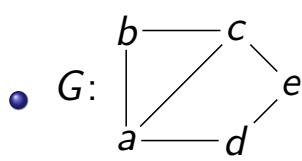
3. Cu matrice de adiacență

Presupunem că $G = (V, E)$ este un graf cu n noduri.

- ① Fixăm o enumerare $[x_1, x_2, \dots, x_n]$ a nodurilor din V .
- ② Matricea de adiacență este $A_G = (a_{ij})$ de dimensiune $n \times n$ cu $a_{ij} :=$ numărul de muchii de la nodul x_i la nodul x_j .

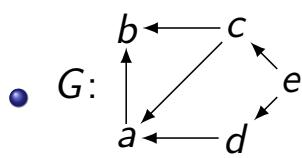
Observații

- ① Înainte de a construi A_G din G , trebuie fixată o enumerare a tuturor nodurilor: $[x_1, x_2, \dots, x_n]$
- ② Dacă G este neorientat, A_G este matrice simetrică
- ③ Dacă G este graf simplu, A_G conține doar 0 și 1



Enumerarea nodurilor: $[a, b, c, d, e]$

$$A_G = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$



Enumerarea nodurilor: $[a, b, c, d, e]$

$$A_G = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$

Reprezentarea digrafurilor

4. Cu matrice de incidență

Presupunem că $G = (V, E)$ este digraf.

- ① Fixăm o enumerare $V = [v_1, \dots, v_n]$ a nodurilor din V
- ② Fixăm o enumerare $E = [e_1, \dots, e_p]$ a muchiilor din E

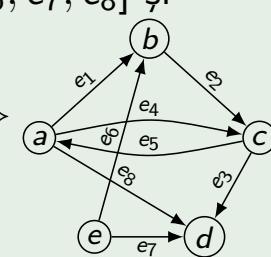
Matricea de incidență $M_G = (m_{ij})$ are dimensiunea $n \times p$ și

$$m_{ij} = \begin{cases} 1 & \text{dacă } e_j \text{ are sursa } v_i; \\ -1 & \text{dacă } e_j \text{ are destinația } v_i; \\ 0 & \text{în toate celelalte cazuri.} \end{cases}$$

Exemplu

Dacă $V = [a, b, c, d, e]$, $E = [e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8]$ și

$$M_G = \begin{pmatrix} 1 & 0 & 0 & 1 & -1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 1 & -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \end{pmatrix}$$



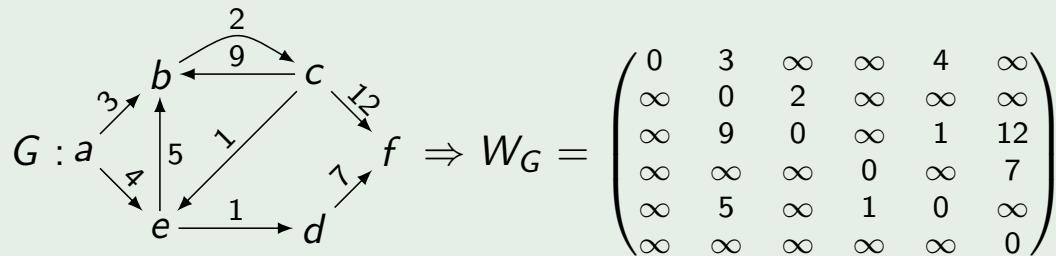
Presupunem că $G = ((V, E), w)$ este un graf simplu ponderat.

- Fixăm o enumerare $[x_1, x_2, \dots, x_n]$ a nodurilor din V .

Matricea de ponderi $W_G = (w_{ij})$ a lui (G, w) este matricea de dimensiune $n \times n$ cu

$$w_{ij} = \begin{cases} 0 & \text{dacă } i = j, \\ w(x_i, x_j) & \text{dacă } x_i - x_j \in E \text{ sau } x_i \rightarrow x_j \in E, \\ +\infty & \text{în celelalte cazuri.} \end{cases}$$

Exemplu (Digraf ponderat cu enumerare de noduri $[a, b, c, d, e, f]$)



Reprezentarea grafurilor

Studiu comparativ pentru un graf cu n noduri și m muchii

- ▶ Reprezentarea cu **listă de muchii**
 - Adekvată pentru reprezentarea grafurilor simple fără noduri izolate, cu $m \ll n^2$
 - Complexitate spațială (memorie ocupată): $O(m)$
 - ▶ Reprezentarea cu **liste de adiacență**
 - Permite enumerarea rapidă a vecinilor unui nod
 - Complexitate spațială (memorie ocupată): $O(n + m)$
 - ▶ Reprezentarea cu **matrice de adiacență** $A_G = (a_{ij})$ sau cu **matrice de ponderi** $W_G = (w_{ij})$
 - Test rapid de conectivitate directă între 2 noduri: $O(1)$
 - $\nexists(v_i, v_j) \in E$ dacă $a_{ij} = 0$ sau dacă $w_{ij} = \infty$
 - Complexitate spațială (memorie ocupată): $O(n^2)$
 - reprezentare neadecvată când $m \ll n^2$
 - Reprezentarea cu **matrice de incidentă** M_G
 - ▶ Complexitate temporală: $O(n \cdot m)$

2 CURS 2: Vocabularul Teoriei grafurilor.

- un graf complet (de forma K_n) va avea mereu diametrul egal cu 1
- complementul unui digraf complet o sa fie mereu un graf cu 2 componente conexe ce reprezinta fiecare cate un graf complet

2. *Diametrul unui graf este lungimea maximă a unui drum de lungime minimă între două vârfuri ale grafului. Două vârfuri care sunt extremitățile unui drum minim de lungime maximă în graf se numesc diametral opuse. Demonstrați că următorul algoritm determină o pereche de vârfuri diametral opuse într-un arbore T :*

- *Dintr-un vârf oarecare se execută o parcurgere BFS a arborelui; fie u ultimul vârf vizitat;*
- *Din vârful u se executa o parcurgere BFS a arborelui; fie v ultimul vârf vizitat;*
- *Return u, v.*

Rămâne valabil algoritmul pentru un graf conex arecare?

Multigraf: ai un graf cu ≥ 1 muchii intre 2 noduri

Graf suprat: graful normal al celui multi.

Digraf: graf orientat (neinteresant)

Turacu: graf orientat complet

Grad maxim: $\Delta(G) = \max_{v \in V} d_G(v)$

$\delta(G) = \min_{v \in V} d_G(v)$

if $\Delta(G) = \delta(G) = k \Rightarrow G$ este k -regulat

Subgraf al lui G : un graf H cu $V(H) \subseteq V(G)$
 $E(H) \subseteq E(G)$

Graf parțial al lui G : un graf H al lui G aî

Subgraf induș al lui G : un subgraf H aî

$$E(H) = \binom{V(H)}{2} \cap E(G)$$

Complementul unui graf G : ~~nu~~ un alt graf
- cu aceleasi noduri
- dar cu alte muchii

K_n -graf complet cu n noduri
mai exact cele pe care
trebuie să le mai pun să
faci un graf complet

Lime graf: modurile de transformi o muchie

K-clică : o submultime de K muchii din G

care induce un graf complet

nr de clică: $\omega(G) = \max_{Q \text{ ciclă în } G} |Q|$

Graf bipartit complet: $K_{s,t}$

Graf planar: un graf care poate fi reprezentat în plan, muchiile se intersecțează doar în extremități

Mers de la u la v de lg r: secuie de noduri și muchii

(v₀, v₀v₁, v₁, ..., v_{r-1}, v_{r-1}v_r, v_r)

Parcurs: mers cu muchii distințe

Drum: mers cu noduri distințe

M: v-t-w-v-x-t-v-w

P: w-t-v-x-t-y-u-v

D: u-y-x-v-w-s

Diametrul unui graf: Fără ATENȚIE

$d(G)$ = maximul lungimii minimale
de la u la v

Graf conex : ai un drum între oricare 2 noduri
Jump to Table of contents

Digraf slab conex : graful ^{sau} _{suport} este conex

Digraf unilateral conex : ai un drum de la u la v

sau

v la u, $\forall u, v \in V$

Digraf fără conex : ai un drum de la u la v,
 $\exists u, v \in V$

Punct de articulație : un nod $v \in V$ al $G - v$ este neconex

Mult de art. :

$p \in N^*$, G p-conex dacă

$|V| = p$ și $G = K_p$ sau

$|V| \geq p + 1$ și G nu are mult de articulație de card $< p$ (G nu poate fi deconectat prin ștergerea a mai puțin de p noduri)

nr de conexiune pe noduri : $K(G) = \max \{ p \in N^* : G \text{ este p-conex} \}$

Punte : o mulțime $e \in E$ al $G - e$ nu este conex
graf p-mulțime-conex dacă G nu poate fi deconectat prin ștergerea a mai puțin de p mulțimi.

Nr de conexiune pe mulțimi : $\lambda(G) = \max \{ p \in N^* : G \text{ este p-mulțime-conex} \}$

G-eulerian - parours inclus sont trece prime
Jump to Table of contents
flécage nul

G-hamiltonien - u mod

Algoritmica grafurilor - Cursul 2

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

Cuprins

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

1 Vocabularul teoriei grafurilor

- **Variatii în definitia unui graf**
- **Grade**
- **Subgrafuri**
- **Operații cu grafuri**
- **Clase de grafuri**
- **Drumuri și circuite**

2 Exerciții pentru seminarul de săptămâna viitoare

3 Exerciții rezolvate (parțial)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Multigraf: $G = (V, E)$, unde V este o mulțime nevidă (de noduri), și E
este un **multiset** (de muchii) pe V , i. e., există o funcție $m : \binom{V}{2} \rightarrow \mathbb{N}$.

$e \in \binom{V}{2}$, cu $m(e) > 0$ este o muchie a multigrafului G ; dacă $m(e) = 1$,
atunci e este o **muchie simplă**, altfel este o **muchie multiplă** cu multi-
plicitatea $m(e)$.

Graful suport al unui graf, G , este graful obținut din G prin înlocuirea
fiecărei muchii multiple printr-una simplă.

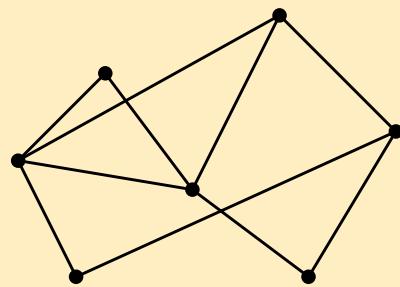
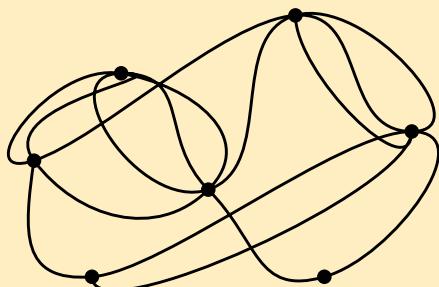
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Variatii în definiția unui graf

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

Exemplu

Un multigraf și graful său suport:



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

Pseudograf (graf general): $G = (V, E)$, unde V este o mulțime (de noduri), și E este un multiset (de muchii) peste $V \cup \binom{V}{2}$, i. e., există o funcție $m : V \cup \binom{V}{2} \rightarrow \mathbb{N}$.

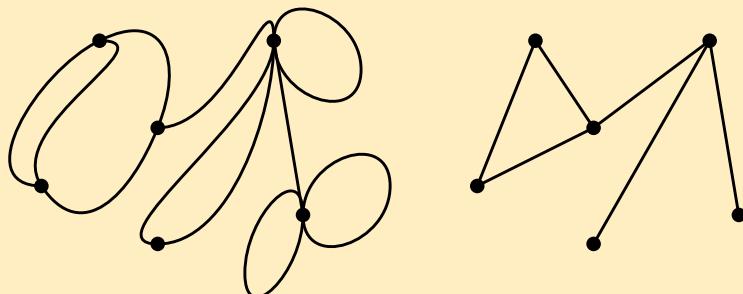
$e \in E \cap V$ (i. e., $|e| = 1$) este numită **bucă**.

Graful suport al unui pseudograaf G este graful obținut din G prin înlocuirea fiecărei muchii multiple printr-una simplă și prin ștergerea buclelor.

Variatii în definiția unui graf

Exemplu

Un pseudograf și graful său suport:



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

Digraf: $D = (V(D), E(D))$, unde $V(D)$ este o mulțime (de noduri), și $E(D) \subseteq V(D) \times V(D)$ este o mulțime de arce (sau muchii orientate).

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

Dacă $e \in E$ atunci $e = (u, v)$ (sau simplu $e = uv$) este un arc orientat de la u către v și spunem că:

- u este extremitatea initială of e , v este extremitatea finală ale lui e ;
- u și v sunt adiacente;
- e este incident din u și către v ;
- v este a sucesor al lui u și u este a predecesor al lui v etc.

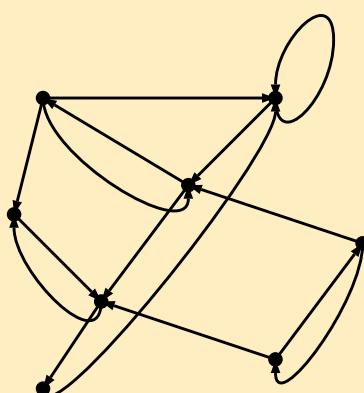
Graph Algorithms * C. Croitoru - Graph Algorithms *

Variatii în definiția unui graf

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

Un digraf:



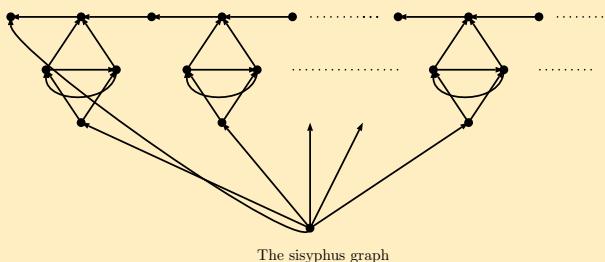
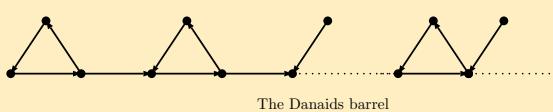
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

- **Pereche simetrică de arce:** (uv, vu) . uv este numit inversul lui vu .
- **Inversul** unui digraf D : se înlocuiește fiecare arc din D cu inversul său.
- **Graful suport** al unui digraf D , $M(D)$, se înlocuiește fiecare arc din D cu mulțimea corespunzătoare de două noduri. $M(D)$ este un multigraf.
- Dacă $M(D)$ este un graf (simplu), atunci D este numit **graf orientat**.
- **Digraf complet simetric**: orice două noduri (distingute) sunt unite printr-o pereche simetrică de arce.
- **Turneu**: un graf orientat complet (orice două noduri (distingute) sunt unite exact printr-un arc).

Variatii în definiția unui graf

(Di)grafuri infinite: mulțimea nodurilor și/sau mulțimea muchiilor (arcelor) este numărabil infinită.

Un graf infinit este **local finit** dacă $N(v)$ este o mulțime finită, pentru orice nod v .

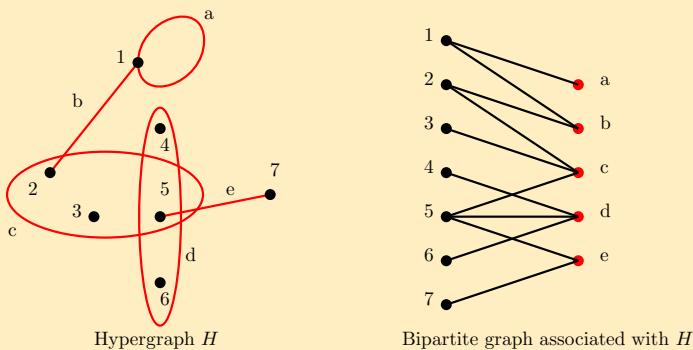


Hipergrafuri (Sisteme de mulțimi finite)

- Muchiile, numite acum **hipermuchiile**, nu mai sunt restricționate să fie submulțimi cu două elemente ale mulțimii de noduri. O hipermuchie este submulțime a mulțimii de noduri.
- **Hipergrafuri k -uniforme:** fiecare muchie are cardinalul k .

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

Fiecare hypergraf poate fi reprezentat ca un graf bipartit:



Grade

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

Fie $G = (V, E)$ un graf și $v \in V$.

- **Gradul** unui nod v : $d_G(v) =$ numărul de muchii incidente cu v .
- v este un **nod izolat** dacă $d_G(v) = 0$ și **pendant** (sau **frunză**) dacă $d_G(v) = 1$.

$$\sum_{v \in V} d_G(v) = 2|E|.$$

- **Gradul maxim** $\Delta(G)$ și **gradul minim** $\delta(G)$:

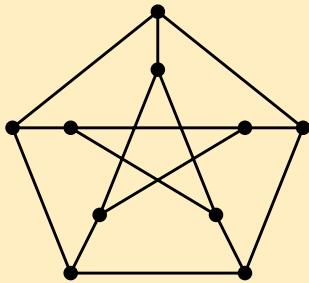
$$\Delta(G) = \max_{v \in V} d_G(v), \quad \delta(G) = \min_{v \in V} d_G(v).$$

- Dacă $\Delta(G) = \delta(G) = k$, atunci G este **k -regulat**.
- **Graf nul**: un graf 0-regulat .

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

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

Un graf 3-regulat (cubic): graful lui Petersen



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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 și $v \in V$.

- **Gradul interior** al unui nod v : $d_G^-(v) =$ numărul de arce incidente spre v .
- **Gradul exterior** al unui nod v : $d_G^+(v) =$ numărul de arce incidente din v .

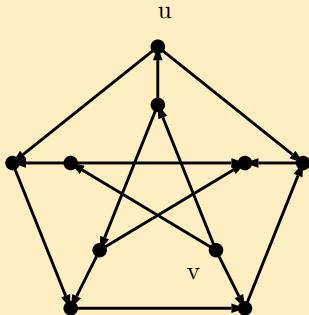
$$\sum_{v \in V} d_G^+(v) = \sum_{v \in V} d_G^-(v) = |E|.$$

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

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

Exemplu

$$d_G^+(u) = 2, d_G^-(u) = 1; d_G^+(v) = 3, d_G^-(v) = 0$$



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

Subgrafuri

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

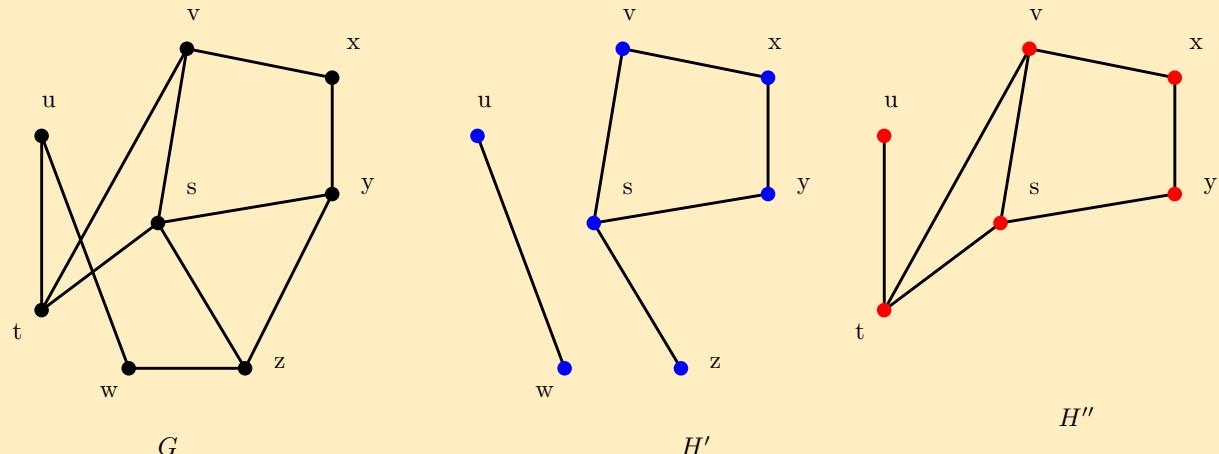
Fie $G = (V(G), E(G))$ un graf.

- **Subgraf** al lui G : un graf $H = (V(H), E(H))$ aşa încât $V(H) \subseteq V(G)$ și $E(H) \subseteq E(G)$.
- **Graf parțial** al lui G : un subgraf H al lui G astfel ca $V(H) = V(G)$.
- **Subgraf generat de $B \subseteq E(G)$ în G** : un subgraf al lui G , $H = (V(H), E(H))$, astfel că $E(H) = B$ și $V(H) = \cup_{uv \in B} \{u, v\}$. Se notează prin $\langle B \rangle_G$.
- **Subgraf induș**: un subgraf H al lui G aşa încât $E(H) = \binom{V(H)}{2} \cap E(G)$. Dacă $A \subseteq V(G)$, **subgraful induș de A în G** este notat cu $[A]_G$ sau $G[A]$.

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

Exemplu

Un graf G , un subgraf H' al lui G , și un subgraf induș al lui G : $H'' = G[\{u, v, x, y, s, t\}]$.



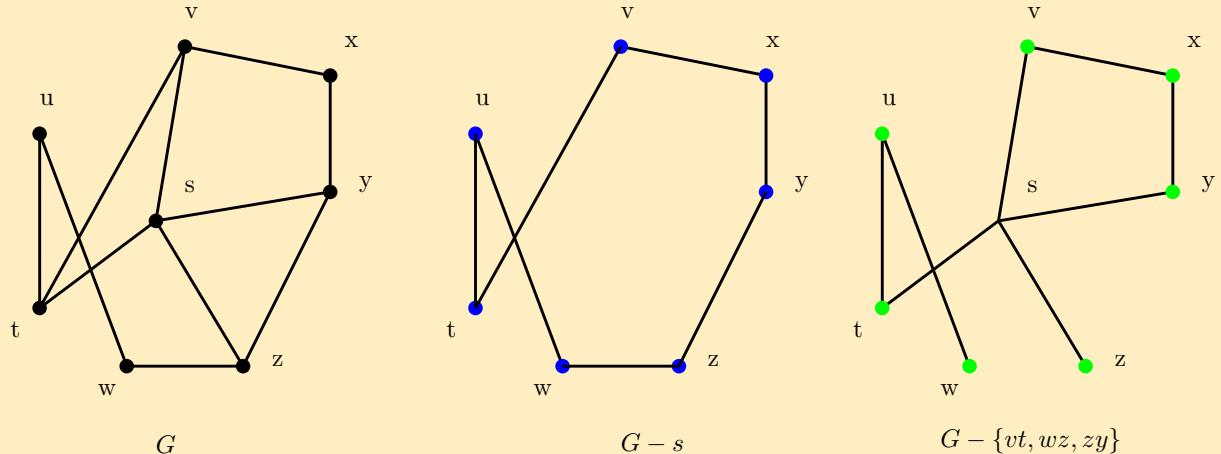
Subgrafuri

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

- Dacă $A \subseteq V(G)$, atunci subgraful $[V(G) \setminus A]_G$, notat prin $G - A$, este subgraful obținut din G prin ștergerea nodurilor lui A . $G - \{u\}$ este numit **subgraf de ștergere** și se notează, simplu cu $G - u$.
- Dacă $B \subseteq E(G)$, atunci subgraful $\langle E(G) \setminus B \rangle_G$, notat prin $G - B$, este subgraful obținut din G prin ștergerea tuturor muchiilor lui B . $G - \{e\}$ se notează $G - e$.
- Definiții și notății similare pentru digrafuri, multigrafuri etc.

Exemplu

Un graf G , $G - s$, și $G - \{vt, wz, zy\}$.

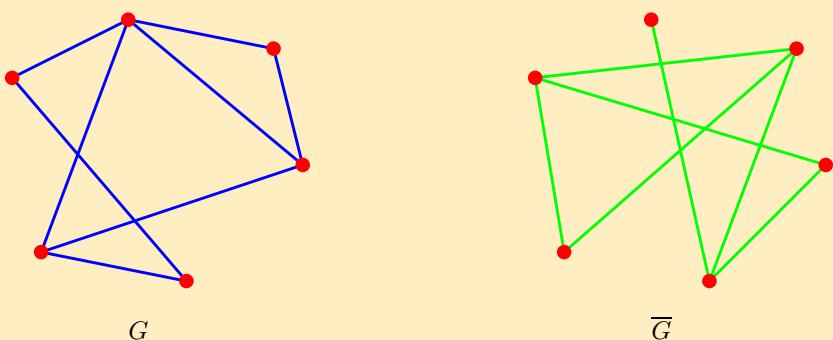


Operații cu grafuri

Operație unară: $G = (V(G), E(G))$

- **Complementul** unui graf G : un graf \overline{G} , cu $V(\overline{G}) = V(G)$ și

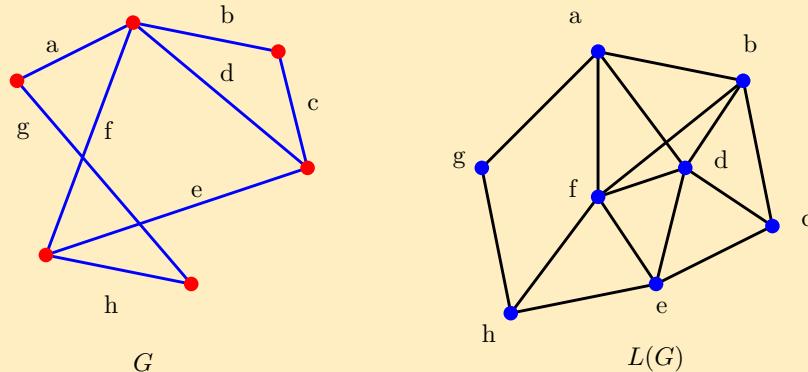
$$E(\overline{G}) = \binom{V(G)}{2} \setminus E(G).$$



Operație unară: $G = (V(G), E(G))$

- **Graful reprezentativ al muchilor (line-graful) lui G :** un graf $L(G)$, cu $V(L(G)) = E(G)$ și

$$E(L(G)) = \{ef : e, f \in E(G), e \text{ și } f \text{ sunt adiacente în } G\}.$$



Operații cu grafuri

Operație unară: $G = (V(G), E(G))$

- **Graful obținut din G prin inserarea unui nou (z) pe o muchie ($e = uv$):** graful G' , cu $V(G') = V(G) \cup \{z\}$ și

$$E(G') = E(G) \setminus \{uv\} \cup \{uz, zv\}.$$

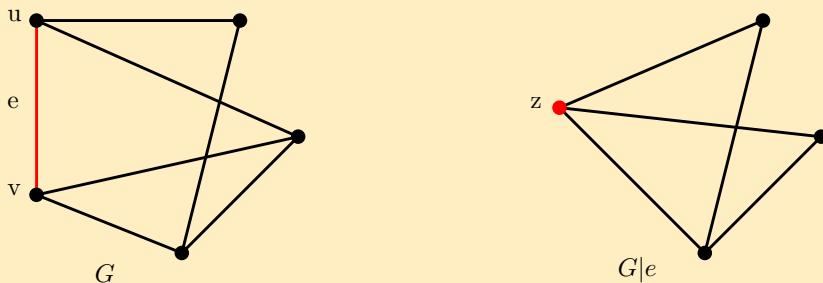


Operație unară: $G = (V(G), E(G))$

- Graful obținut din G prin contractia unei muchii $e = uv \in E(G)$: graful $G|e$ cu

$$V(G|e) = V(G) \setminus \{u, v\} \cup \{z\},$$

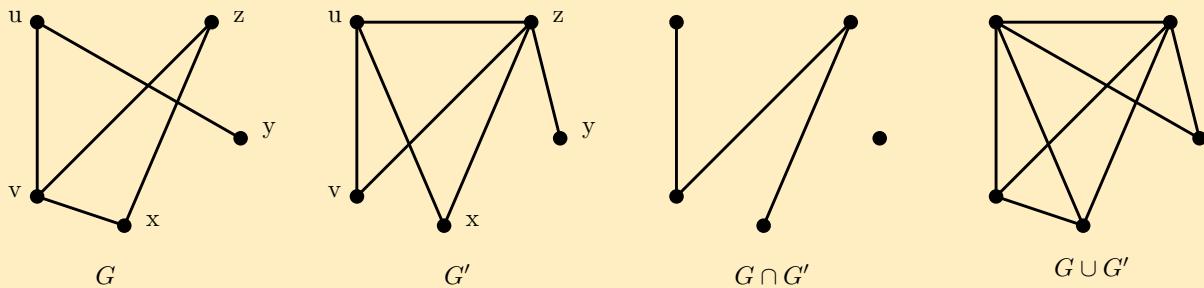
$$E(G|e) = E([V(G) \setminus \{u, v\}]_G) \cup \{yz : yu \text{ sau } yv \in E(G)\}.$$



Operații cu grafuri

Operații binare: G, G' cu $V(G) = V(G')$

- Intersecția $G \cap G' = (V(G), E(G) \cap E(G'))$.
- Reuniunea $G \cup G' = (V(G), E(G) \cup E(G'))$.

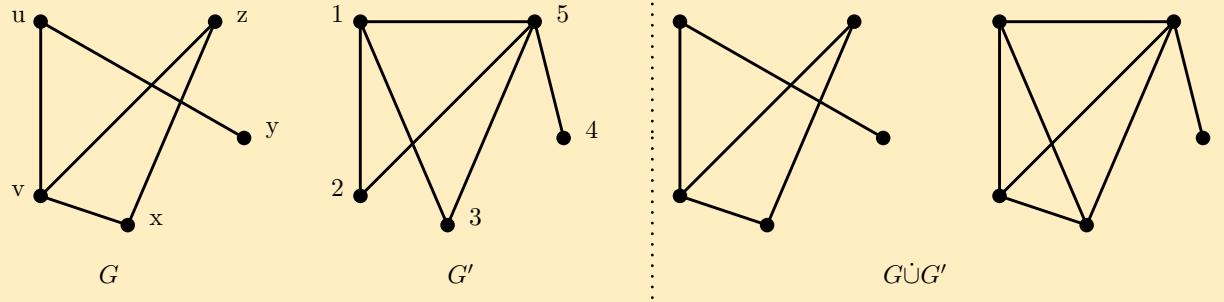


C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C.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

Operație binară: G, G' cu $V(G) \cap V(G') = \emptyset$

- **Reuniunea disjunctă** $G \dot{\cup} G' = (V(G) \cup V(G'), E(G) \cup E(G'))$.

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



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

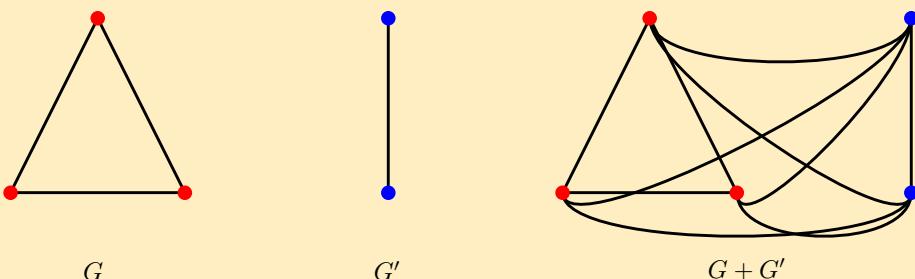
Operații cu grafuri

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

Operație binară: G, G' cu $V(G) \cap V(G') = \emptyset$

- **Suma directă (join)** $G + G' = \overline{G \dot{\cup} G'}$.

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



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

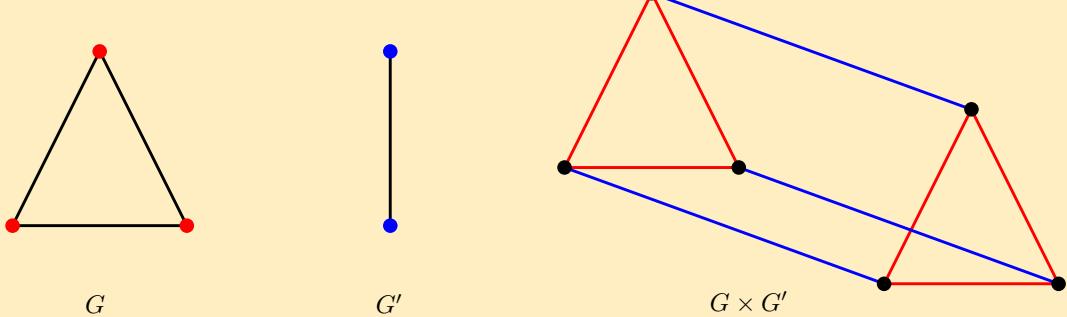
Operație binară: G, G' cu $V(G) \cap V(G') = \emptyset$

- **Produsul cartezian** al grafurilor G și G' : graful $G \times G'$ cu

$$V(G \times G') = V(G) \times V(G').$$

$$E(G \times G') = \{(u, u')(v, v') : u, v \in V(G), u', v' \in V(G'),$$

$$u = v \text{ și } u'v' \in E(G') \text{ sau } u' = v' \text{ și } uv \in E(G)\}.$$

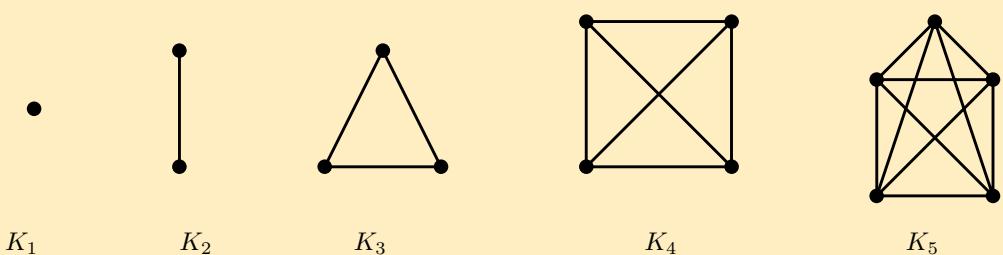


Clase de grafuri - Grafurile complete

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

Graful complet de ordin n , K_n : $|V(K_n)| = n$ și $E(K_n) = \binom{V(K_n)}{2}$.

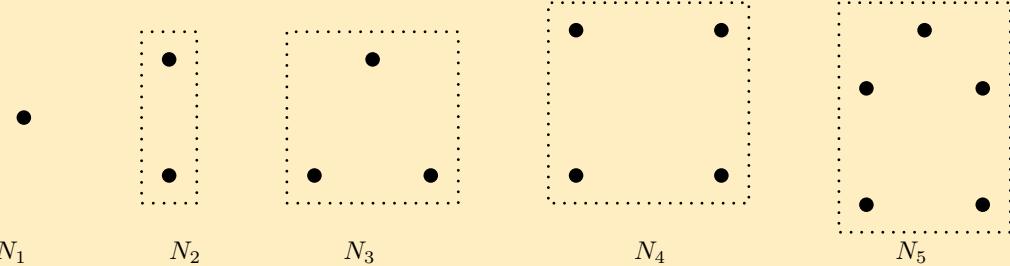
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru



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

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

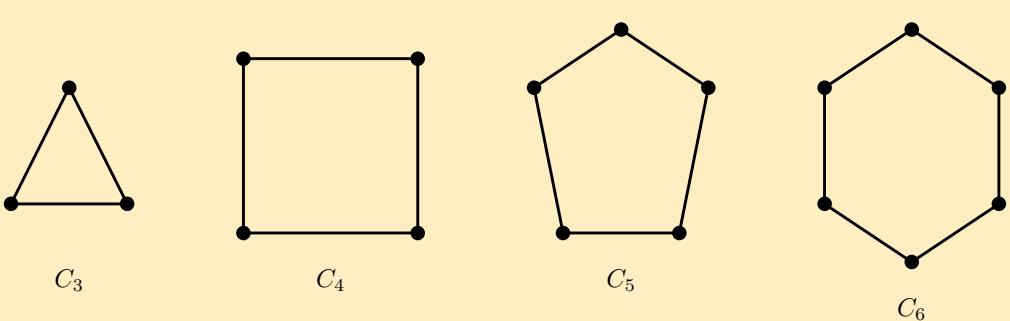
Graful nul de ordin n , N_n : $|V(N_n)| = n$ și $E(N_n) = \emptyset$.



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

Clase de grafuri - Circuitele C_n

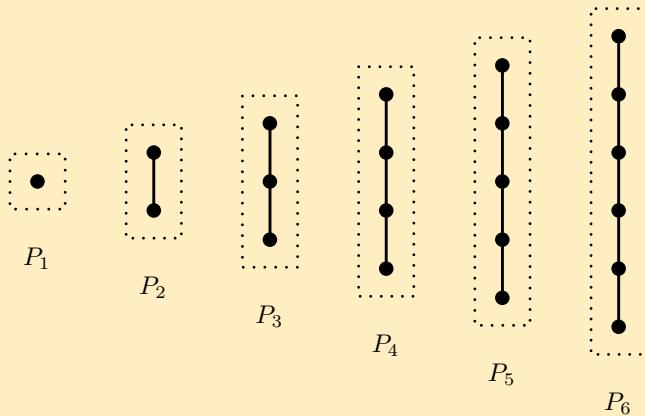
Circuitul de ordin n , C_n : $V(C_n) = \{1, 2, \dots, n\}$ și $E(C_n) = \{12, 23, \dots, n-1n, n1\}$.



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

Drumul de ordin n , P_n : $V(P_n) = \{1, 2, \dots, n\}$ și $E(P_n) = \{12, 23, \dots, n-1n\}$.

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



Algoritmică grafurilor - Cursul 2 9 octombrie 2020 31 / 72

Clase de grafuri - Clicile

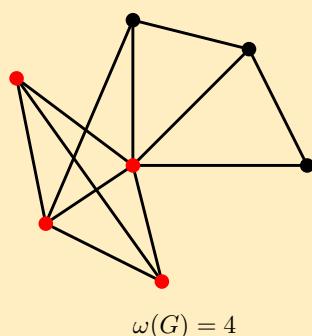
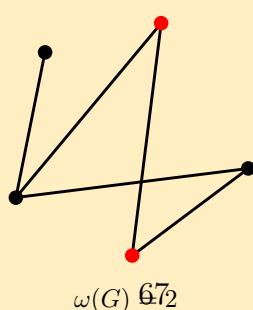
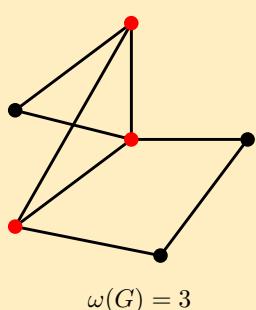
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

O submulțime de k noduri a graf G care induce un graf complet este numită o **k -clică**.

numărul de clică al lui G : $\omega(G) = \max_{Q \text{ clică în } G} |Q|$.

Remarcăm că $\omega(G) = \alpha(\overline{G})$.

Exemplu

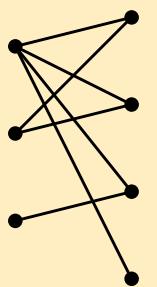


Graf bipartit: un graf G cu proprietatea că $V(G)$ poate fi partionat în două clase care sunt multimi stabile.

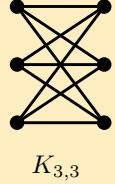
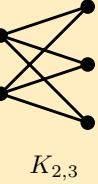
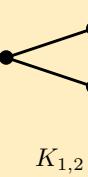
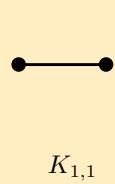
Dacă $V(G) = S \cup T$, $S \cap T = \emptyset$, $S, T \neq \emptyset$, cu S și T multimi stabile în G , atunci G este notat $G = (S, T; E(G))$.

Graf bipartit complet: $G = (S, T; E(G))$, cu $uv \in E(G)$, $\forall u \in S$ și $\forall v \in T$; se notează cu $K_{s,t}$, unde $s = |S|$, $t = |T|$.

Exemplu



A bipartite graph



Clase de grafuri - Grafuri planare

Graf planar: un graf care poate fi reprezentat într-un plan astfel ca fiecărui nod să îi corespundă un punct al aceluia plan și fiecărei muchii să îi corespundă o curbă simplă care unește punctele corespunzătoare extremităților și aceste curbe se intersectează doar în extremitățile lor. Un graf care nu este planar este numit **graf ne-planar**.

Grafuri planare: Problemă de decizie

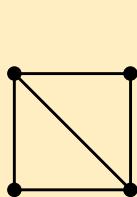
PLAN Instanță: G graf.
întrebare: Este G planar?

PLAN aparține clasei de complexitate P (Hopcroft, Tarjan, 1972, $\mathcal{O}(n + m)$).

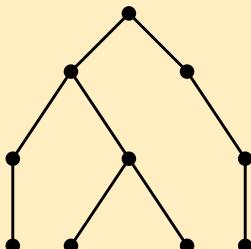
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C.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

Exemplu

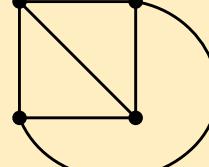
Grafuri planare.



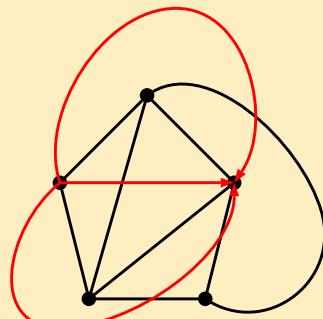
planar graph



planar graph



planar graph



K_5 non-planar graph

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

Clase de grafuri - Grafuri \mathcal{F} -free

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

- Aceasta este metoda obișnuită de a defini o clasă de grafuri prin interzicerea unor anumite subgrafuri.
- Dacă \mathcal{F} este o mulțime de grafuri atunci un graf G este **\mathcal{F} -free** dacă G nu conține niciun subgraf inducăt izomorf cu vreun graf din \mathcal{F} .
- Dacă \mathcal{F} este un singleton, $\mathcal{F} = \{H\}$, atunci scriem simplu **H -free**.

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

Exemplu

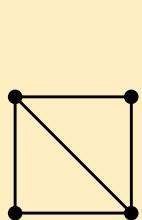
- clasa grafurilor nule este exact clasa grafurilor K_2 -free.
- Un graf P_3 -free este o reuniune disjunctă de grafuri complete.
- **Grafuri triangulate (cordale)**: grafurile $(C_k)_{k \geq 4}$ -free.

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

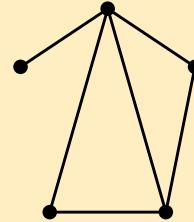
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C.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

Exemplu

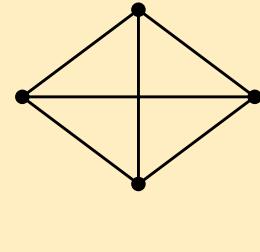
Grafuri \mathcal{F} -free.



a $2K_2$ -free graph



a C_4 -free graph



a P_3 -free graph

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 și circuite - Mersuri, parcursuri, drumuri

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

- Un **mers de lungime r** de la u până la v în G : o secvență de noduri și **muchii** de forma

$$(u =) v_0, v_0 v_1, v_1, \dots, v_{r-1}, v_{r-1} v_r, v_r (= v).$$

u și v sunt extremitățile mersului.

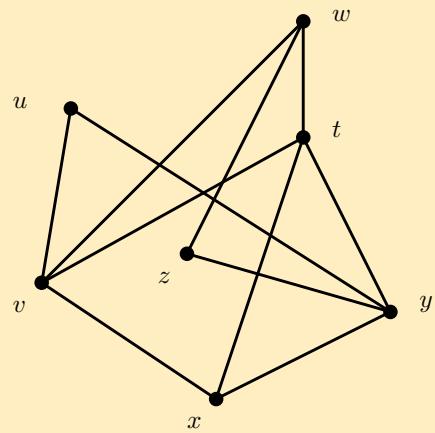
- **Parcurs**: un mers cu muchii distințe.
- **Drum**: un mers cu noduri distințe.

Un nod este un mers (parcurs, drum) de lungime 0.

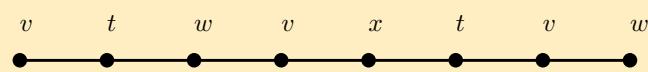
Algorithms * C. Croitoru - Graph Algor70ms * C. Croitoru - Graph Algorithms *

Exemplu

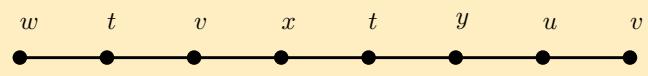
Mersuri, parcursuri, drumuri.



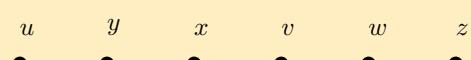
walk:



trail:



path:



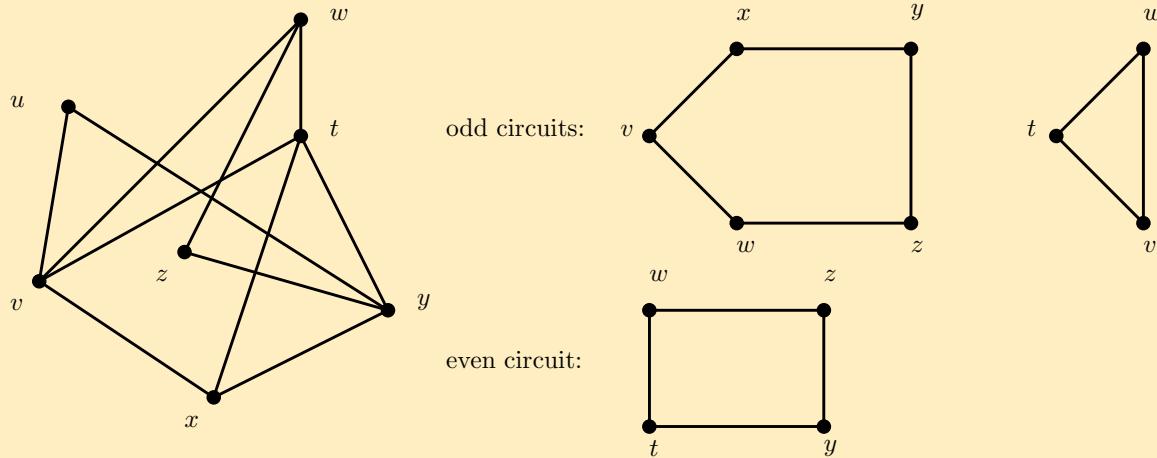
Drumuri și circuite - Mersuri închise, circuite

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

- **Mers închis**: un mers de la u la u .
 - **Circuit (drum închis)**: un mers cu noduri care sunt distințe cu excepția extremităților care coincid.
 - Un **circuit** este **par** sau **impar** în funcție de paritatea lungimii sale.
 - Lungimea celui mai scurt circuit (dacă există) este **grația**, $g(G)$, lui G .
 - Lungimea celui mai lung circuit este **circumferința**, $c(G)$, lui G .

Exemplu

Mersuri închise, circuite.

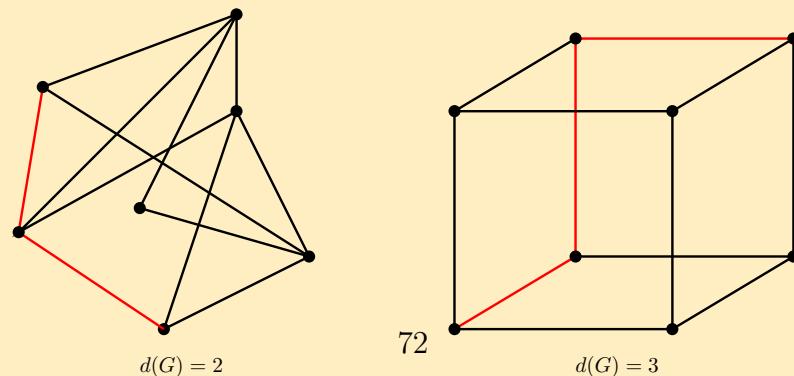


Drumuri și circuite - Distanță, diametru

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

- **Distanța în G de la u la v , $d_G(u, v)$** lungimea celui mai scurt drum în G de la u la v (dacă există un astfel de drum).
- **Diametrul unui graf G , $d(G)$:**

$$d(G) = \max_{u, v \in V} d_G(u, v).$$



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 * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
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 $D = (V, E)$ un digraf.

Toate definițiile de mai sus se păstrează considerând **arce** (muchii orientate) în locul muchiilor.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 și circuite - Conexiune

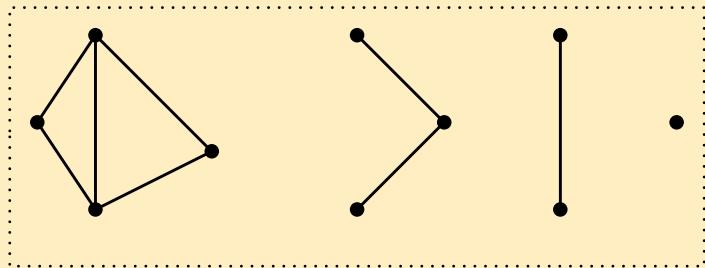
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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

- **Graf conex:** există câte un drum între orice două noduri ale grafului. Altfel graful este **neconex**.
- **Componentă conexă** a unui graf G : un subgraf maximal conex, H , of G (i. e., nu există vreun subgraf conex H' of G , $H' \neq H$, iar H este subgraf al lui H').
- Orice graf poate fi scris ca o reuniune disjunctă a componentelor sale conexe.
- Următoarea relație binară este o relație de echivalență: $\rho \subseteq V \times V$, dată prin $u\rho v$ (i. e., $(u, v) \in \rho$) dacă există un drum în G între u și v .
- Componentele conexe ale lui G sunt subgrafurile induse de clasele de echivalență ale relației ρ .

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

Exemplu



four connected components

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 și circuite - Conexiune

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

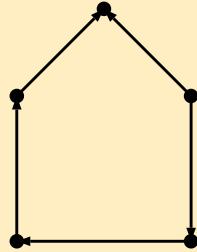
Fie $D = (V, E)$ un digraf.

- **Digraf slab conex** (sau simplu, **conex**) graful său suport $G(D)$ este conex.
- **Digraf unilateral conex**: există un drum de la u la v sau de la v la u , pentru orice două noduri $u, v \in V$.
- **Digraf tare conex**: există un drum de la u la v , pentru orice două noduri $u, v \in V$.

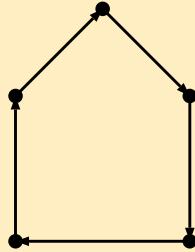
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

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

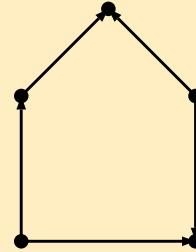
Exemplu



unilaterally connected



strongly connected



weakly connected

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 și circuite - Conexiune

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 graf conex.

- **Punct de articulație (cut-vertex):** un nod $v \in V$ astfel că $G - v$ este neconex.
- **Mulțime de articulație (vertex cutset):** o mulțime de noduri $S \subseteq V$ așa încât $G - S$ este neconex.
- Un **arbore** este un graf conex fără circuite.
- Un **graf ale cărui componente conexe sunt arbori** este o **pădure**.

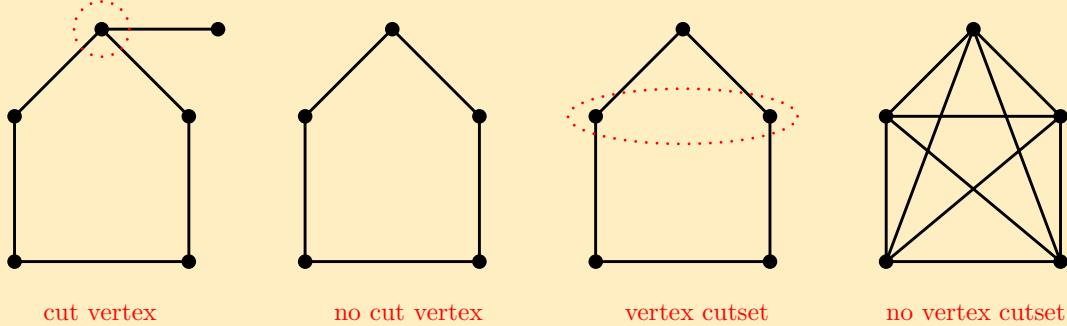
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Mai general: într-un graf G care nu este neapărat conex $v \in V(G)$ este un **punct de articulație** dacă $G - v$ are mai multe componente conexe decât G .

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

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

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

Drumuri și circuite - Conexiune

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

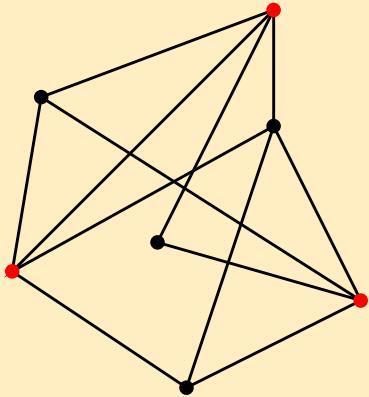
Fie $G = (V, E)$ un graf.

- Pentru $p \in \mathbb{N}^*$, G este **graf p -conex** dacă
 - ▶ $|V| = p$ și $G = K_p$ sau
 - ▶ $|V| \geq p + 1$ și G nu are multime de articulație de cardinal $< p$ (G nu poate fi deconectat prin stergerea a mai puțin de p noduri).
- Evident, G este 1-conex dacă și numai dacă este conex.
- **Numărul de conexiune pe noduri**, $k(G)$, al unui graf G este

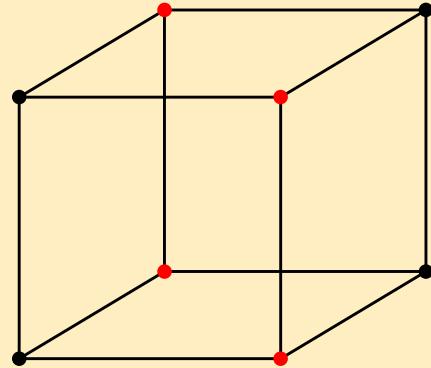
$$k(G) = \max \{p \in \mathbb{N}^* : G \text{ este } p\text{-conex}\}.$$

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

Exemplu



$$k(G) = 3$$



$$k(G) = 4$$

Drumuri și circuite - Conexiune

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

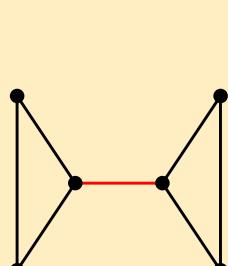
- **Punte (bridge):** o muchie $e \in E$ astfel că $G - e$ nu este conex.
- **Mulțime de muchii de articulație (tăietură sau edge-cutset):** O sub-mulțime de muchii $S \subseteq E$ așa încât $G - S$ este neconex.
- Pentru $p \in \mathbb{N}^*$, G este **graf p -muchie-conex** dacă G nu are o mulțime de muchii de articulație de cardinal $< p$ (G nu poate fi deconectat prin ștergerea a mai puțin de p muchii).
- **Numărul de conexiune pe muchii, $\lambda(G)$,** al unui graf G este

$$\lambda(G) = \max \{p \in \mathbb{N}^* : G \text{ este } p\text{-muchie-conex}\}.$$

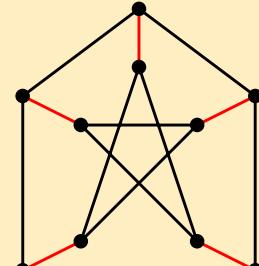
Mai general: într-un graf G care nu este neapărat conex $e \in E(G)$ este o **punte** dacă $G - e$ are mai multe componente conexe decât G .

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

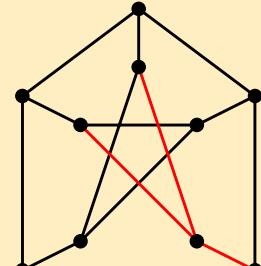
Exemplu



a cut edge



an edge-cutset



$\lambda(G) = 3$

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

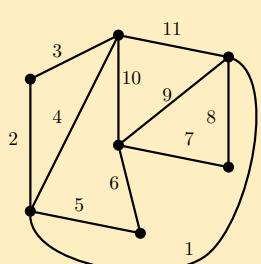
Drumuri și circuite - Grafuri Euleriene și Hamiltoniene

Fie G un (di)graf.

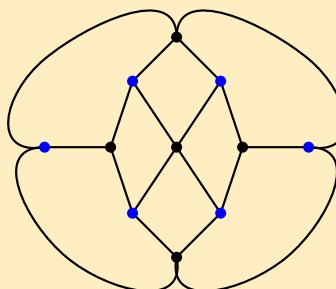
- G este **Eulerian** dacă există un parcurs închis în G care trece prin fiecare muchie a lui G .
- G este **Hamiltonian** dacă există un circuit în G care trece prin fiecare nod al lui G .

Recunoașterea (di)grafurilor Euleriene se face în timp polinomial (**Euler**, 1736).

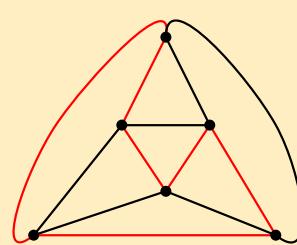
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph



an Eulerian graph



78
a non Hamiltonian graph
(bipartite of odd order)



a Hamiltonian graph

Probleme Hamiltoniene

HAM Instanță: G un graf.
Întrebare: Este G Hamiltonian?

NP-completă (Karp, 1972).

NH Instanță: G un graf.
Întrebare: Este G ne-Hamiltonian?

NH \in NP?

Exerciții pentru seminarul de săptămâna viitoare

Exercițiul 1. Fie G_1 și G_2 două grafuri. Arătați că dacă $G_1 \times G_2$ este conex, atunci G_1 și G_2 sunt conexe.

Exercițiul 2.

Pentru $d \in \mathbb{N}^*$, considerăm $G_d = K_2 \times K_2 \times \dots \times K_2$ (de d ori).

- Determinați ordinul și dimensiunea lui G_d .
- Arătați că G_d este bipartit și determinați $\alpha(G_d)$.

Exercițiu 3.

Fie D un turneu conținând un circuit C de lungime $n \geq 4$. Arătați că pentru orice nod u al lui C se poate determina, în timpul $\mathcal{O}(n)$, un circuit de lungime 3 care trece prin u .

Exercițiu 4.

Fie G un graf conex cu $n \geq 2$ noduri și m muchii. Arătați că:

- Dacă G are exact un circuit, atunci $m = n$.
- Dacă G nu are frunze, atunci $m \geq n$.
- Dacă G este un arbore, atunci are cel puțin două frunze.

Exerciții pentru seminarul de săptămâna viitoare

Exercițiu 5

Fie G un graf cu $n \geq 2$ noduri. Arătați că:

- Dacă G este conex, atunci conține cel puțin un nod care nu este punct de articulație.
- Dacă $n \geq 3$ și G este conex atunci conține două noduri care nu sunt puncte de articulație.
- Adevărat sau fals: dacă G este conex iar x este un punct de articulație al său, atunci x este o frunză într-un arbore parțial al lui G ?

Exercițiu 6

Fie G un graf conex care nu conține două noduri pendante (frunze) cu un vecin în comun. Arătați că există două noduri adiacente prin stergerea cărora G nu se deconectează.⁸⁰

Exercițiu 7

Fie G un graf și H graful său reprezentativ al muchiilor ($H = L(G)$). Arătați că H este $K_{1,3}$ -free.

Exercițiu 8

Fie G un graf. Arătați că:

- Dacă G are exact două noduri de grad impar, atunci aceste două noduri sunt unite printr-un drum în G .
- Dacă G este conex cu toate nodurile de grad par, atunci G are o muchie care nu este puncte (ștergerea ei nu deconectează graful).
- Dacă G este conex cu toate nodurile de grad par, atunci nicio muchie a lui G nu este puncte.

Exerciții pentru seminarul de săptămâna viitoare

Exercițiu 9

Fie G un graf. Arătați că

- Numărul de noduri de grad impar este par.
- Dacă G este conex și are k noduri de grad impar, atunci G este o reuniune $[k/2]$ parcursuri disjuncte pe muchii.

Exercițiu 10

Fie G un graf astfel ca $N_G(u) \cup N_G(v) = V(G)$, $\forall u, v \in V(G)$, $u \neq v$. Arătați că G este graf complet.

Exercițiu 11

Fie G un graf cu proprietatea că $d_G(u) + d_G(v) \geq |G| - 1$, $\forall u, v \in V(G)$, $u \neq v$. Arătați că diametrul lui $d(G) \leq 2$.

Exercițiu 12

Fie $G = (V, E)$ un graf cu $V = \{v_1, v_2, \dots, v_n\}$ astfel ca $d_G(v_1) \leq d_G(v_2) \leq \dots \leq d_G(v_n)$. Arătați că G este conex dacă $d_G(v_p) \geq p$, pentru orice $p \leq n - d_G(v_n) - 1$.

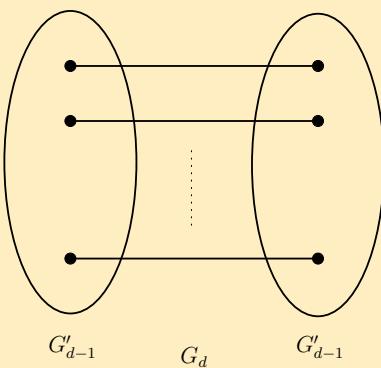
Exercițiu 13

Fie $G = (V, E)$ un graf și S o mulțime stabilă a lui G . Demonstrați că S este stabilă de cardinal maxim dacă și numai dacă pentru orice mulțime stabilă a lui G , $S' \subseteq V \setminus S$, avem

$$|S'| \leq |N_G(S') \cap S|.$$

Exerciții rezolvate (partial)**Exercițiu 2. Soluție.**

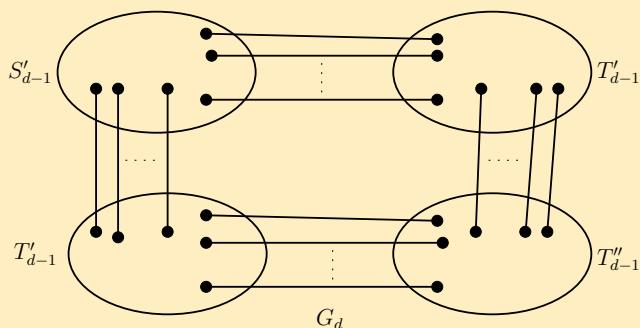
- (a) Deoarece pentru $d \geq 2$, $G_d = K_2 \times G_{d-1}$, putem construi G_d astfel: considerăm $G'_{d-1} \stackrel{\varphi}{\sim} G''_{d-1}$ două copii isomorfe ale lui G_{d-1} și legăm perechile de noduri corespunzătoare (prin izomorfism) cu câte o muchie (vezi figura).



- Astfel $|G_d| = 2|G_{d-1}|$ and $|E(G_d)| = 2|E(G_{d-1})| + |G_{d-1}|$, cu $|G_1| = 2$ și $|E(G_1)| = 1$.
- Rezolvați aceste două recurențe.

(b) Demonstrăm prin inducție după d că G_d este bipartit cu bipartiția (S_d, T_d) și că $\alpha(G_d) = S_d = T_d = 2^{d-1}$.

- Afirmația este evidentă pentru $d = 1$.
- În pasul inductiv presupunem că G_{d-1} ($d \geq 2$) este bipartit cu bipartiția (S_{d-1}, T_{d-1}) și $\alpha(G_{d-1}) = S_{d-1} = T_{d-1} = 2^{d-2}$ (vezi figura de mai jos).

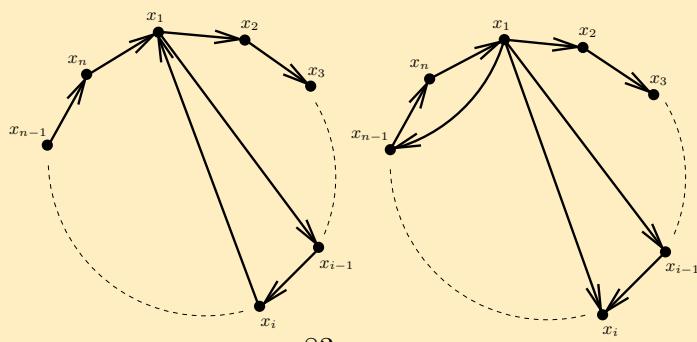


- Cum se poate defini o bipartiție pentru G_d folosind submulțimile $S_{d-1}, T_{d-1}, S'_{d-1}, T'_{d-1}$?

Exerciții rezolvate (partial)

Exercițiul 3. Soluție.

- Fie $V(C) = \{x_1, x_2, \dots, x_n\}$, $E(C) = \{x_1x_2, x_2x_3, \dots, x_nx_1\}$; presupunem că $x = x_1$. Parcurgem acest circuit: dacă $x_3x_1 \in E(D)$, returnăm $\{x_1, x_2, x_3\}$, altfel $x_1x_3 \in E(D)$; dacă $x_4x_1 \in E(D)$, atunci returnăm $\{x_1, x_3, x_4\}$ altfel ...
- Ce putem returna când toate corzile incidente cu x_1 pleacă spre exterior?



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

Algoritmul de complexitate $\mathcal{O}(n)$ este:

```
i ← 3;  
while( $x_1 x_i \in E(D)$ )  
    i ++;  
if( $i < n - 1$ )  
    return  $\{x_1, x_{i-1}, x_i\}$ ;  
else  
    return what?;
```

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

- (a) $G - e$ este un arbore, $\forall e \in E(C)$ unde C este circuitul din G .
Puteți explica de ce?
- (b) $2m = 2|E(G)| = \sum_{v \in V(G)} d_G(v) \geq 2|V(G)| = 2n$.
- (c) Dacă G nu are frunze, atunci $m \geq n$ - contradicție.
• Dacă G are doar o frunză, atunci $2|E(G)| = \sum_{v \in V(G)} d_G(v) \geq 2|V(G)| - 1 \Rightarrow 2(|V(G)| - 1) \geq 2|V(G)| - 1$ - contradicție.

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

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

Exercițiu 5. Soluție.

- (b) Dacă T este un arbore parțial al lui G și $x \neq y$ sunt două frunze
ale lui T , atunci $G - x$, $G - y$ sunt grafuri conexe. **De ce?**
- (c) Adevărat. **Dar de ce?**

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiu 6. Soluție.

- Fie D un cel mai lung drum în G , $V(D) = \{x_1, x_2, \dots, x_p\}$;
- $N_G(x_1) \subseteq V(D) \setminus \{x_1\}$ altfel există drumuri mai lungi decât D . **De ce?**
- Presupunem că $G - \{x_1, x_2\}$ nu este conex, atunci există un nod
 $y \notin V(D)$ astfel ca $N_G(y) \cap V(D) = \{x_2\}$; mai mult, $d_G(y) = 1$.
De ce?
- Repetând aceeași demonstrație cu $D - x_1 + y$ în loc de D obținem
un alt nod z , cu $N_G(z) = \{x_2\}$ și $d_G(z) = 1$.
- Dar y și z sunt noduri pendante care au un același vecin (x_2) -
contradicție.

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

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

Exercițiu 8. Soluție.

- (a) Dacă x și y (nodurile de grad impar) sunt în componente conexe ale lui G , atunci componenta care conține pe x are un singur nod de grad impar - contradicție. **Puteți explica de ce?** (Aveți în vedere suma tuturor gradelor.)
- (b) G are circuite (**de ce?**) iar o muchie de pe un circuit nu poate fi puncte (**de ce?**).
- (c) Fie $xy \in E(G)$; x și y sunt singurele noduri de grad impar din $G - xy$, deci $G - xy$ este conex (i. e., xy nu este puncte în G).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiu 9. Soluție.

- (a) $2|E(G)| = \sum_{v \in V(G)} d_G(v)$ iar un număr impar de noduri de grad impar ar face suma aceasta impară (!).
- (b) Adăugăm un nod nou la G și îl facem adiacent cu toate cele k noduri de grad impar; noul graf H are doar grade pare și este și conex, deci admite un parcurs eulerian închis. **Puteți spune de ce?**
- Ștergând nodurile adăugate obținem $k/2$ parcursuri disjuncte pe muchii. (k este par iar muchiile sterse sunt în perechi de muchii adiacente - **de ce?**)

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algor86ms * C. Croitoru - Graph Algorithms *

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

Exercițiu 11. Soluție.

- Fie $u, v \in V(G)$, dacă $uv \notin E(G)$ și $N_G(u) \cap N_G(v) = \emptyset$, atunci $\{u, v\} \cap (N_G(u) \cup N_G(v)) = \emptyset$, adică $d_G(u) + d_G(v) \leq |G| - 2$ - contradicție.
 - Astfel, dacă $u, v \in V(G)$ și $uv \notin E(G)$, atunci $N_G(u) \cap N_G(v) \neq \emptyset$ - de unde $d_G(u, v) = 2$. De ce?

Exercitii rezolvate (partial)

Exercițiul 12. Soluție.

- Să presupunem prin reducere la absurd că G nu este conex.
 - Fie G' o componentă conexă a lui G aşa încât $v_n \notin V(G') = \{v_{j_1}, v_{j_2}, \dots, v_{j_p}\}$, $1 \leq j_1 < j_2 < \dots < j_p < n$.
 - Deoarece G' nu conține v_n și nici vreun vecin de-al său, $p \leq n - d_G(v_n) - 1$. **De ce?**
 - Astfel $d_G(v_p) \geq p$, dar $d_G(v_{j_p}) \geq d_G(v_p)$ și cum $\{v_{j_p}\} \cup N_G(v_{j_p}) \subseteq V(G')$, urmează că $p = |V(G')| \geq p + 1$ - contradicție.

3 CURS 3: Vocabularul Teoriei grafurilor. Probleme de drum minim in (di)grafuri.

- circuit de lungime impara \implies graful nu e bipartit

Algoritmica Grafurilor - Cursul 3

Cuprins

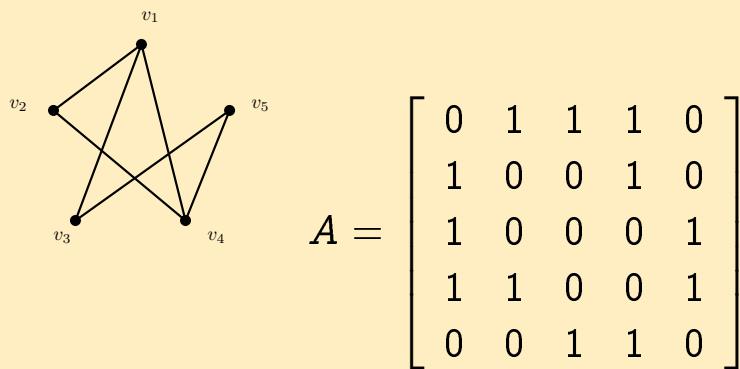
- 1 **Vocabularul teoriei grafurilor**
 - Matrici asociate
 - 2 **Structuri de date**
 - 3 **Probleme de drumuri în (di)grafuri**
 - Traversarea (di)grafurilor
 - Drumuri de cost minim
 - 4 **Exerciții pentru seminarul de săptămâna viitoare**
 - 5 **Exerciții rezolvate (partial)**

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.



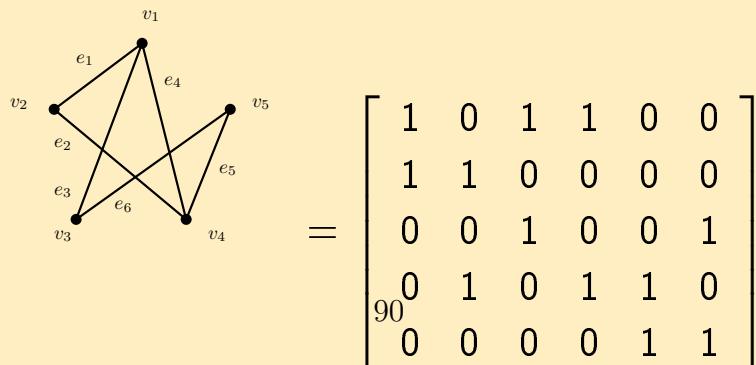
Matrici asociate - Matricea de incidentă

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



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 incidentă 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

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

Structuri de date pentru matricea de adiacență

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.

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 înlăntuite sau tablouri.
- Testarea dacă un nod u este adjacent 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).

Probleme de drumuri - Traversarea (di)grafurilor

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.

```

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

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

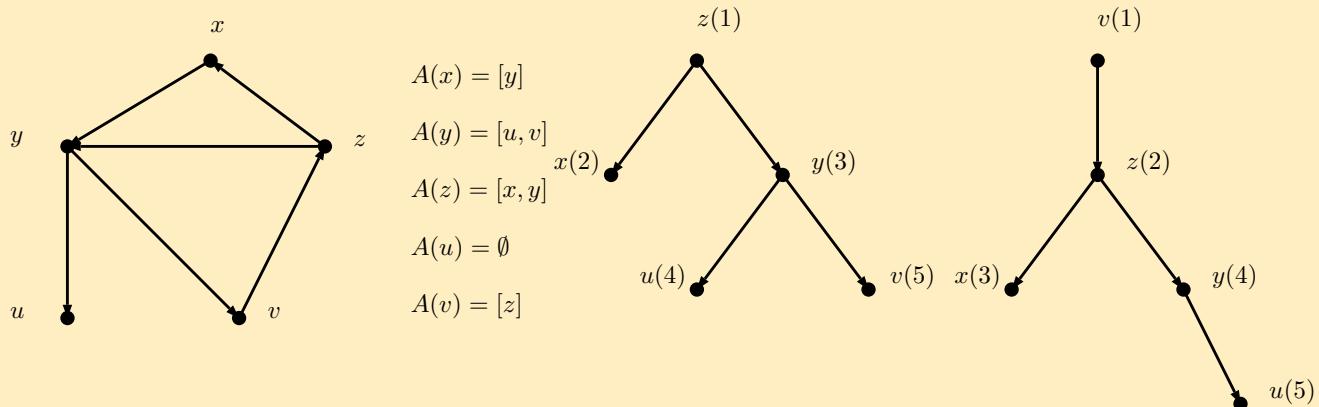
Proprietăți ale BFS. Nu este dificil de arătat că:

- $S = \{u \in V : label(u) \geq 0\}$;
- $\forall u \in V, label(u) = d_G(s, u)$ (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 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 * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exemplu

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



(Am marcat în paranteze ordinea de vizitare.)

Probleme de drumuri - Depth-First Search (DFS)

```

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
                 $delete(S, u)$ ;

```

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 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 ;
- $\forall u \in V, \text{label}(u) =$ 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).

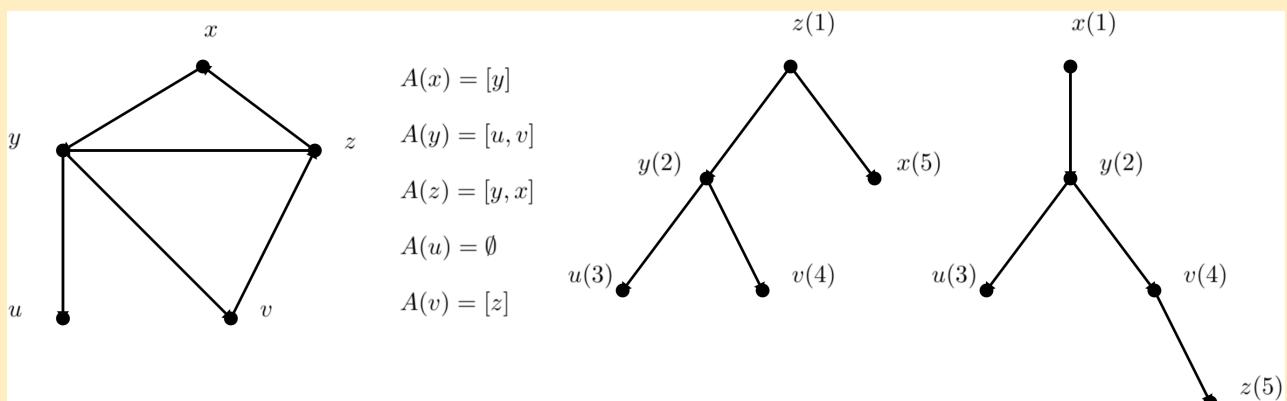
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

Exemplu

Două parcurgeri DFS ale unui același digraf (începând din z și din x):



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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 (muchiie 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 ușurință 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 *

Drumuri de cost minim - Notații

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 asemenei 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 Algor96ms * C. Croitoru - Graph Algorithms *

- C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $P_{ij} \in \mathcal{P}_{ij}$, $P : (i =)v_0, v_0v_1, v_1, \dots, v_{r-1}, v_{r-1}v_r, v_r (= j)$, atunci

$$V(P_{ij}) = \{v_0, v_1, \dots, v_r\}, E(P_{ij}) = \{v_0v_1, \dots, v_{r-1}v_r\}.$$

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

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

- In particular $a(P_{ii}) = 0$.

Principalele probleme de drum de cost minim

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 si toate celelalte noduri

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

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

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

Remarci 1

- Reprezentarea cu matrice de cost-adiacentă 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.

Drumuri de cost minim - Aplicații

1. Rețele de comunicare (Communication Networks). Digaful $G = (V, E)$ 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.

Drumuri de cost minim - Aplicații

2. Rețele PERT - Metoda drumului critic (Critical path method - CPM).

PERT (Path Evaluation and Review Technique) este o metodă de analiză (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.

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

Asociem acestei probleme 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 doar 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

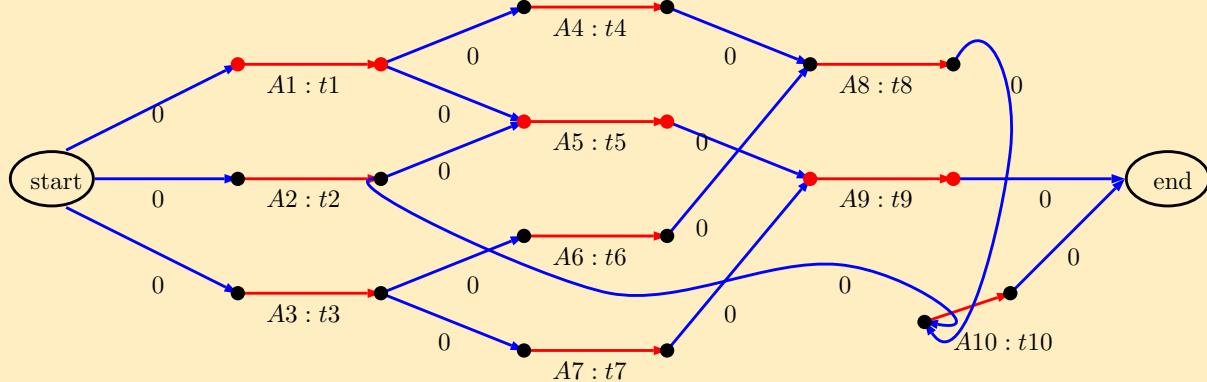
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

- 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 care nu intră vreun arc și a unui nod t corespunzând finalizării proiectului, legat prin arce $j_p t$ pentru fiecare activitate A_p din care nu ieșe 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 maxim este numit a **drum critic** deoarece orice întârziere a unei activități de pe acest drum implică o întârziere a întregului proiect.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo100ns * C. Croitoru - Graph Algorithms *

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

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
 - 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 -
 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 Algo101ns * 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 $j - a_i$).

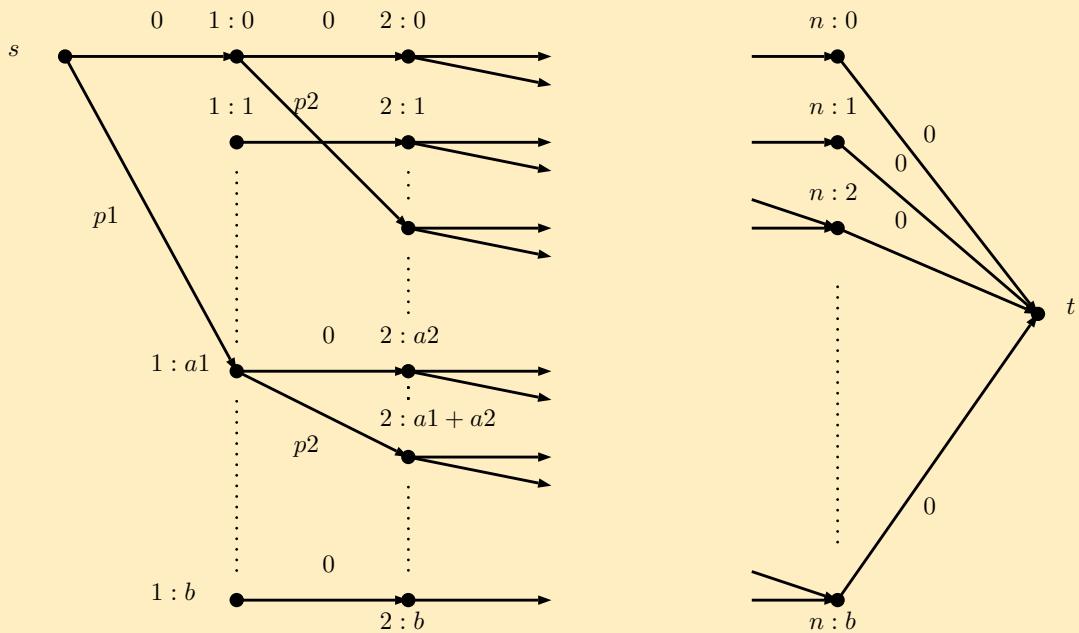
Drumuri de cost minim - Aplicații

- ▶ $n^j t$ 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 dela s la t în G , urmează că problema rucsacului poate fi rezolvată prin determinarea unui drum de cost maxim în digraful aciclic G .
- Descrierea statică dată mai sus pentru G poate fi transformată într-o procedură, folosind programarea dinamică. Problema aceasta este NP-hard (ordinul lui G poate fi exponențial în dimensiunea problemei).

Exemplu



- 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

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

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

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

Proof:

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

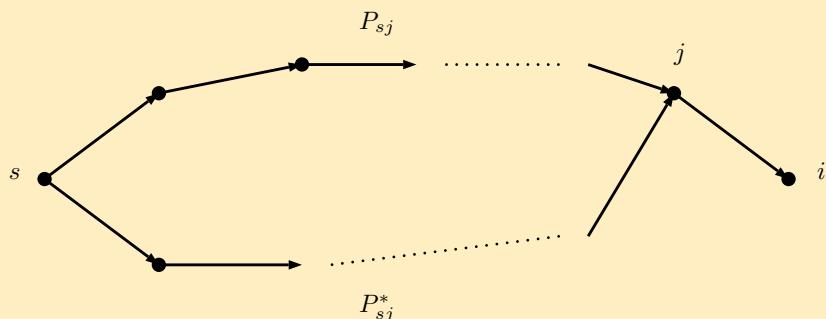
Ipoteza (I) implică $u_s = 0$, i.e., prima ecuație a sistemului (B) este satisfăcută. Pentru $i \neq s$, drumul P_{si}^* are penultimul nod j . Dacă P_{sj} este drumul de la s la j determinat pe P_{si}^* de j , avem

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

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

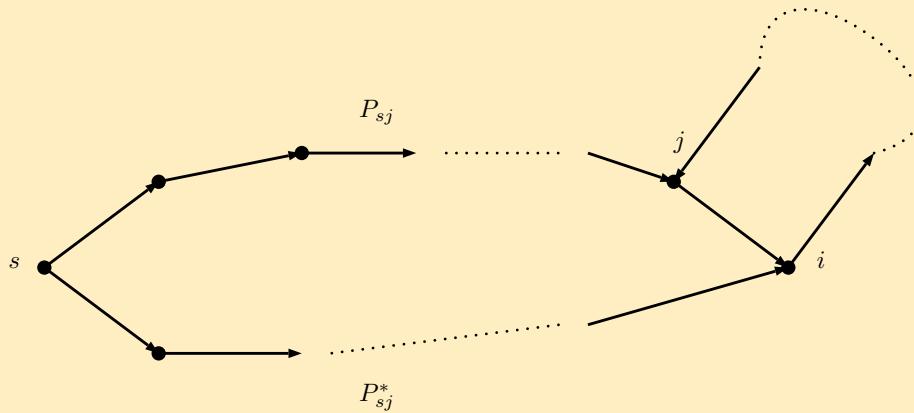
Drumuri de cost minim - Rezolvarea problemei P2

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).



Cazul 2. $i \in V(P_{sj}^*)$. Fie $P_{sj}^* = P_{si} \circ P_{ij}$ cele două drumuri determinate de nodul i pe 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 -

Astfel partea " \Leftarrow " este demonstrată.

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 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).

" \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} satisfacă $a(P_{ss}) = 0 = u_s$. 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 satisfacă (B)

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

Algoritmul determină drumul $P : (s =)i_{k+1}, i_{k+1}i_k, i_k, \dots, i_1, i_1i_0, i_0 (= i)$ cu $P \in \mathcal{P}_{si}$ și

$$a(P) = a(i_{k+1}i_k) + \dots + a(i_1i_0) =$$

$$(u_{i_k} - u_{i_{k+1}}) + (u_{i_{k-1}} - u_{i_k}) + \dots + (u_{i_0} - u_{i_1}) =$$

$$u_{i_0} - u_{i_{k+1}} = u_i - u_s = u_i,$$

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

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

Atunci $u_i > \bar{u}_i = \bar{u}_{i_1} + a_{i_1 i} = u_{i_1} + a_{i_1 i} \geq u_i$ (prima inegalitate are loc datorită modului de alegere a lui i , a doua inegalitate are loc deoarece u_i satisface (B)).

Contradicția găsită arată că $u = \bar{u}$, i. e., elementele lui u sunt costuri minime.



Drumuri de cost minim - Rezolvarea problemei P2

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). Denumirile 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 $before[i] =$ nodul dinaintea lui i pe un drum de cost minim de la s la i . Nodurile de pe acest drum pot fi determinate în $\mathcal{O}(n)$ prin construcția secvenței $i, before[i], before[before[i]], \dots, s$.

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

Remarci 4

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

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

Drumuri de cost minim - Rezolvarea problemei P2

Remarci 5

- Dacă, în problemele P1 - P3, G este un graf și nu un digraf, putem folosi algoritmii 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 * C.

Remarci 6

- Deoarece mulțimile \mathcal{P}_{ij} sunt finite (și nevide), putem considera probleme similare cu P1 - P3 înlocuind *min* cu *max*.
- Relația evidentă $\max_{x \in A} x = -\min_{x \in A} (-x)$, prin înlocuirea costurilor a_{ij} cu $-a_{ij}$, se poate folosi doar în digrafuri în care, pentru fiecare circuit C , avem $a(C) \leq 0$ (în particular, această abordare merge pentru digrafuri fără circuite). **Dacă digraful are circuite, problema celui mai lung drum este, în general, NP-hard.**

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

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 Algo109ns * C. Croitoru - Graph Algorithms *

Exercițiu 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 incidentă 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.

Exercitii pentru seminarul de săptămâna viitoare

Exercițiu 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 incidentă 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).

Exercițiu 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 incidentă 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).

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

Exerciții pentru seminarul de săptămâna viitoare

Exercițiu 7. Fie M_G matricea de incidentă 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

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

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

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

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

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

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

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

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

Exercițiu 8. Fie $G = (V, E)$ un graf cu n noduri și $m \geq 1$ muchii. Considerăm următorul algoritm:

```
 $G' \leftarrow G;$ 
while ( $\exists u \in V(G')$  a. 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 .

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

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

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

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

Exercițiu 10. Arătați că o parcuregere DFS poate fi utilizată pentru a determina un circuit par într-un graf 3-regulat în $\mathcal{O}(n)$.

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

Exercițiu 11.

- Arătați că pentru un graf bipartit cu n noduri și m muchii avem $4m \leq n^2$.
- 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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

Exercițiu 12. Demonstrați că un graf G este bipartit dacă și numai dacă orice subgraf induc H al lui G satisface inegalitatea: $2\alpha(H) \geq |H|$.

Exercițiu 13. 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

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

- Arătați că G este S -lanț dacă și numai dacă este T -lanț.
- 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 $\mathcal{O}(n + m)$.

Exerciții pentru seminarul de săptămâna viitoare

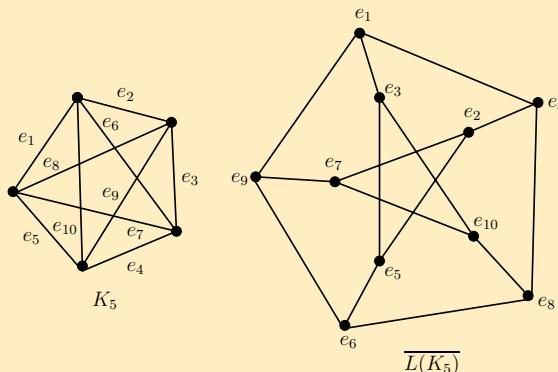
Exercițiu 14. Fie G un graf; notăm cu $b(G)$ graful obținut din G prin inserarea simultană a câte unui nou nod pe fiecare muchie a lui G .

- 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.

Exercițiu 15. Fie G un graf bipartit; arătați că G este conex dacă și numai dacă admite o singură bipartiție cu mulțimi stabile.

Exercițiul 2. Soluție.

- Să presupunem prin reducere la absurd că există o permutare a $E(K_5)$, e_1, e_2, \dots, e_{10} , aşa încât oricare două muchii consecutive (dar și e_1 și e_{10} also) nu sunt adiacente.
 - Astfel complementul grafului reprezentativ al muchiilor lui K_5 , $\overline{L(K_5)}$, are un circuit hamiltonian.
 - $L(K_5)$ are 10 noduri și este 6-regulat (de ce?), deci $\overline{L(K_5)}$ este 3-regulat (vezi figura).



Exercitii rezolvate (partial)

- Un graf hamiltonian și 3-regulat admite o 3-colorare a muchiilor.
 - Cum se obține o astfel de colorare?
 - Dar $\overline{L(K_5)}$ este graful lui Petersen (verificați!) care nu admite o astfel de colorare (pentru o foarte scurtă demonstrație: <https://www.sciencedirect.com/science/article/pii/S0012365X03001389>).

Exercițiu 3. Soluție. Fie

$$B = A + QQ^T = (b_{uv})_{u,v \in V(G)},$$

unde \vec{G} este orientarea aleasă a lui G . Fie u și v două noduri ale lui G .

$$b_{uv} = a_{uv} + \sum_{e \in E(\vec{G})} q_{ue} q_{ve}$$

- când $u \neq v$ și $uv \notin E(G)$ ($a_{uv} = 0$), pentru orice arc $e \in E(\vec{G})$, produsul $q_{ue} q_{ve} = ?$
- când $u \neq v$ și $uv \in E(G)$ ($a_{uv} = 1$ și există un singur arc, e_0 , incident cu amândouă nodurile u și v), pentru orice arc $e \in E(\vec{G})$, $e \neq e_0$, produsul $q_{ue} q_{ve} = ?$ iar produsul $q_{ue_0} q_{ve_0} = -1$. De ce?

Exerciții rezolvate (partial)

- când $u = v$ avem $b_{uu} = \sum_{e \in E(\vec{G})} q_{ue}^2$ și

$$q_{ue}^2 = \begin{cases} 1, & \text{dacă } e \text{ este incident cu } u \\ 0, & \text{altfel} \end{cases}$$

- astfel $b_{uu} =$ numărul de arce incidente cu $u = ?$ (în termenii numărului de muchii incidente cu u în G).

Exercițiu 4. Soluție.

- Pe fiecare coloană a lui B avem o valoare de 1, o valoare de -1 și 0 în rest.
- Fie M o submatrice pătratică a lui B .
- Demonstrăm prin inducție după k ordinul matricii M . Pentru $k = 1$, M este un element al lui B , deci $\det(M) \in \{-1, 0, 1\}$.
- În pasul inductiv presupunem că, $\det(M) \in \{-1, 0, 1\}$, pentru orice submatrice $k \times k$, M , a lui B ; fie M o submatrice $(k+1) \times (k+1)$. Sunt trei cazuri:

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

1. o întreagă coloană a lui M este nulă, astfel $\det(M) = ?$;
2. există o coloană h care conține un singur element nenul: $m_{ih} \in \{1, -1\}$, atunci

$$\det(M) = \sum_{j=1}^{k+1} m_{jh} (-1)^{j+h} \cdot \det(M_{jh}) = m_{ih} (-1)^{i+h} \cdot \det(M_{ih}),$$

unde $\det M_{ih}$ este minorul matricii B obținu prin stergerea liniei i și a coloanei h din matricea B , astfel, din ipoteza inductivă $\det(M_{ih}) \in \{-1, 0, 1\}$ și $\det(M) \in \{-1, 0, 1\}$;

3. pe fiecare coloană a lui M avem un 1 și un -1 (restul elementelor rămase fiind 0); adunând toate liniile la una singură obținem o linie nulă (liniile lui M sunt liniar dependente). Astfel $\det(M) = ?$

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

Exercițiu 5. Soluție: similară cu cea de mai sus cu excepția faptului că matricea B nu este neapărat pătratică iar fiecare coloana a sa conține exact două valori nenule (egale cu 1) ceea ce modifică demonstrația ultimului caz de mai sus (**cum?**).

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiu 6. Soluție. Fie $D = (V, E)$ un digraf; următorul algoritm construiește o ordonare aciclică a nodurilor lui D (dacă o astfel de ordonare există):

```
i ← 0;  
while(∃ x ∈ V with  $d_D^-(x) = 0$ ) {  
    i ++;  
     $v_i = x$ ;  
     $V \leftarrow V \setminus \{x\}$ ;  
    for ( $v \in A^+(x)$ )  
         $d_D^-(v) --$ ;  
}
```

- Demonstrația se bazează pe următorul fapt: într-un digraf aciclic există noduri de grad interior 0 (dar și noduri de grad exterior 0).

De ce?

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

- “ \implies ” Dacă D are o singură ordonare aciclică, atunci nu conține
circuite iar în bucla *while* de mai sus la orice iterație există un
singur nod de grad interior 0. **De ce?**
- La iterarea i , v_{i+1} are gradul interior 1, astfel $v_i v_{i+1} \in E$, $\forall i =$
 $1, n - 1$; obținem de aici drumul hamiltonian dorit.
“ \Leftarrow ” Dacă v_1, v_2, \dots, v_n nodurile unui drum hamiltonian (în
această ordine), atunci algoritmul de mai sus va construi aceeași
aceeași ordonare aciclică. **De ce?**

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiul 7. Soluție.

- (a) Ordăm nodurile (coloanele) de la frunze spre rădăcină (putem transforma arborele într-unul cu rădăcină printr-o parcurgere **bfs**) nivel cu nivel. Stergem coloana corespunzătoare rădăcinii.
- Definim următoarea ordonare a muchiilor (liniilor) și nodurilor (coloanelor): prima coloană corespunde unei frunze iar prima linie muchiei incidente cu acea frunză, a doua coloană corespunde unei frunze din arborele obținu prin stergerea frunzei anterioare iar a doua linie muchiei incidente cu acea frunză etc.
 - Matricea rezultată este superior triunghiulară și are doar valori de 1 pe diagonală. **De ce?**

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo118ns * C. Croitoru - Graph Algorithms *

$$(b) \text{ Fie } V(C) = \{x_1, x_2, \dots, x_{2p+1}\} \text{ si } E(C) = \{x_1x_2, x_2x_3, \dots, x_{2p+1}x_1\}.$$

- Descrieți M_C și calculați-i determinantul.

Exerciții rezolvate (partial)

Exercițiu 8. Soluție. (a)

```

 $G' \leftarrow G;$ 
calculează ( $d_{G'}$ ) pentru orice  $u \in V(G)$ ; //  $\mathcal{O}(n + m)$ , de ce?;
for ( $k = \overline{1, n - 1}$ ) do
     $B_k \leftarrow \emptyset$ ; //  $\mathcal{O}(n)$ ;
for ( $u \in V(G)$ ) do
     $mark[u] \leftarrow 1$ ;  $push(B_{d_G(u)}, u)$ ; //  $\mathcal{O}(n)$ , ca la BucketSort, de ce?;
let  $d = \delta(G)$ ; //  $\mathcal{O}(n)$ ;
while ( $d < m/n$ ) do
     $u \leftarrow pop(B_d)$ ;  $mark[u] = 1$ ; //  $\mathcal{O}(1)$ ;
    for ( $x \in A_{G'}(u)$ ) do
        if ( $mark[x] = 0$ ) then
             $push(B_{d_G(x)-1}, x)$ ; //  $\mathcal{O}(1)$ ;
             $delete(A_{G'}(x), u)$ ; //  $\mathcal{O}(1)$ , de ce?;
    let  $d = \min \{h : B_h \neq \emptyset\}$ ; //  $\mathcal{O}(1)$ ;
return  $G'$ ; // prin listele de adiacență;

```

- (b) La fiecare pas al algoritmului ștergem un nod și mai puțin de m/n muchii (de ce?), deci numărul total de muchii eliminate este strict mai mic decât $n \cdot m/n = m$. Astfel G' nu poate fi nul.
- (c) $G' \subseteq G$ și orice nod u din G' are $d_{G'}(u) \geq m/n$.

```

let  $x \in V(G')$ ;  $V(P) \leftarrow \{x\}$ ;  $E(P) \leftarrow \emptyset$ ;
while ( $N_G(x) \neq \emptyset$ ) do
    let  $u \leftarrow \text{pop}(N_G(x))$ ;
     $V(P) \leftarrow V(P) \cup \{x\}$ ;
     $E(P) \leftarrow E(P) \cup \{xu\}$ ;
     $x \leftarrow u$ ;

```

Drumul construit în acest fel are lungimea cel puțin m/n . De ce?

Exerciții rezolvate (partial)

Exercițiul 11. Soluție.

- (a) Numărul de muchii dintr-un graf bipartit este egal cu suma gradelor nodurilor dintr-o clasă a bipartiției. De ce?
- Dacă $G = (S, T; E)$ is a bipartite graph:

$$\begin{aligned}
m = |E(G)| &= \sum_{v \in S} d_G(v) \leq \sum_{v \in S} |T| = |S| \cdot |T| \stackrel{\text{de ce?}}{\leq} \\
&\leq \frac{(|S| + |T|)^2}{4} = ?
\end{aligned}$$

- (b) Dacă \overline{G} este un graf bipartit, atunci

$$m = |E(G)| = \frac{n(n-1)}{2} - |E(\overline{G})| \geq \frac{n(n-1)}{2} - \frac{n^2}{4} = \frac{n(n-2)}{4}$$

- Dacă această inegalitate nu are loc, atunci complementul lui G nu poate fi bipartit.

- Altfel, dimensiunea lui G , $m = \mathcal{O}(?)$ ceea ce înseamnă că $\mathcal{O}(n + m) = \mathcal{O}(?)$.
- Putem construi liste de adiacență ale lui \overline{G} din cele ale lui G în $\mathcal{O}(n + m)$. Putem aplica apoi un algoritm simplu de recunoaștere a grafurilor bipartite asupra lui \overline{G} în $\mathcal{O}(|V(\overline{G})| + |E(\overline{G})|) = \mathcal{O}(?)$. De ce?
- Algoritmul:

```
bip ← true;
if (4m < n(n - 2))
    bip ← false;
else {
    construiește liste de adiacență ale lui  $\overline{G}$ ;
    bip ← bipartiteness( $\overline{G}$ );
}
```

Exerciții rezolvate (partial)

Exercițiul 12. Soluție.

- “ \implies ” Dacă H este un subgraf al unui graf bipartit, atunci H este bipartit (de ce?); $H = (S, T; E)$ și $|H| = |S| + |T| \leq 2\alpha(H)$ (de ce?).
- “ \impliedby ” Să presupunem G nu este bipartit, atunci G conține un circuit impar (nu neapărat induc): C cu $|V(C)| = 2k + 1$.
- Fie $H = [V(C)]_G$ subgrafenul induc de nodurile lui C . Din ipoteza știm că $\alpha(H) \geq \frac{1}{2}|H|$; dacă ștergem din H toate muchiile care nu sunt în C , obținem graful C_{2k+1} , astfel $\alpha(C_{2k+1}) \geq \alpha(H) \geq \frac{2k+1}{2}$.
- Urmează că $\alpha(C_{2k+1}) \geq \alpha(H) \geq k + 1$. Dar $\alpha(C_{2k+1}) = k$ (de ce?) - contradicție.

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

Exercițiu 13. Soluție.

- (a) Fie $G = (S, T; E)$ un graf bipartit care este și S -lanț; presupunem prin reducere la absurd că G nu ar fi T -lanț, atunci există două noduri $u, v \in T$ astfel ca $N_G(u) \not\subseteq N_G(v)$ și $N_G(v) \not\subseteq N_G(u)$ (**de ce?**);
- Putem alege $x \in N_G(u) \setminus N_G(v)$ și $y \in N_G(v) \setminus N_G(u)$; urmează că $u \in N_G(x) \setminus N_G(y)$ și $v \in N_G(y) \setminus N_G(x)$;
 - Cum G este un S -lanț, $(x, y \in S)$ $N_G(x) \subseteq N_G(y)$ sau $N_G(y) \subseteq N_G(x)$ - contradicție.

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

- (b) Sortăm nodurile lui S în ordinea descrescătoare a gradelor (in $\mathcal{O}(n)$)
- **de ce?** - folosind **bucketSort** sau **radixSort**):

$$d_G(x_1) \geq d_G(x_2) \geq \dots \geq d_G(x_k)$$

- Apoi aplicăm următoarea procedură (de complexitate timp $\mathcal{O}(n + m)$), folosind observația că dacă G este S -lanț și $u, v \in S$ cu $d_G(u) \geq d_G(v)$, atunci trebuie să avem $N_G(u) \supseteq N_G(v)$ (**de ce?**):

Observație: pentru a face demonstrația independentă de alegerea bipartiției, se poate demonstra că G este S -lanț dacă și numai dacă este $2K_2$ -free. **Cum?**

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo122ns * C. Croitoru - Graph Algorithms *

```

chain ← true;
for  $u \in V$ 
    neighbour[u] ← 0;
for  $u \in A(x_1)$ 
    neighbour[u] ← 1;
for  $i = 2$  to  $k$ 
    for  $u \in A(x_i)$ 
        if ( $neighbour[u] \neq i - 1$ ) {
            chain ← false;
            return;
        }
    else
        neighbour[u] ←  $i$ ;

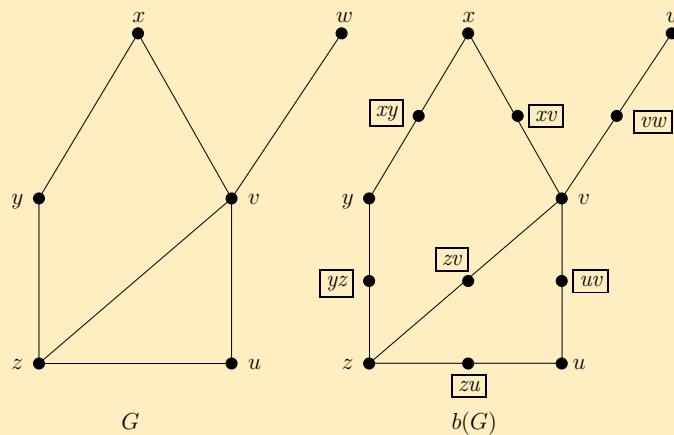
```

Dacă $chain$ rămâne true, atunci $N_G(x_1) \supseteq N_G(x_2) \supseteq \dots \supseteq N_G(x_k)$, altfel, folosind observația de mai sus G nu poate fi S -lanț. **De ce?**

Exerciții rezolvate (partial)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 14. Soluție. Fie $G = (V, E)$; notăm cu \boxed{uv} noul nod inserat pe muchia uv (vezi mai jos).



Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 - Graph Algorithms * C. Croitoru - Graph Algo123ns * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 $\widehat{V} = \{\boxed{uv} : uv \in E(G)\}$; atunci $V(b(G)) = V \cup \widehat{V}$ și
 $E(b(G)) = \bigcup_{uv \in E} \{u\boxed{uv}, v\boxed{uv}\}$.

- (a) $b(G)$ este un graf bipartit cu bipartiția (V, \widehat{V}) : V și \widehat{V} sunt
mulțimi stable în $b(G)$. De ce?

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

Arbore cu rădăcină = digraf G obținut dintr-un arbore $T = (V, E)$:

- alegem un nod $s \in V$ = **rădăcina** digrafului
 - toate muchiile se orientează să meargă dinspre părinte spre fiu

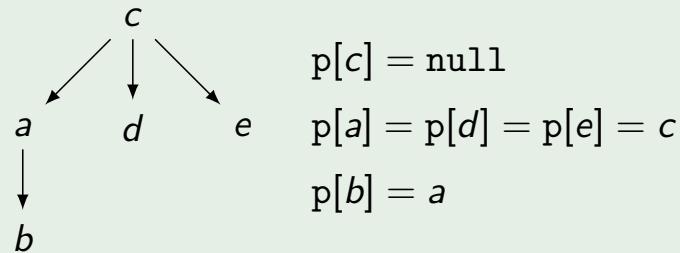
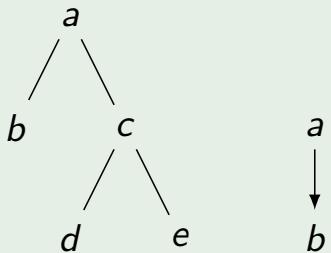
Reprezentarea cu predecesor:

$p[s] = \text{null}$

dacă $x \in V - \{s\}$: $p[x]$ = părintele, sau predecesorul direct al lui x

Exemplu

Arbore cu rădăcina c



`p[c] = null`

$$p[a] = p[d] = p[e] = c$$

$$p[b] = a$$

Traversarea grafurilor

Traversarea în adâncime (depth first search) de la un nod sursă

Se vizitează nodul s.

- Vizitarea unui nod x se face astfel:
 - ① Se marchează x ca nod vizitat.
 - ② Se vizitează recursiv toți vecinii nevizitați ai lui x . De obicei, pentru fiecare vecin y care se vizitează se setează $p[y] = x$ pentru a reține faptul că traversarea s-a făcut de la x la y .

Traversarea grafurilor

Traversarea în adâncime (depth first search) de la un nod sursă [mp to Table of contents](#)

Se vizitează nodul s .

- Vizitarea unui nod x se face astfel:
 - ① Se marchează x ca nod vizitat.
 - ② Se vizitează recursiv toți vecinii nevizitați ai lui x . De obicei, pentru fiecare vecin y care se vizitează se setează $p[y] = x$ pentru a reține faptul că traversarea s-a făcut de la x la y .

⇒ un arbore de traversare în adâncime cu rădăcina s .

Traversarea grafurilor

Traversarea în adâncime (depth first search) de la un nod sursă

Se vizitează nodul s .

- Vizitarea unui nod x se face astfel:
 - ① Se marchează x ca nod vizitat.
 - ② Se vizitează recursiv toți vecinii nevizitați ai lui x . De obicei, pentru fiecare vecin y care se vizitează se setează $p[y] = x$ pentru a reține faptul că traversarea s-a făcut de la x la y .

⇒ un arbore de traversare în adâncime cu rădăcina s .

```
private void dfs(Graph G, int i) { // inițiază traversarea de la i
    vizitat[i] = true;
    for (int j : G.adj(i))
        if (!vizitat[j]) {
            p[j] = i;      // reține că s-a traversat muchia i-j
            dfs(G,j);
        }
}
```

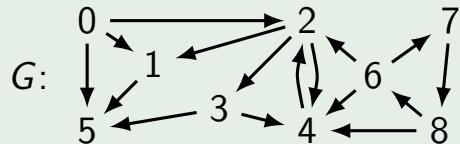
Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

[Jump to Table of contents](#)

- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



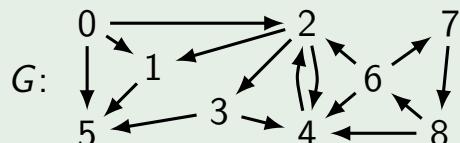
Cursul 9

Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.



\Rightarrow arborele de traversare în adâncime:



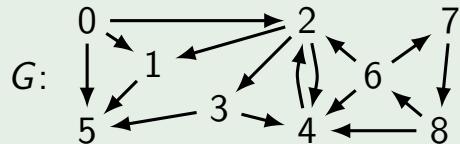
Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

[Jump to Table of contents](#)

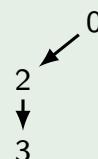
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



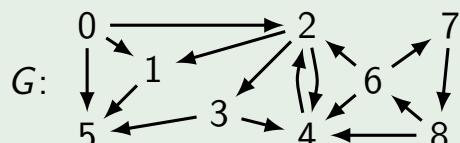
Cursul 9

Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

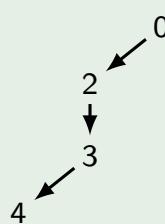
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



128

Cursul 9

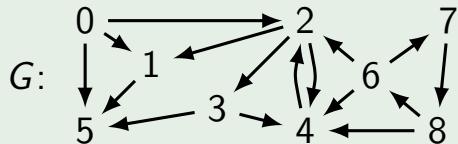
Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

[Jump to Table of contents](#)

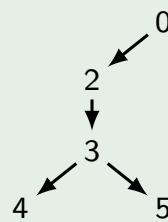
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



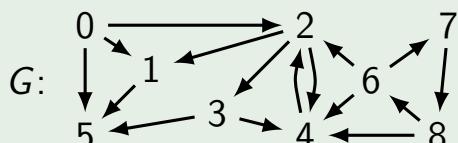
Cursul 9

Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

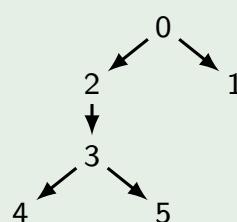
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



129

Cursul 9

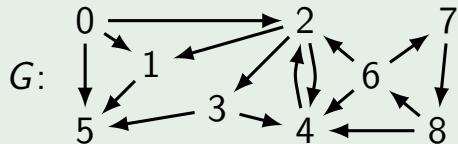
Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

[Jump to Table of contents](#)

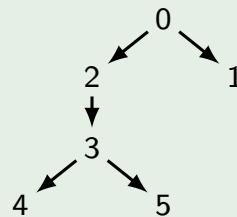
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



- $\{x \mid 0 \rightsquigarrow x\} = \{0, 1, 2, 3, 4, 5\}$



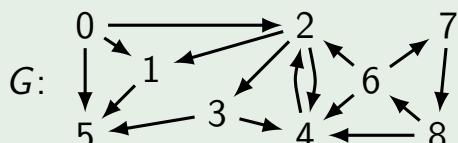
Cursul 9

Traversarea grafurilor

Proprietăți ale traversării în adâncime de la un nod s

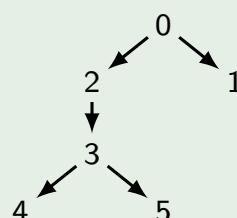
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x .

Exemplu



$\text{adj}[0] = [2, 1, 5]$, $\text{adj}[1] = [5]$, $\text{adj}[2] = [3, 1, 4]$,
 $\text{adj}[3] = [4, 5]$, $\text{adj}[4] = [2]$, $\text{adj}[5] = []$,
 $\text{adj}[6] = [4, 2]$, $\text{adj}[7] = [8]$, $\text{adj}[8] = [6]$.

\Rightarrow arborele de traversare în adâncime:



- $\{x \mid 0 \rightsquigarrow x\} = \{0, 1, 2, 3, 4, 5\}$
- s-au găsit drumurile $[0], [0, 1], [0, 2], [0, 2, 3], [0, 2, 3, 4], [0, 2, 3, 5]$



Cursul 9

Traversarea în lățime de la un nod sursă s se face în runde:

- în prima rundă vizităm s
- în fiecare rundă următoare vizităm vecinii nevizitați ai nodurilor vizitate în runda precedentă.

```
private void bfs(Graph G, int s) {  
    vizitat[s] = true;           // vizitează nodul sursă  
    Queue<Integer> Q = new Queue<Integer>();  
    Q.enqueue(s);               // și il pune în coadă  
    while(!Q.isEmpty()) {  
        int v = Q.dequeue();  
        for(int w : G.adj(v))  
            if(!vizitat[w]) {  
                p[w] = v;  
                vizitat[w] = true;  
                Q.enqueue(w);  
            }  
    }  
}
```



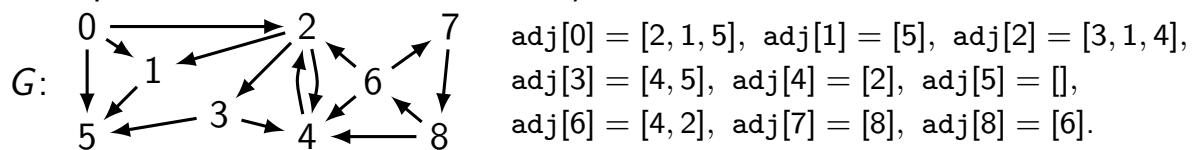
Traversarea în lățime

Proprietăți

- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x .

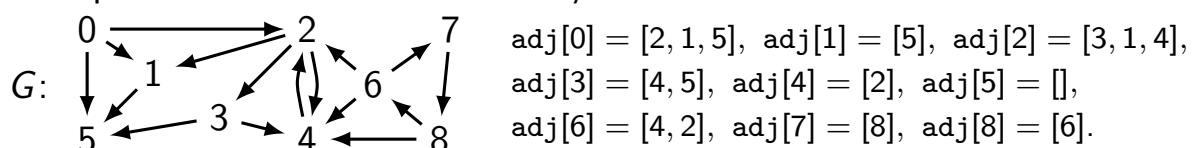
- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x .

Exemplu ilustrat de traversare în lățime de la nodul 0



- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x .

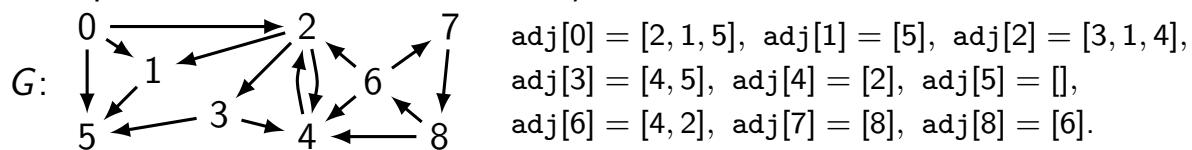
Exemplu ilustrat de traversare în lățime de la nodul 0



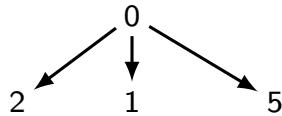
\Rightarrow arborele de traversare în lățime:

- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x .

Exemplu ilustrat de traversare în lățime de la nodul 0

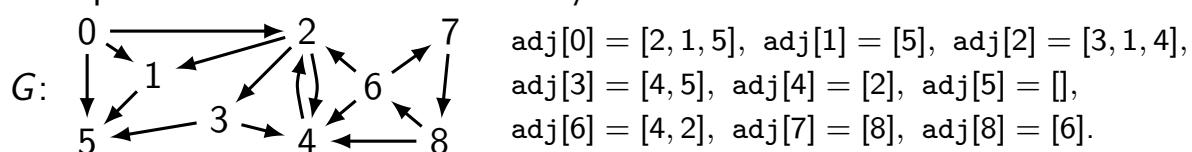


\Rightarrow arborele de traversare în lățime:

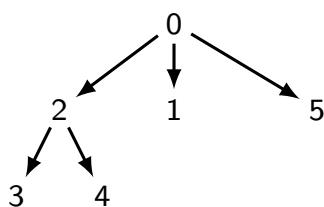


- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x .

Exemplu ilustrat de traversare în lățime de la nodul 0

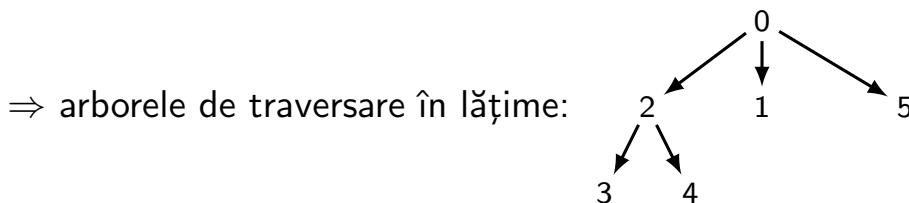
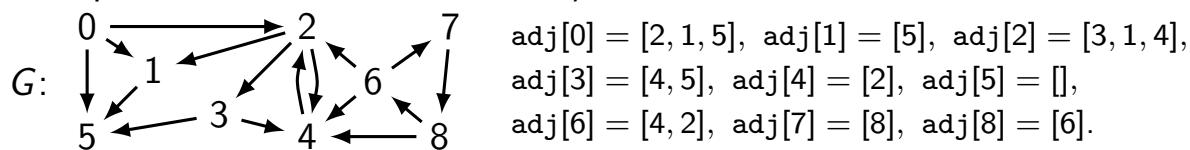


\Rightarrow arborele de traversare în lățime:



- ① Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
- ② $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x .

Exemplu ilustrat de traversare în lățime de la nodul 0



Cele mai scurte căi găsite sunt [0], [0, 2], [0, 2, 3], [0, 2, 4], [0, 1], [0, 5].

Ordini de traversare în adâncime

Traversarea în adâncime a tuturor nodurilor unui graf

- Produce o **pădure** de arbori de traversare în adâncime
- Definește trei ordini de traversare:

Preordine: se adaugă nodurile într-o coadă înaintea apelului recursiv al lui `dfs()`. Avem $x <_{\text{pre}} y$ dacă x apare înaintea lui y în coadă.

Postordine: se adaugă nodurile în o coadă de noduri după apelul recursiv al lui `dfs()`. Avem $x <_{\text{post}} y$ dacă x apare înaintea lui y în coadă.

Postordine inversă: Avem $x <_{\text{revpost}} y$ dacă $y <_{\text{post}} x$. Deci postordinea inversă se poate calcula punând nodurile în o stivă după apelul recursiv al lui `dfs()`. Avem $x <_{\text{revpost}} y$ dacă x apare deasupra lui y în stivă.

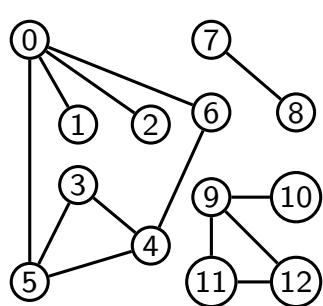
Ordini de traversare în adâncime

Exemplu ilustrat

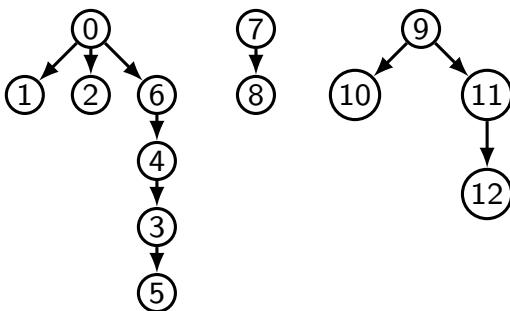
[Jump to Table of contents](#)

$\text{adj}[0] = [1, 2, 5, 6]$, $\text{adj}[1] = [0]$, $\text{adj}[2] = [0]$, $\text{adj}[3] = [4, 5]$,
 $\text{adj}[4] = [3, 5, 6]$, $\text{adj}[5] = [0, 3, 4]$, $\text{adj}[6] = [0, 4]$, $\text{adj}[7] = [8]$,
 $\text{adj}[8] = [7]$, $\text{adj}[9] = [10, 11, 12]$, $\text{adj}[10] = [9]$, $\text{adj}[11] = [9, 12]$,
 $\text{adj}[11] = [9, 12]$, $\text{adj}[12] = [9, 11]$.

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



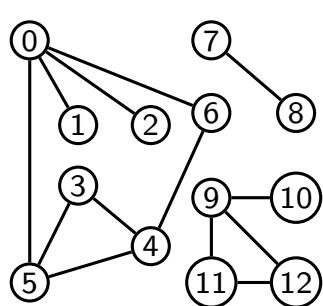
Cursul 9

Ordini de traversare în adâncime

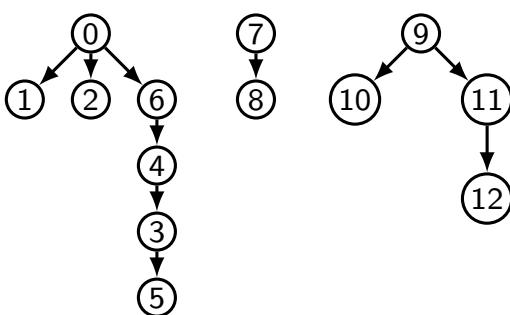
Exemplu ilustrat

$\text{adj}[0] = [1, 2, 5, 6]$, $\text{adj}[1] = [0]$, $\text{adj}[2] = [0]$, $\text{adj}[3] = [4, 5]$,
 $\text{adj}[4] = [3, 5, 6]$, $\text{adj}[5] = [0, 3, 4]$, $\text{adj}[6] = [0, 4]$, $\text{adj}[7] = [8]$,
 $\text{adj}[8] = [7]$, $\text{adj}[9] = [10, 11, 12]$, $\text{adj}[10] = [9]$, $\text{adj}[11] = [9, 12]$,
 $\text{adj}[11] = [9, 12]$, $\text{adj}[12] = [9, 11]$.

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



Preordine: [0, 1, 2, 6, 4, 3, 5, 7, 8, 9, 10, 11, 12]

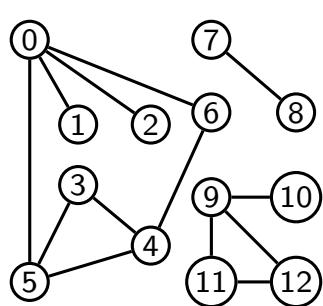
Ordini de traversare în adâncime

Exemplu ilustrat

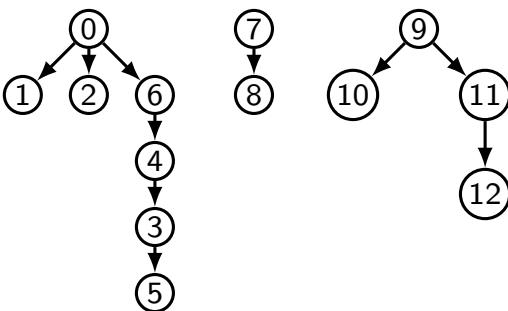
[Jump to Table of contents](#)

$\text{adj}[0] = [1, 2, 5, 6]$, $\text{adj}[1] = [0]$, $\text{adj}[2] = [0]$, $\text{adj}[3] = [4, 5]$,
 $\text{adj}[4] = [3, 5, 6]$, $\text{adj}[5] = [0, 3, 4]$, $\text{adj}[6] = [0, 4]$, $\text{adj}[7] = [8]$,
 $\text{adj}[8] = [7]$, $\text{adj}[9] = [10, 11, 12]$, $\text{adj}[10] = [9]$, $\text{adj}[11] = [9, 12]$,
 $\text{adj}[11] = [9, 12]$, $\text{adj}[12] = [9, 11]$.

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



Preordine: [0, 1, 2, 6, 4, 3, 5, 7, 8, 9, 10, 11, 12]

Postordine: [1, 2, 5, 3, 4, 6, 0, 8, 7, 10, 12, 11, 9]



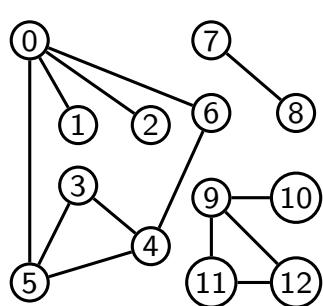
Cursul 9

Ordini de traversare în adâncime

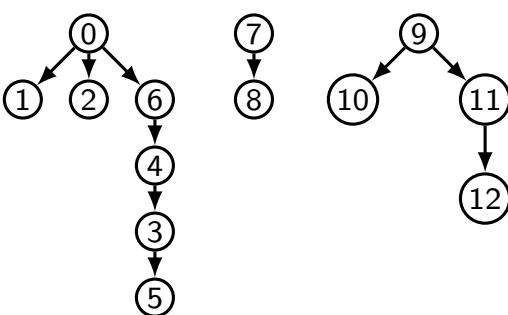
Exemplu ilustrat

$\text{adj}[0] = [1, 2, 5, 6]$, $\text{adj}[1] = [0]$, $\text{adj}[2] = [0]$, $\text{adj}[3] = [4, 5]$,
 $\text{adj}[4] = [3, 5, 6]$, $\text{adj}[5] = [0, 3, 4]$, $\text{adj}[6] = [0, 4]$, $\text{adj}[7] = [8]$,
 $\text{adj}[8] = [7]$, $\text{adj}[9] = [10, 11, 12]$, $\text{adj}[10] = [9]$, $\text{adj}[11] = [9, 12]$,
 $\text{adj}[11] = [9, 12]$, $\text{adj}[12] = [9, 11]$.

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



Preordine: [0, 1, 2, 6, 4, 3, 5, 7, 8, 9, 10, 11, 12]

Postordine: [1, 2, 5, 3, 4, 6, 0, 8, 7, 10, 12, 11, 9]

Postordine inversă: [9, 11, 12, 10, 7, 8, 0, 6, 14, 13, 5, 2, 1]



Cursul 9

Aplicații ale traversării în adâncime

1. Detectia componentelor conexe în grafuri neorientate

[Jump to Table of contents](#)

public class CC	
CC(Graph G)	Constructor pentru componentele conexe ale grafului neorientat G
boolean connected(int i,int j)	Există drum de la i la j?
int count()	Numărul de componente conexe
int id(int v)	Identifierul de componentă al nodului v (între 0 și count()-1)


```
public class CC {  
    private boolean[] vizitat;  
    private int[] id;  
    private int count;  
    public CC(Graph G) {  
        vizitat = new boolean[G.V()]; id = new int[G.V()];  
        for (int s=0; s<G.V(); s++) if (!vizitat(s)) { dfs(G,s); count++; }  
    }  
    private void dfs(Graph G, int i) {  
        vizitat[i] = true; id[i] = count;  
        for (int j : G.adj(i)) if (!vizitat[j]) dfs(G,j);  
    }  
    ...  
}
```



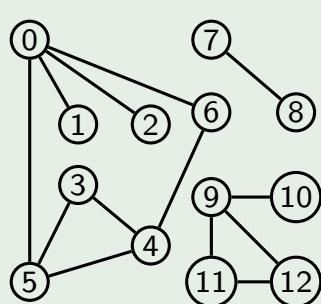
Cursul 9

Aplicații ale traversării în adâncime

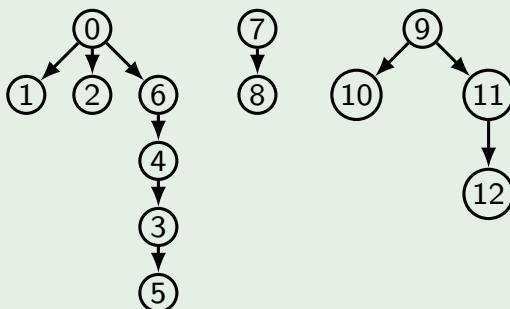
1. Detectia componentelor conexe în grafuri neorientate

Exemplu

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



- Nodurile $x \in \{0, 1, 2, 6, 4, 3, 5\}$ din primul arbore fac parte din prima componentă conexă; pentru ele setăm $\text{id}[x] = 0$.
- Nodurile $x \in \{7, 8\}$ din al doilea arbore fac parte din a doua componentă conexă; pentru ele setăm $\text{id}[x] = 1$.
- Nodurile $x \in \{9, 10, 11, 12\}$ din al treilea arbore fac parte din a treia componentă conexă,¹³⁷ pentru ele setăm $\text{id}[x] = 2$.



Cursul 9

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \rightarrow v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \rightarrow v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \rightarrow v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.



Aplicații ale traversării în adâncime

2. Detectia ciclurilor în digrafuri

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \rightarrow v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \rightarrow v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \rightarrow v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Observații

Aplicații ale traversării în adâncime

2. Detectia ciclurilor în digrafuri

[Jump to Table of contents](#)

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \rightarrow v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \rightarrow v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \rightarrow v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Observații

- 1 G are un ciclu dacă și numai dacă are o muchie de întoarcere.

Cursul 9

Aplicații ale traversării în adâncime

2. Detectia ciclurilor în digrafuri

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \rightarrow v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \rightarrow v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \rightarrow v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Observații

- 1 G are un ciclu dacă și numai dacă are o muchie de întoarcere.
- 2 Vom folosi abrevierea DAG (engl. *directed acyclic graph*) pentru un digraf fără cicluri.

139

Cursul 9

Aplicații ale traversării în adâncime

2. Detectia ciclurilor în digrafuri

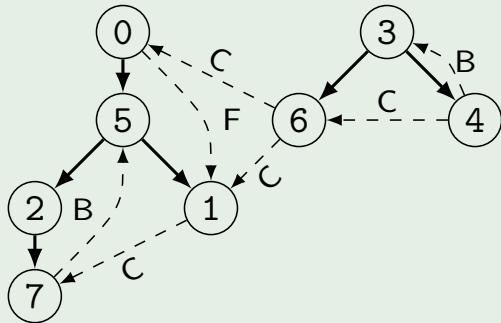
[Jump to Table of contents](#)

Exemplu

Traversarea în adâncime a digrafului reprezentat cu

$$\begin{aligned} \text{adj}[0] &= [5, 1], & \text{adj}[1] &= [7], & \text{adj}[2] &= [7], & \text{adj}[3] &= [6, 4], \\ \text{adj}[4] &= [3, 6], & \text{adj}[5] &= [2, 1], & \text{adj}[6] &= [0, 1], & \text{adj}[7] &= [5]. \end{aligned}$$

produce 2 arbori de căutare:



Muchiile de întoarcere au fost etichetate cu B, cele de salt înainte cu F, și cele transversale cu C.



Cursul 9

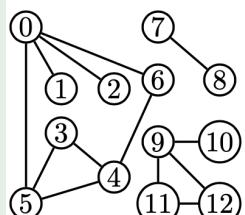
Aplicații ale traversării în adâncime

3. Detectia ciclurilor în grafuri neorientate

Traversarea în adâncime a tuturor nodurilor unui graf G produce o pădure de arbori de căutare în adâncime.

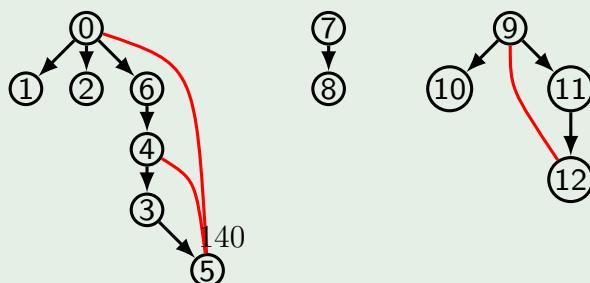
- Toate muchiile lui G care nu apar ca muchii în arbori sunt între un nod x și un predecesor indirect al lui x într-un arbore.

Exemplu



$$\begin{aligned} \text{adj}[0] &= [1, 2, 5, 6], \text{adj}[1] = [0], \text{adj}[2] = [0], \\ \text{adj}[3] &= [4, 5], \text{adj}[4] = [3, 5, 6], \text{adj}[5] = [0, 3, 4], \\ \text{adj}[6] &= [0, 4], \text{adj}[7] = [8], \text{adj}[8] = [7], \\ \text{adj}[9] &= [10, 11, 12], \text{adj}[10] = [9], \text{adj}[11] = [9, 12], \\ \text{adj}[12] &= [9, 11]. \end{aligned}$$

Traversarea în adâncime produce o pădure cu 3 arbori



Cursul 9



- În literatură, arborii de traversare în adâncime împreună cu muchiile de întoarcere se numesc **palmieri** (engl. *palm trees*).
- Un graf neorientat are un ciclu dacă și numai dacă există o muchie de întoarcere într-un palmier.
- Un API java de detectie a ciclurilor în grafuri neorientate este descris în materialul extins de curs.

```
public class CC
{
    ...
    boolean hasCycle(int c) Există ciclu în componenta cu identificatorul c?
    Iterable<Integer> cycle(int c) Nodurile din un ciclu al componentei
                                         conexe c (dacă există unul)
```

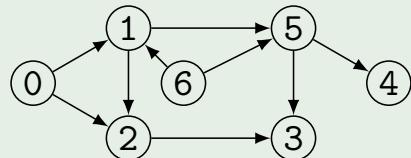
Aplicații ale traversării în adâncime

4. Sortarea topologică a unui DAG

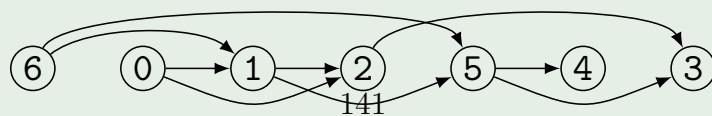
Sortare topologică a unui DAG $(V, E) = \text{enumerare } [x_1, x_2, \dots, x_n]$ a tuturor nodurilor din V astfel încât $\forall e \in E : e = (x_i \rightarrow x_j)$ cu $i < j$.

⇒ dacă redesenăm graful cu nodurile x_1, x_2, \dots, x_n ordonate de la stânga la dreapta pe o dreaptă imaginată, atunci toate arcele sunt orientate de la stânga la dreapta.

Exemplu (Un DAG cu o sortare topologică)



are sortarea topologică $[6, 0, 1, 2, 5, 4, 3]$:

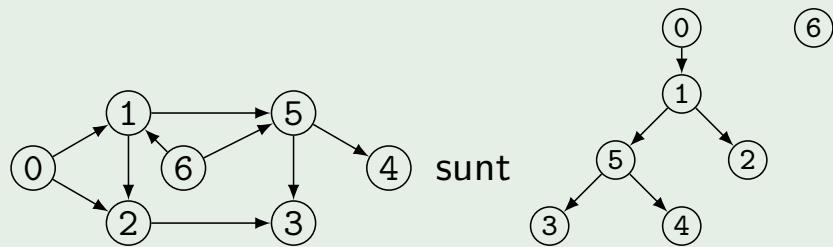


Observație

Enumerarea nodurilor în postordine inversă produsă de traversarea în adâncime este o sortare topologică.

Exemplu

Arborii de traversare în adâncime produși de traversarea în adâncime a digrafului



Postordine: [3, 4, 5, 2, 1, 0, 6]

Postordine inversă: [6, 0, 1, 2, 5, 4, 3]



Cursul 9

Aplicații ale traversării în adâncime

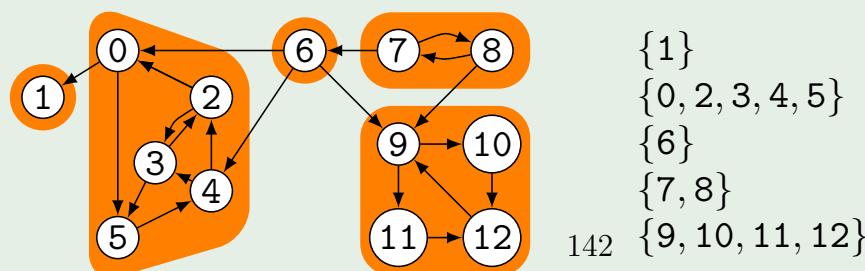
5. Detectia componentelor tare conexe

Fie $G = (V, E)$ un digraf. O componentă tare conexă a lui G este o clasă de echivalență a relației de echivalență

$x \sim_{sc} y$ dacă și numai dacă $x \rightsquigarrow y$ și $y \rightsquigarrow x$.

Exemplu (Componentele tare conexe ale unui digraf)

$\text{adj}[0] = [1, 5]$,	$\text{adj}[1] = []$,	$\text{adj}[2] = [0, 3]$,	$\text{adj}[3] = [2, 5]$,
$\text{adj}[4] = [2, 3]$,	$\text{adj}[5] = [4]$,	$\text{adj}[6] = [0, 4, 9]$,	$\text{adj}[7] = [6, 8]$,
$\text{adj}[8] = [7, 9]$,	$\text{adj}[9] = [10, 11]$,	$\text{adj}[10] = [12]$,	$\text{adj}[11] = [12]$,
$\text{adj}[12] = [9]$.			



Cursul 9

Algoritmul lui Kosaraju

- ① Calculează postordinea inversă a nodurilor din graful invers G^r .
- ② Traversează toate nodurile lui G în adâncime, însă în ordinea calculată în pasul 1.
- ③ Toate nodurile din un arbore de căutare în adâncime calculat în felul acesta formează o componentă tare conexă a lui G .



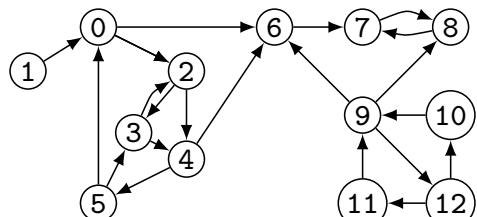
Cursul 9

Detectia componentelor tare conexe

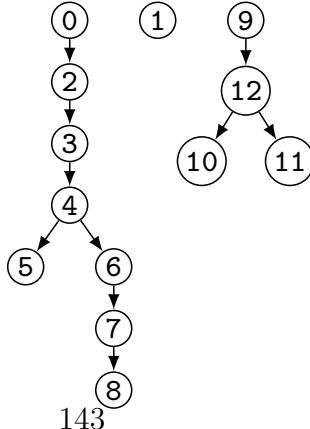
Algoritmul lui Kosaraju: exemplu ilustrat

Considerăm digraful G din exemplul precedent. Digraful invers G^r are reprezentarea

$$\begin{array}{llll} \text{adj}[0] = [2, 6], & \text{adj}[1] = [0], & \text{adj}[2] = [3, 4], & \text{adj}[3] = [2, 4], \\ \text{adj}[4] = [5, 6], & \text{adj}[5] = [0, 3], & \text{adj}[6] = [7], & \text{adj}[7] = [8], \\ \text{adj}[8] = [7], & \text{adj}[9] = [6, 8, 12], & \text{adj}[10] = [9], & \text{adj}[11] = [9], \\ \text{adj}[12] = [10, 11] & & & \end{array}$$



iar traversarea în adâncime produce arborii



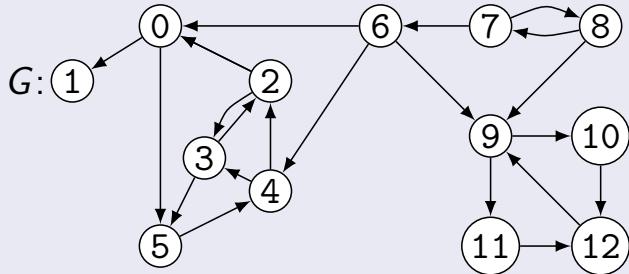
\Rightarrow postordinea inversă $[9, 12, 11, 10, 1, 0, 2, 3, 4, 6, 7, 8, 5]$



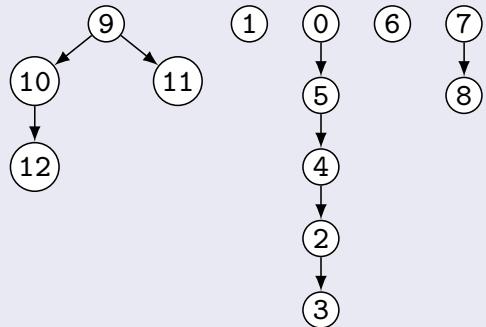
Cursul 9

Algoritmul lui Kosaraju ilustrat (continuare)

Traversarea în adâncime a lui



în ordinea [9, 12, 11, 10, 1, 0, 2, 3, 4, 6, 7, 8, 5] produce pădurea de arbori



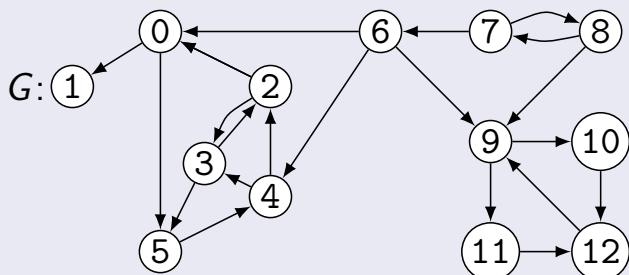
Cursul 9

Aplicații ale traversării în adâncime

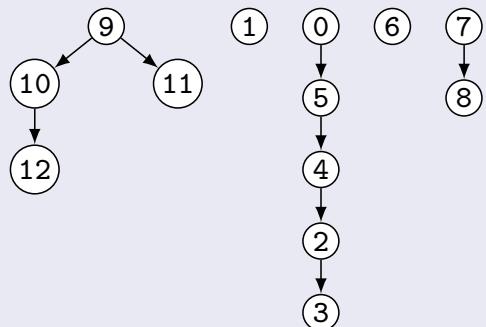
5. Detectia componentelor tare conexe

Algoritmul lui Kosaraju ilustrat (continuare)

Traversarea în adâncime a lui



în ordinea [9, 12, 11, 10, 1, 0, 2, 3, 4, 6, 7, 8, 5] produce pădurea de arbori



Rezultă componentele tare conexe
{9, 10, 11, 12}
{1}
{0, 5, 4, 2, 3}
{6}
{7, 8}

144



Cursul 9

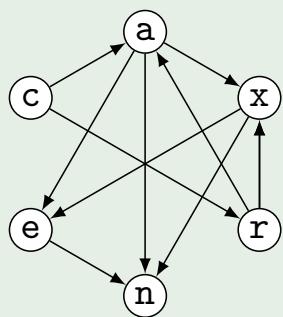
- Determinarea de drumuri elementare de lungime maximă de la un nod sursă în un DAG.
- ...

Exemplu

Listele de adiacență

$$\begin{array}{lll} \text{adj}[a] = [e, x, n], & \text{adj}[c] = [a, r], & \text{adj}[x] = [e, n], \\ \text{adj}[r] = [a, x], & \text{adj}[e] = [n], & \text{adj}[n] = [] \end{array}$$

reprezintă DAG-ul



Cursul 9

Alte aplicații ale traversării în adâncime

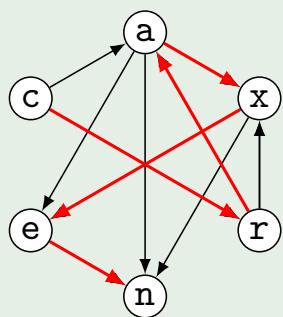
- Determinarea de drumuri elementare de lungime maximă de la un nod sursă în un DAG.
- ...

Exemplu

Listele de adiacență

$$\begin{array}{lll} \text{adj}[a] = [e, x, n], & \text{adj}[c] = [a, r], & \text{adj}[x] = [e, n], \\ \text{adj}[r] = [a, x], & \text{adj}[e] = [n], & \text{adj}[n] = [] \end{array}$$

reprezintă DAG-ul



Drumuri elementare de lungime maximă de la sursa c:

[c], [c, r],
[c, r, a], [c, r, a, x]
[c, r, a, x, e]
[c, r, a, x, e, n]¹⁴⁵



Cursul 9

4 CURS 4: Probleme de drum minim in (di)grafuri.

Dijkstra :

- ai cost negativ \implies nu poti aplica
- nu poti pleca din nodul de start nicaieri \implies nu poti aplica
- daca se poate aplica: alegi mereu nodul vecin nezitat cu costul cel mai mic si faci update la valoare; te opresti cand termini de vizitat toate nodurile.

Bellman & Warshall :

- accepta costuri negative,dar nu si cicluri negative
- cicluri negative = cicluri unde suma costurilor este negativa

ordonare topologica :

- faci o ordonare unde mereu pui inainte nodul din care pleci, apoi ala unde poti ajunge
- adica daca ai muchia xy inatai, pui x, apoi y; NU INVERS

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

23 octombrie 2020

Cuprins

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

1 Probleme de drumuri în digrafuri

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

2 Exerciții pentru seminarul de săptămâna următoare

3 Exerciții rezolvate (parțial)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 funcție injectivă $ord : V \rightarrow \{1, 2, \dots, n\}$ ($ord[u] = \text{numărul de ordine al nodului } u, \forall u \in V$) așa încât

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

Lema 1

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

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

Drumuri de cost minim - Problema P2 pentru dags

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

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

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

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

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

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

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

□

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

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

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

Drumuri de cost minim - Problema P2 pentru dags

```

construct the array  $d_G^-[u], \forall u \in V$ ;
count  $\leftarrow 0$ ;  $S \leftarrow \{u \in V : d_G^-[u] = 0\}$ ; //  $S$  este o coadă sau o stivă;
while ( $S \neq \emptyset$ ) do
     $v \leftarrow pop(S)$ ; count  $\leftarrow$  count + +;  $ord[v] \leftarrow count$ ;
    for ( $w \in A[v]$ ) do
         $d_G^-[w] \leftarrow d_G^-[w] - 1$ ;
        if ( $d_G^-[w] = 0$ ) then
            push( $S, w$ );
    // complexitatea timp  $\mathcal{O}(n + m)$ ;
    if (count  $= n$ ) then
        return "yes"  $ord$ ;
    return "no";

```

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

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

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

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

```

 $u_1 \leftarrow 0$ ;  $before[1] \leftarrow 0$ ;
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  $\mathcal{O}(n^2)$  sau chiar  $\mathcal{O}(n+m)$  dacă folosim liste
    // de adiacență în loc de matrice de cost-adiacență;

```

Drumuri de cost minim - Problema P2 pentru costuri nenegative

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

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

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

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

(D)

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

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

Algoritmul lui Dijkstra

```

 $S \leftarrow \{s\}; before[s] \leftarrow 0; u_s \leftarrow 0;$ 
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
             $u_j \leftarrow u_{j^*} + a_{j^*j}; before[j] \leftarrow j^*$ ;

```

Drumuri de cost minim - Problema P2 pentru costuri nenegative

Demonstrația corectitudinii algoritmului lui Dijkstra

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

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

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

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

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

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

Demonstrația corectitudinii algoritmului lui Dijkstra (cont.)

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

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

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

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

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

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

Drumuri de cost minim - Problema P2 pentru costuri nenegative

Demonstrația corectitudinii algoritmului lui Dijkstra (cont.)

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

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

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

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



Complexitatea timp a algoritmului lui Dijkstra

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

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

timpul general necesar algoritmului pentru a actualiza valorile u_j este

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

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

Drumuri de cost minim - Problema P2 pentru costuri nenegative

Complexitatea timp a algoritmului lui Dijkstra

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

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

Remarci 1

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

Drumuri de cost minim - Problema P2 pentru costuri nenegative

Remarci 2

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

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

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 folosește matricea de cost-adiacență. Pentru digrafuri foarte mari sau pentru digrafuri date printr-o funcție care returnează lista de adiacență a unui nod dat, implementările folosind matricea de cost-adiacență 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 și 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 *

Drumuri de cost minim - Algoritmul Partition Shortest Path (PSP)

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

Partition Shortest Path (PSP) algoritm

```
us ← 0; before(s) ← 0; S ← ∅; NOW ← {s}; NEXT ← ∅;  
while (NOW ∪ NEXT ≠ ∅) do  
    while (NOW ≠ ∅) do  
        extrge i from NOW; S ← S ∪ {i};  
        L ← NG+(i); //conține adiacențele și costurile arcelor care pleacă din i.  
        for (j ∈ L) do  
            if (j ∉ NOW ∪ NEXT) then  
                uj ← ui + aij; before(j) ← i; insert j into NEXT;  
            else if (uj > ui + aij) then  
                uj ← ui + aij; before(j) ← i;  
        if (NEXT ≠ ∅) then  
            find d = mini ∈ NEXT ui; move to NOW each i ∈ NEXT cu ui = d;
```

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 funcționează). Presupunând că

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

vom rezolva sistemul

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

prin aproximări succesive.

Algoritmul Bellman-Ford-Moore

Definim

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

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

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

atunci u^{n-1} este o soluție a sistemului (B). Deoarece valorile lui u^1 sunt evidente, atunci, dacă vom indica o regulă de a trece de la u^k la u^{k+1} , vom obține următorul algoritm pentru a rezolva (B):
¹⁵⁶

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

Algoritmul Bellman-Ford-Moore

```

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

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

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

Algoritmul Bellman-Ford-Moore

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

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

$$\begin{aligned} A &= \{P : P \in \mathcal{P}_{si}, \text{length}(P) \leq k+1\} \\ B &= \{P : P \in \mathcal{P}_{si}, \text{length}(P) \leq k\} \\ C &= \{P : P \in \mathcal{P}_{si}, \text{length}(P) = k+1\} \end{aligned}$$

Atunci, $A = B \cup C$, și

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

Deoarece u_i^k satisfacă (BM) avem

$$\min \{a(P) : P \in A\} = \min (u_i^k, \min \{a(P) : P \in C\})^{157}$$

Algoritmul Bellman-Ford-Moore

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

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

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

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

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

Drumuri de cost minim - Problema P2 pentru costuri reale

Remarci 4

Algoritmul Bellman-Ford-Moore

- Putem adăuga algoritmului următorul pas:

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

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

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

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

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

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

Drumuri de cost minim - Rezolvarea problemei P3

Iterarea algoritmului lui Dijkstra

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

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

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

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

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

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

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

Iterarea algoritmului lui Dijkstra

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

- 1: determină α și construiește matricea \bar{A} ;
- 2: rezolvă P3 pentru \bar{A} , returnând \bar{U} și $Before$;
- 3: determină U ($u_{ij} = \bar{u}_{ij} - \alpha(i) + \alpha(j)$);

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

Iterarea algoritmului lui Dijkstra

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

Algoritmul Floyd - Warshall

Fie

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

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

Algoritmul Floyd - Warshall

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

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

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

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

Drumuri de cost minim - Rezolvarea problemei P3

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph 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 știm că digraful nu are circuite de cost negative, atunci **dacă elementele diagonale ale lui A sunt inițializate 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 *

Drumuri de cost minim - Rezolvarea problemei P3

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

Înmulțirea (rapidă) a matricilor

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

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

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

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

Algoritmi * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo162ns * C. Croitoru - Graph Algorithms *

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

Înmulțirea (rapidă) a matricilor

```

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

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

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

Înmulțirea (rapidă) a matricilor

Considerăm următorul “produs de matrici”

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

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

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

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

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

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

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

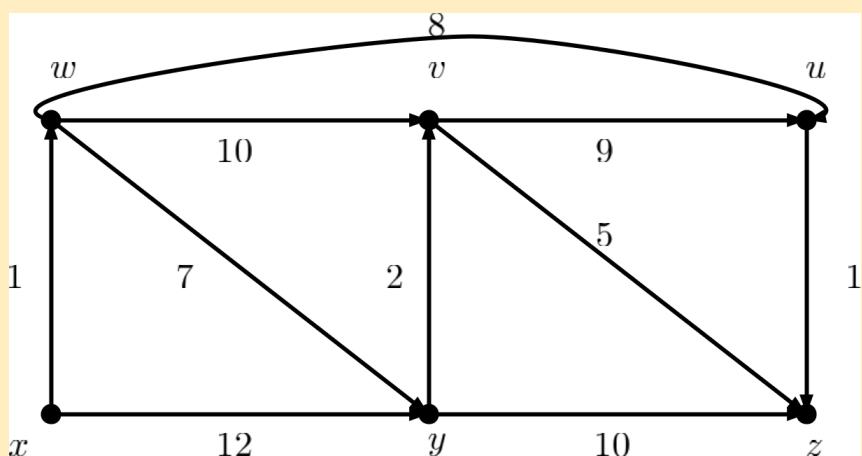
Dacă operația \otimes este implementată cu algoritmi mai rapizi, n^3 de mai sus devine $n^{\log_2 7} = n^{2.81}$ (Strassen 1969) sau $n^{2.3728639}$ (Coopersmith & Winograd 1987, Le Gall 2014).

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

Exerciții pentru seminarul de săptămâna următoare

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

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



C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 164ns * C. Croitoru - Graph Algorithms *

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

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

Exerciții pentru seminarul de săptămâna următare

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

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

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

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

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

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

Arătați că această procedură nu este corectă, oferind un contra-exemplu.

Exerciții pentru seminarul de săptămâna următoare

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

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

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

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

Exercițiul 7. Fie $G = (V, E)$ un digraf, $a : E \rightarrow \mathbb{R}_+$ o funcție de cost pe arcele sale și $x_0 \in V$ un nod din care toate celelalte noduri ale lui G sunt accesibile. Un SP-arbore pentru tripla (G, a, x_0) este un arbore cu rădăcină al lui G , $T = (V, E')$, așa încât costul (cu aceeași funcție a) drumului de la x_0 la u în T este costul minim al unui drum de la x_0 la u în G , pentru orice $u \in V$.

(a) Arătați că un astfel de SP-arbore există întotdeauna.

(b) Descrieți un algoritm care să determine un SP-arbore.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul de săptămâna următoare

[Jump to Table of contents](#)

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

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

Exercițiu 8. Fie G un graf conex. Arătați că

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

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

Exercițiu 9. Fie $G = (V, E)$ un graf conex.

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

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

Exerciții pentru seminarul de săptămâna următoare

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

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

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

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

Exercițiu 10. Arătați că orice arbore, T , are cel puțin $\Delta(T)$ noduri pendante (frunze).

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

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

Exercițiu 11. Fie G un graf cu $n \geq 2$ noduri și m muchii.

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

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

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

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

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

Exercițiu 12. Fie $G = (S, T; E)$ un graf bipartit cu următoarele proprietăți:

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

Arătați că $m \leq n$;

Exercițiu 13. Fie $G = (V, E)$ un digraf; definim o funcție $f_G : \mathcal{P}(V) \rightarrow \mathbb{N}$ prin $f_G(\emptyset) = 0$ și $f_G(S) = |\{v : v \text{ este accesibil din } S\}|$, pentru $\emptyset \neq S \subseteq V$.

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

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

(b) Demonstrați că (a) este echivalentă cu

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

Exerciții rezolvate (partial)

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

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

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

Exercițiu 1. Soluție.

- Evident,

$$\text{Offline}(P) = \begin{cases} \text{length}(P), & \text{if } \text{length}(P) < T, \\ ?, & \text{if } \text{length}(P) \geq T. \end{cases}$$

- Putem defini următoarea strategie: atâtă vreme cât lungimea drumului curent este mai mică decât T plătim 1\$ pe muchie, când drumul ajunge la lungimea T plătim $T$$. Astfel

$$\text{Online}(P) = \begin{cases} ?, & \text{dacă } \text{length}(P) < T, \\ ?, & \text{dacă } \text{length}(P) \geq T. \end{cases}$$

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

Exercițiu 2. Soluție. Algoritmul de mai jos caută în $\mathcal{O}(n^3)$ (folosind algoritmul Bellman-Ford-Moore) un circuit C' de cost negativ relativ la funcția de cost $c(\cdot) = a(\cdot) - \lambda b(\cdot)$, unde $\lambda = \frac{a(C^*)}{b(C^*)}$, pentru un circuit C^* . Dacă se determină un astfel de circuit, atunci C^* este înlocuit cu C' , deoarece $\frac{a(C')}{b(C')} < \frac{a(C^*)}{b(C^*)}$ (de ce?). Altfel $\frac{a(C')}{b(C')} \geq \frac{a(C^*)}{b(C^*)}$, $\forall C'$ circuit în D (de ce?).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

```
fie  $C^*$  un circuit în  $D$ ; // altfel  $D$  este digraf aciclic și problema nu mai are obiect;  
 $\lambda \leftarrow a(C^*)/b(C^*)$ ;  
for ( $e \in E$ )  
     $c(e) \leftarrow a(e) - \lambda b(e)$ ;  
while( $\exists C'$  un circuit în  $D$  cu  $c(C') < 0$ ) {  
     $C^* \leftarrow C'$ ;  $\lambda \leftarrow a(C^*)/b(C^*)$ ;  
    for ( $e \in E$ )  
         $c(e) \leftarrow a(e) - \lambda b(e)$ ;  
}  
return  $C^*$ ;
```

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

Algoritmul poate fi scris și ca o procedură de căutare binară. Cum?

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo169ns * C. Croitoru - Graph Algorithms *

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

Exercițiu 3. Soluție.

- Complexitatea timp poate fi redusă întreținând o familie de liste $(\mathcal{Q}_i)_{0 \leq i \leq n_C}$, unde \mathcal{Q}_h conține toate nodurile v pentru care costul minim (current) al unui sv -drum este h .
 - Orice nod de tip j^* este definitiv șters din listele \mathcal{Q} și toți vecinii lui din $V \setminus S$ sunt luați în considerare. **De ce?**
 - Complexitatea dorită rezultă din această ultimă observație. **Cum?**

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

Exercitii rezolvate (partial)

```

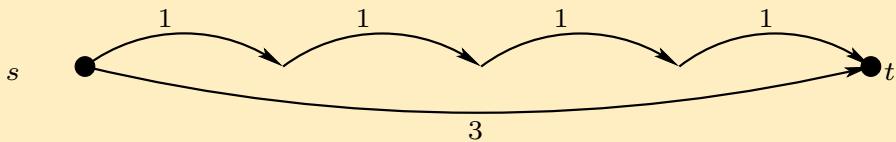
for  $i = 0$  to  $(nC + 1)$  do  $\mathcal{Q}_i \leftarrow \emptyset$ ;
 $S \leftarrow \{s\}$ ;  $u_s \leftarrow 0$ ;  $before[s] \leftarrow 0$ ;  $i \leftarrow 0$ ;
for  $(i \in V \setminus S)$  do  $\{ u_i \leftarrow \infty; \mathcal{Q}_{nC+1} \leftarrow \mathcal{Q}_{nC+1} \cup \{u_i\}; before[i] \leftarrow s; \}$ 

//  $\mathcal{Q}_{nC+1}$  plays the role of  $\mathcal{Q}_\infty$ .
while  $(i < nC)$  {
    let  $i$  minimum such that  $\mathcal{Q}_i \neq \emptyset$ ; // we must keep a pointer to this list;
    let  $j^* \in \mathcal{Q}_i$ ;  $S \leftarrow S \cup \{j^*\}$ ;  $\mathcal{Q}_i \leftarrow \mathcal{Q}_i \setminus \{j^*\}$ ;
    for( $j \in N_G(j^*) \cap (V \setminus S)$ ) do
        if  $(u_j > u_{j^*} + c(jj^*))$  {
             $\mathcal{Q}_{u_j} \leftarrow \mathcal{Q}_{u_j} \setminus \{j\}$ ;  $u_j \leftarrow u_{j^*} + c(jj^*)$ ;  $before[j] \leftarrow j^*$ ;
             $\mathcal{Q}_{u_j} \leftarrow \mathcal{Q}_{u_j} \cup \{j\}$ ;
        }
         $i++$ ;
}

```

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

Exercițiu 5. Soluție. Pentru următorul digraf procedura propusă nu construiește un drum de la s la t de cost minim (care este 3).



Figure

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

Exercițiu 7. Soluție.

- Aplicăm algoritmul lui Dijkstra plecând din x_0 . *before* este un tablou de părinți într-un arbore care are proprietatea cerută. **De ce?**
- Algoritmul lui Dijkstra cu nodul de start x_0 .

Exercițiu 8. Soluție.

- Fie D_1 și D_2 două drumuri de lungime maximă din $G = (V, E)$. Să presupunem prin reducere la absurd că $V(D_1) \cap V(D_2) = \emptyset$. Trebuie să existe un x_1x_2 -drum, D , care leagă D_1 și D_2 ($x_i \in V(D_i)$) (**de ce?**); putem presupune că $V(D) \cap V(D_i) = \{x_i\}$ (**de ce?**). x_i împarte D_i în D'_i și D''_i ; fără a restrângе generalitatea putem presupune că $\text{length}(D'_i) \geq \text{length}(D''_i)$. Drumul $D'_1 \cup D \cup D'_2$ este strict mai lung decât D_1 (**de ce?**) - contradicție.

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

- (b) Fie D_1 și D_2 două drumuri de lungime maximă, L , în arborele $T = (V, E)$. Observăm că $D_1 \cap D_2$ este un drum, D , (de ce?); fie D'_i și D''_i subdrumurile lui D_i obținute prin stergerea muchiilor lui D .

Se poate arăta că D_1 și D_2 au în comun nodul (sau nodurile) din mijloc. Cum?

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiu 9. Soluție.

- (a) Prin inducție după $|G|$.
- (b) Fie $G = (V, E)$ un graf conex și S o mulțime stabilă astfel ca graful bipartit $H = (S, T; E')$ este și el conex (folosind (a)).
- Avem $|H| - 1 \leq |E'|$ (de ce?); pe de altă parte

$$|E'| \stackrel{\text{de ce?}}{=} \sum_{x \in S} d_H(x) \leq \sum_{x \in S} d_G(x) \leq \sum_{x \in S} \Delta(G) = |S| \cdot \Delta(G)$$

- Combinând aceste două inegalități obținem $\alpha(G) \geq |S| \geq \frac{|G| - 1}{\Delta(G)}$.

Algoritmics * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo172ns * C. Croitoru - Graph Algorithms *

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

Exercițiu 11. Soluție.

(a) Să presupunem prin reducere la absurd că în G nu există două noduri de același grad. Dacă $V(G) = \{v_1, v_2, \dots, v_n\}$ astfel încât $d_G(v_1) \leq d_G(v_2) \leq \dots \leq d_G(v_n)$, atunci

$d_G(v_1) = 0, d_G(v_2) = 1, \dots, d_G(v_n) = n - 1$. De ce?

Exercitii rezolvate (partial)

Exercțiul 12. Soluție.

(b) " \Rightarrow " Luând $S = X \cup \{v\}$ și $T = Y$, obținem din (a)

$$f_G(X \cup \{v\}) + f_G(Y) \geq f_G(Y \cup \{v\}) + f_G(X) \iff$$

$$f_G(X \cup \{v\}) - f_G(X) \geq f_G(Y \cup \{v\}) - f_G(Y),$$

$$\forall X \subseteq Y \subseteq V, \forall v \in V \setminus Y.$$

II → III

$$f_G(Y) - f_G(X) \geq f_G(Y \cup \{v\}) - f_G(X \cup \{v\}),$$

$$\forall X \subseteq Y \subseteq V, \forall v \in V \setminus Y,$$

astfel, aplicând în mod repetat această inegalitate obținem (de ce?)

$$f_G(Y) - f_G(X) \geq f_G(Y \cup A) - f_G^{173}(X \cup A), \forall X \subseteq Y \subseteq V, \forall A \subseteq V \setminus Y.$$

Un **graf ponderat** este un graf $G = (V, E)$ cu o funcție $w : E \rightarrow \mathbb{R}$ care atribuie o **pondere** $w(e)$ fiecărei muchii $e \in E$.

- Ponderile pot reprezenta distanțe dintre noduri dar și alte metriki precum timpi, costuri, penalizări, pierderi sau alte cantități care se acumulează liniar de-a lungul unui drum și pe care vrem să le minimizăm.
- Vom studia doar **grafuri ponderate simple**, adică
 - ▶ fără bucle
 - ▶ cu cel mult o muchie de la un nod la alt nod
- Vom scrie $w(x, y)$ în loc de $w(e)$ atunci când e este muchia $x-y$ sau arcul $x \rightarrow y$.
- Deasemenea, vom presupune că $w(x, x) = 0$ și $w(x, y) = +\infty$ dacă nu există muchie de la x la y .

Grafuri ponderate

Noțiuni fundamentale

Scriem $x \xrightarrow{\pi} y$ pentru a indica faptul că π este o cale de la x la y .

Lungimea ponderată a unei liste de noduri $\pi = [x_1, x_2, \dots, x_n]$ este

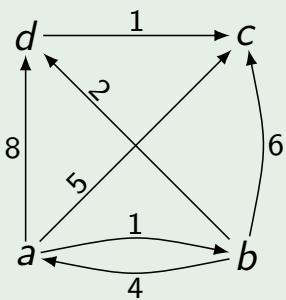
$$\text{length}_w(\pi) = \sum_{i=1}^{n-1} w(x_i, x_{i+1}).$$

Dacă $n = 1$ atunci $\pi = [x_1]$ și $\text{length}_w(\pi) = 0$.

Distanța ponderată de la x la y în G este

$$\delta_w(x, y) = \begin{cases} +\infty & \text{dacă } x \not\xrightarrow{\pi} y, \\ \min\{\text{length}_w(\pi) \mid x \xrightarrow{\pi} y\} & \text{în caz contrar.} \end{cases}$$

Exemplu



$\delta_w(x, y)$	$y = a$	$y = b$	$y = c$	$y = d$
$x = a$	0	1	4	3
$x = b$	4	0	3	2
$x = c$	$+\infty$	$+\infty$	$+\infty$	$+\infty$
$x = d$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

$$\text{length}_w([a, b, c]) = 7,$$

$$\text{length}_w([a, d, c]) = 9,$$

$$\text{length}_w([a, b, d, c]) = 4.$$

Grafuri ponderate

Probleme fundamentale

Vom prezenta soluții algoritmice la problemele următoare:

- ① Să se determine căi cu lungime ponderată minimă de la un nod sursă s la toate nodurile la care se poate ajunge din s .
- ② Să se determine căi cu lungime ponderată minimă de la x la y pentru toate perechile de noduri conectate $x \rightsquigarrow y$.

Remarcă

Dacă $\pi = [x_1, x_2, \dots, x_k]$ este o cale de la x_1 la x_k cu $\text{length}_w(\pi) = \delta_w(x_1, x_k)$, atunci pentru toți $1 \leq i \leq j \leq n$:

- Dacă $\pi_{i,j} = [x_i, x_{i+1}, \dots, x_j]$ atunci $\text{length}_w(\pi_{i,j}) = \delta(x_i, x_j)$.

Altfel spus, *toate subcările unei căi cu lungime ponderată minimă au lungime ponderată minimă*.

Muchiile e cu $w(e) < 0$ pot forma cicluri cu lungimi ponderate negative \Rightarrow pentru orice noduri x, y :

- Dacă există un nod z într-un ciclu c cu lungime ponderată negativă și $x \rightsquigarrow z \rightsquigarrow y$ atunci nu există $x \rightsquigarrow y$ cu lungime ponderată minimă fiindcă traversarea repetată a lui c produce căi cu lungime ponderată ce tind la $-\infty$. În acest caz definim $\delta_w(x, y) = -\infty$.
- În caz contrar, $\delta_w(x, y) \in \mathbb{R}$ și există $x \rightsquigarrow y$ cu $\text{length}_w(\pi) = \delta_w(x, y)$.

Cicluri cu lungimi ponderate negative

Exemplu

Digaful de mai jos are cicluri cu lungime ponderată negativă:

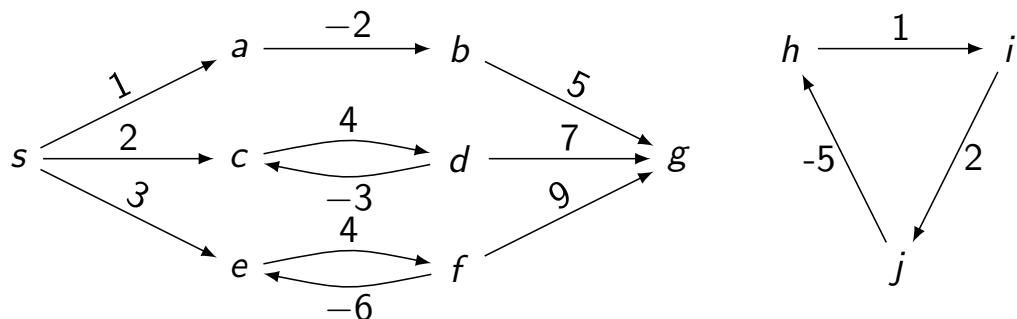
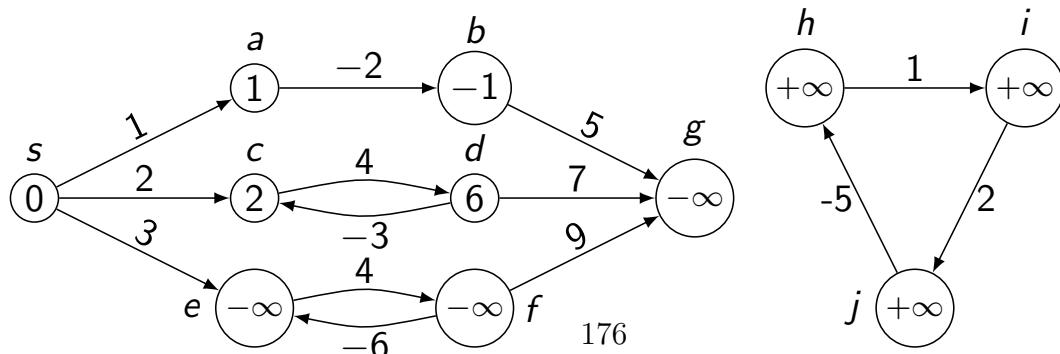


Figura următoare indică valorile lui $\delta_w(s, x)$ pentru toate nodurile x :



Fie $x \xrightarrow{\pi} y$ o cale cu lungime ponderată minimă. Observăm că

- ① π nu poate conține un ciclu cu lungime ponderată strict negativă pentru că ar rezulta că $\delta_w(x, y) = -\infty$.
- ② π nu poate conține un ciclu cu lungime ponderată strict pozitivă pentru că dacă-l eliminăm din π obținem $x \xrightarrow{\pi'} y$ cu $\text{length}_w(\pi') < \text{length}_w(\pi) = \delta_w(x, y)$, contradicție.
- ③ Putem presupune că π nu conține cicluri cu lungime ponderată 0 fiindcă ele se pot elimina din π fără să-i afecteze lungimea ponderată.

Deci ne putem limita să căutăm doar căi aciclice $i \xrightarrow{\pi} j$ cu lungime ponderată minimă. Căile aciclice conțin cel mult $|V| = n$ noduri distincte, deci cel mult $n - 1$ muchii.



Căi cu lungime ponderată minimă de la un nod sursă s

Algoritmul lui Bellman-Ford și Algoritmul lui Dijkstra

Ambii algoritmi calculează reprezentarea cu predecesori a unui arbore T_s cu rădăcina s astfel încât

- ① mulțimea de noduri a lui T_s este $S_s = \{x \in V \mid s \rightsquigarrow x\}$
- ② pentru fiecare $s \in S_s$, lista de noduri pe ramura de la s la x în T_s este o cale cu lungime ponderată minimă de la s la x în G .

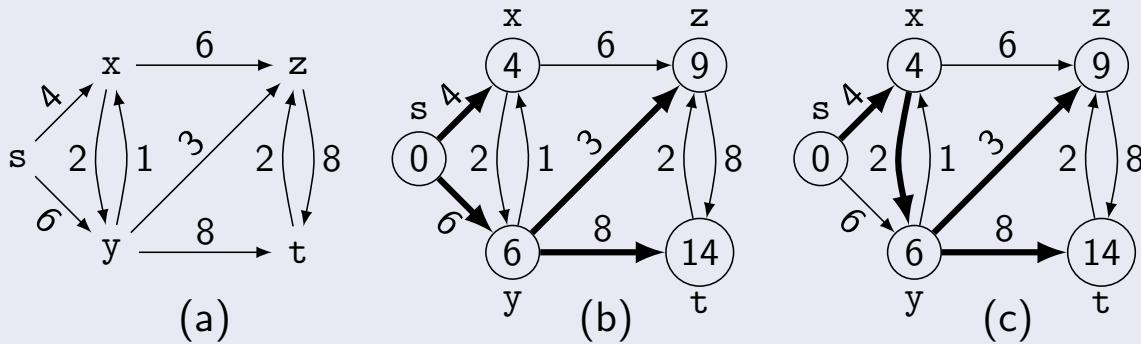
Un astfel de arbore se numește **arbore de căi cu lungimi ponderate minime de la s în G** .

- **Algoritmul lui Dijkstra** este definit pentru grafuri ponderate cu $w(e) > 0$ pentru toate muchiile e .
- **Algoritmul lui Bellman-Ford** este definit pentru cazul general, când putem avea muchii e cu $w(e) < 0$.
 - Detectează eventualele cicluri cu lungime ponderată negativă la care se poate ajunge din s . În acest caz, returnează false pentru a semnala existența unui astfel de ciclu și abandonează calculul arborelui T_s .



Exemplu ilustrat

Digraful ponderat din figura (a) are 2 arbori de căi cu lungimi ponderate minime cu rădăcina s. Figurile (b) și (c) ilustrează muchiile celor doi arbori cu linii îngroșate, iar valoarea lui $\delta_w(s, x)$ este indicată în interiorul fiecărui nod x.



Algoritmul lui Bellman-Ford și Algoritmul lui Dijkstra

Caracteristici comune (1)

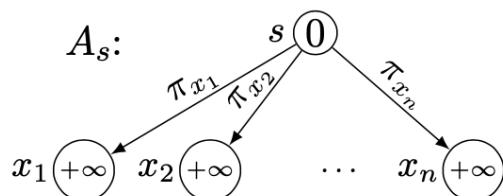
Algoritmii operează cu:

- ① reprezentarea cu precedenți a unui arbore A_s cu rădăcina s și mulțimea de noduri V . Vom presupune că, pentru fiecare $x \in V$, π_x este lista de noduri pe ramura de la s la x în A_s .
- ② $d[x]$: o margine superioară a lui $\text{length}_w(\pi_x)$:

$$\forall x \in V. \delta_w(s, x) \leq \text{length}_w(s, x) \leq d[x].$$

Valorile inițiale sunt

- $p[s] = \text{null}$ și $p[x] = s$ pentru toți $x \in V - \{s\}$, și
- $d[s] = 0$ și $d[x] = +\infty$ pentru toți $x \in V - \{s\}$.



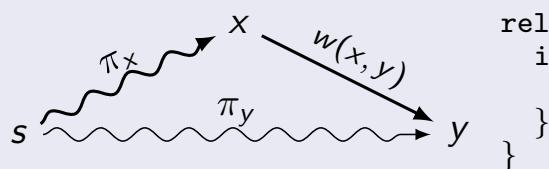
unde $V = \{s, x_1, x_2, \dots, x_n\}$.
 Valorile lui $d[x]$ sunt indicate
¹⁷⁸în interiorul nodurilor respective.

Valorile lui $d[]$ și $p[]$ se modifică efectuând un număr finit de **relaxări de muchii** și garantează că, atunci când se termină:

- ▶ A_s este arbore de căi cu lungimi ponderate minime de la s în G .
- ▶ $d[x] = \delta_w(s, x)$ pentru toți $x \in V$.

Relaxarea unei muchii de la x la y

Dacă $d[x] + w(x, y) < d[y]$ și considerăm calea $\pi'_y = s \xrightarrow{\pi_x} x \rightarrow y$ atunci
 $\delta_w(x, y) \leq \text{length}_w(\pi'_y) = \text{length}_w(\pi_x) + w(x, y) \leq d[x] + w(x, y) < d[y]$
 \Rightarrow putem înlocui $p[y]$ cu $p[x]$ și $d[y]$ cu $d[x] + w(x, y)$.



```
relax(x,y) {
    if (d[x] + w(x,y) < d[y]) {
        p[y] = x; d[y] = d[x] + w(x,y);
    }
}
```

Algoritmul lui Bellman-Ford

Pseudocod

```
boolean BellmanFord(G,s) {
    initializeaza(G,s);
    for i=1 to G.V()-1
        foreach x ∈ V(G)
            for (y:adj[x])
                relax(x,y);
    foreach x ∈ V(G)
        for (y:adj[x])
            if (d[x] > d[y]+w(x,y))
                return false;
    return true;
}
```

```

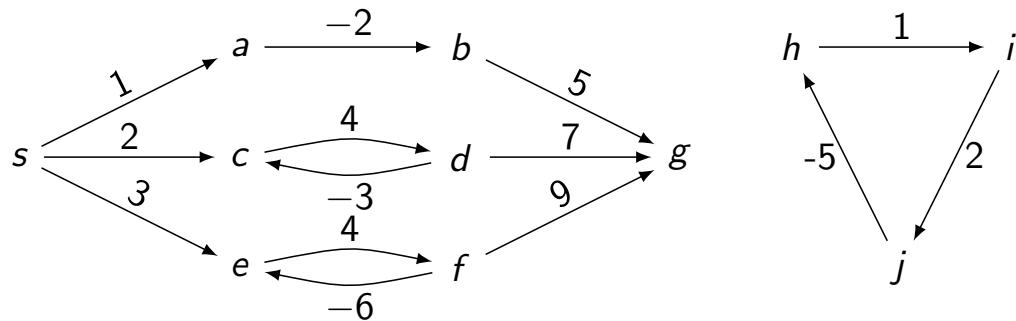
boolean BellmanFord(G,s) {
    initializeaza(G,s);
    for i=1 to G.V()-1
        foreach x ∈ V(G)
            for (y:adj[x])
                relax(x,y);
    foreach x ∈ V(G)
        for (y:adj[x])
            if (d[x] > d[y]+w(x,y))
                return false;
    return true;
}

```

Complexitate (temp de execuție): $O(n \cdot m^2)$

Algoritmul lui Bellman-Ford

Exemplu ilustrat

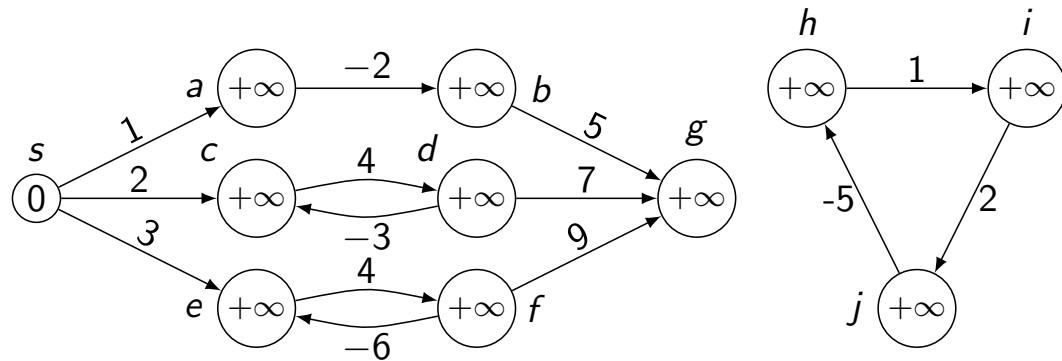


Algoritmul lui Bellman-Ford

Exemplu ilustrat

[Jump to Table of contents](#)

După pasul de inițializare:

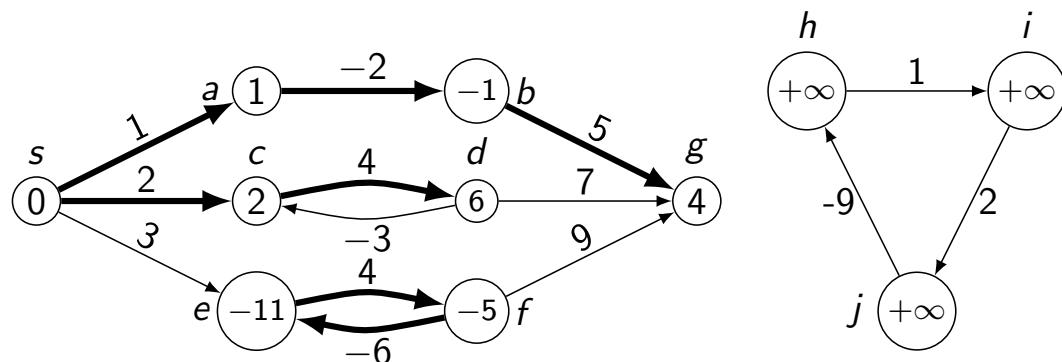


Cursul 10/11: Grafuli ponderate

Algoritmul lui Bellman-Ford

Exemplu ilustrat

După a 6-a buclă for:



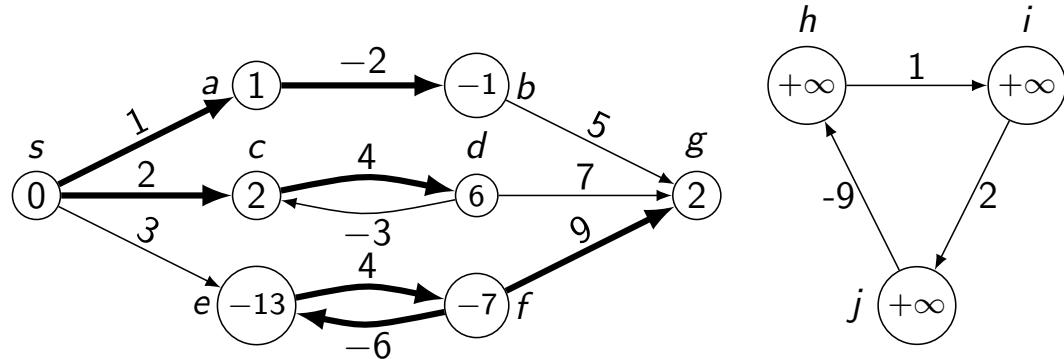
Cursul 10/11: Grafuli ponderate

Algoritmul lui Bellman-Ford

Exemplu ilustrat

[Jump to Table of contents](#)

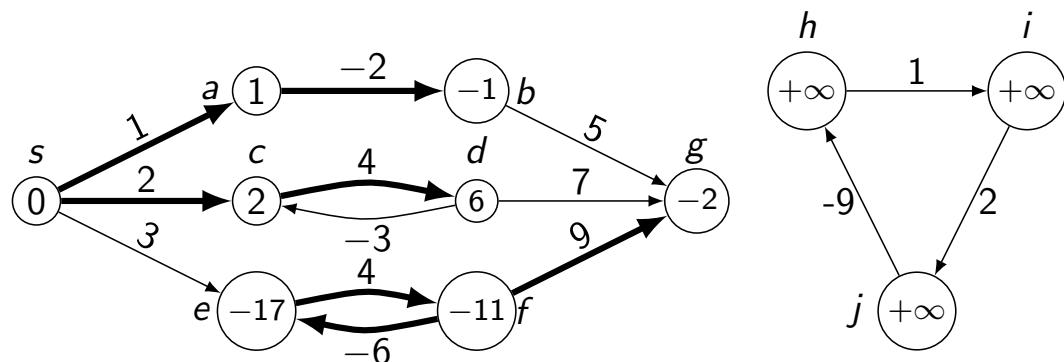
După a 8-a buclă for:



Algoritmul lui Bellman-Ford

Exemplu ilustrat

După a 10-a buclă for:



Algoritmul returnează false pentru că detectează

$$d[f] = -11 > d[e] + w(e, f).$$

```

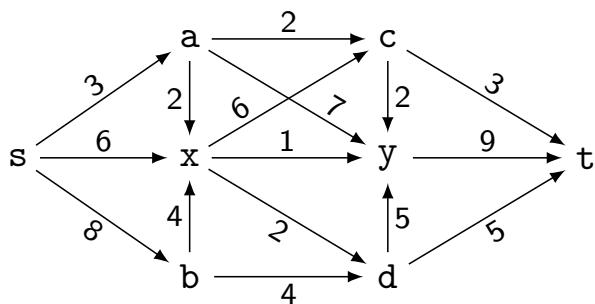
void Dijkstra(G,s) {
    initializeaza(G,s);
    Q=multimea de noduri a lui G;
    while (!Q.isEmpty()) {
        extrage u cu d[u] = min{d[x] | x ∈ Q} din Q;
        for (v:G.adj(u))
            if (Q.contains(v))
                relax(u,v);
    }
}

```

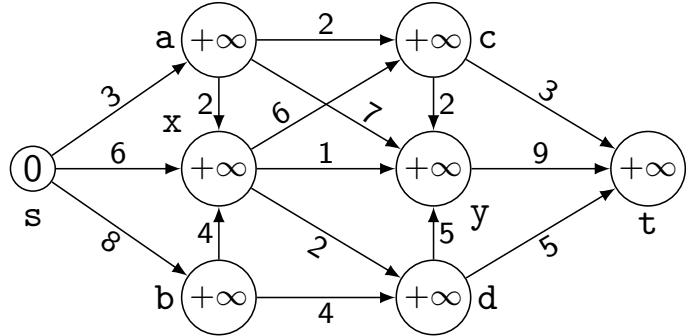
Complexitate (temp de execuție): $O(n^2)$

Algoritmul lui Dijkstra

Exemplu ilustrat



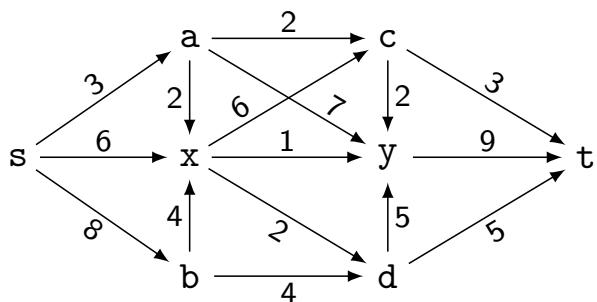
După inițializare avem



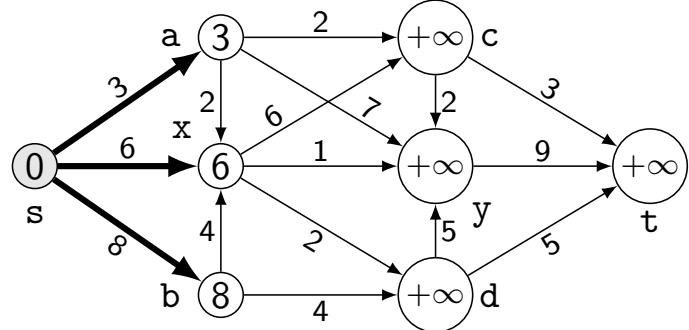
Algoritmul lui Dijkstra

Exemplu ilustrat

[Jump to Table of contents](#)



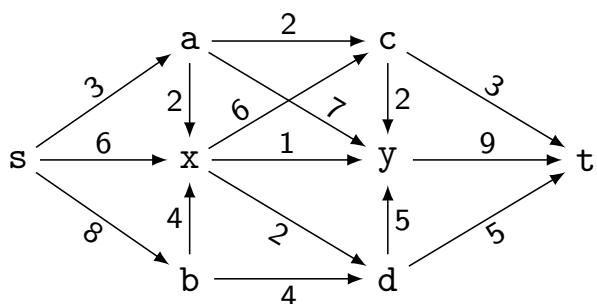
După relaxarea muchiilor din s avem



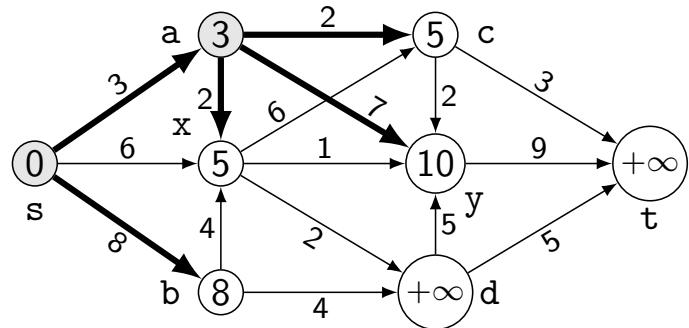
Cursul 10/11: Grafuri ponderate

Algoritmul lui Dijkstra

Exemplu ilustrat



După relaxarea muchiilor din a avem

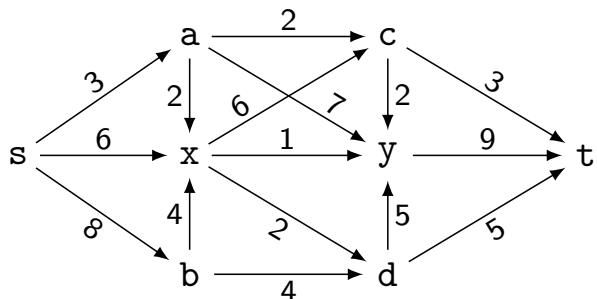


Cursul 10/11: Grafuri ponderate

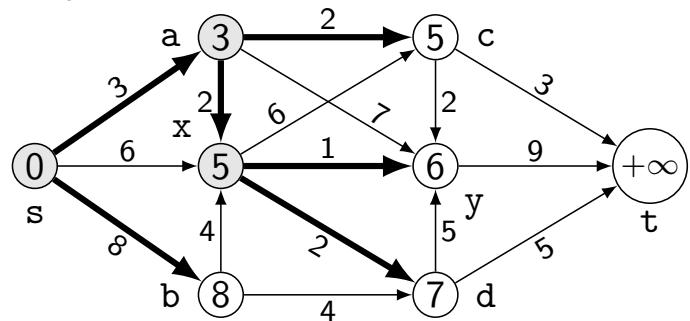
Algoritmul lui Dijkstra

Exemplu ilustrat

[Jump to Table of contents](#)

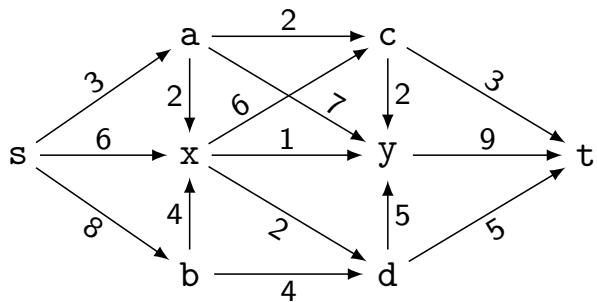


După relaxarea muchiilor din x avem

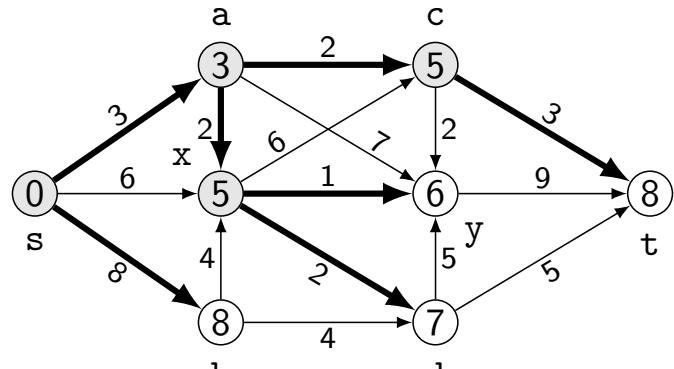


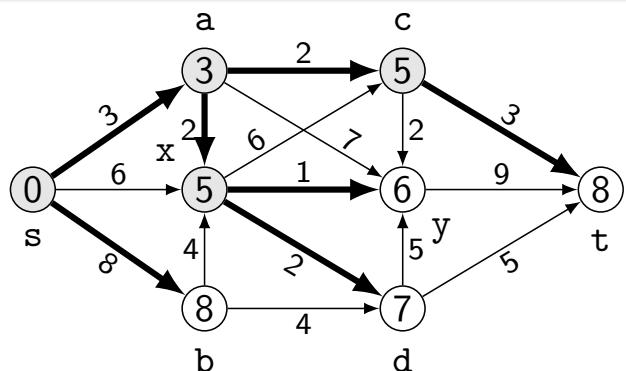
Algoritmul lui Dijkstra

Exemplu ilustrat



După relaxarea muchiilor din c avem

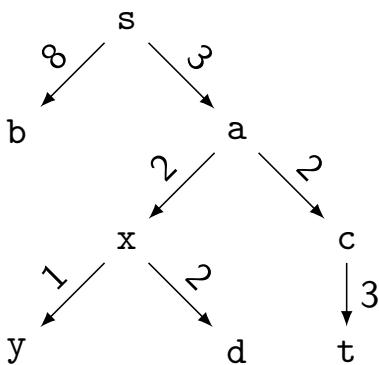




Relaxările ulterioare nu mai modifică valorile lui $p[]$ și $d[]$:

x	s	a	x	b	c	y	d	t
p[x]	null	s	a	s	a	x	x	c
d[x]	0	3	5	8	5	6	7	8

⇒ arborele de căi cu lungimi ponderate minime calculat de algoritm este



Căi cu lungime ponderată minimă între toate nodurile

Se dă un graf ponderat G cu n noduri și m muchii

Se cere: pentru toți $x, y \in V$ să se găsească $x \xrightarrow{\pi_{x,y}} y$ cu $\text{length}_w(\pi_{x,y}) = \delta_w(x, y)$.

OBSERVAȚII:

- ① Această problemă se poate rezolva rulând de n ori unul din algoritmii deja prezențați, câte o dată pentru fiecare nod $x \in V(G)$ ca sursă.
- ② Complexitate temporală:
 - $O(n^4)$ dacă se folosește alg. lui Bellman-Ford pentru cazul general, când putem avea muchii cu ponderi negative.
 - $O(n^3)$ dacă se folosește alg. lui Dijkstra pentru cazul general particular, când $w(e) > 0$ pentru toate muchiile $e \in E$.
- ③ Vom prezenta o metodă nouă: Algoritmul lui Floyd-Warshall:
 - Complexitate temporală: $O(n^3)$ pentru cazul când putem avea muchii cu ponderi negativi, dar fără cicluri cu lungime ponderată negativă.

Două tablouri $n \times n$, astfel încât, pentru orice $x, y \in V$:

- ① $d[x][y]$: o margine superioară a lui $\delta_w(x, y)$.
- ② $P[x][y] \in \{\text{null}\} \cup V$.

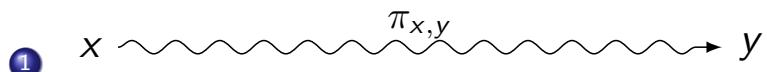
La terminarea algoritmului, valorile lui $P[][]$ și $d[][]$ au proprietățile următoare:

- $d[x][y] = \delta_w(x, y)$.
- Dacă $x \neq y$ și există un drum cu lungime ponderată minimă de la x la y atunci $P[x][y]$ este predecesorul nodului x pe o cale $x \xrightarrow{\pi} y$ cu lungime ponderată minimă.

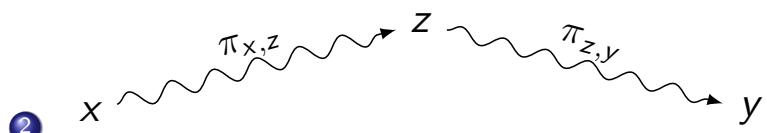
Algoritmul lui Floyd-Warshall

Idee de bază

Dacă $x, y, z \in V$ atunci orice drum $\pi_{x,y}$ cu lungime ponderată minimă de la x la y are una din următoarele 2 forme:



unde z nu este nod intermediar al drumului $\pi_{x,y}$, sau



unde z nu este nod intermediar al drumurilor $\pi_{x,z}$ și $\pi_{z,y}$.

⇒ putem defini o metodă recursivă de calcul al elementelor tablourilor $P[][]$ și $d[][]$.

Algoritmul lui Floyd-Warshall

Calculul recursiv al elementelor tablourilor $d[][]$ și $P[][]$

[Jump to Table of contents](#)

Fie $[x_1, x_2, \dots, x_n]$ o enumerare fixată a nodurilor din $V(G)$.

Definim pentru $0 \leq k \leq n$ definim

- ▶ $d[k][i][j]$ este cea mai mică lungime ponderată a unei căi de la x_i la x_j care trece doar prin noduri intermediare din mulțimea $\{x_1, \dots, x_k\}$. Dacă o astfel de cale nu există, atunci $d[k][i][j] = +\infty$.
- ▶ $P[k][i][j]$ este null dacă $i = j$ sau $d[k][i][j] = +\infty$. În caz contrar, $P[k][i][j]$ este predecesorul nodului x_j pe un drum cu lungime ponderată minimă de la x_i la x_j care trece doar prin noduri intermediare din mulțimea $\{x_1, \dots, x_k\}$.



Cursul 10/11: Grafuli ponderate

Algoritmul lui Floyd-Warshall

Calculul recursiv al elementelor tablourilor $d[][]$ și $P[][]$ (continuare)

Rezultă că, pentru toți $i, j \in \{1, 2, \dots, n\}$ avem

$$\begin{aligned} d[0][i][j] &= w(x_i, x_j), \\ P[0][i][j] &= \begin{cases} \text{null} & \text{dacă } i = j \text{ sau } w(x_i, x_j) = +\infty, \\ x_i & \text{în caz contrar} \end{cases} \end{aligned}$$

iar dacă $1 \leq k \leq n$ atunci

$$\begin{aligned} d[0][i][j] &= \min(d[k-1][i][j], d[k-1][i][k] + d[k-1][k][j]), \\ P[k][i][j] &= \begin{cases} P[k-1][i][j] & \text{dacă } d[k-1][i][j] = d[k][i][j], \\ P[k-1][k][j] & \text{în caz contrar.} \end{cases} \end{aligned}$$

REMARCA FINALĂ: Deoarece nodurile intermediare ale oricărei căi sunt în mulțimea $\{x_1, x_2, \dots, x_n\}$, putem defini

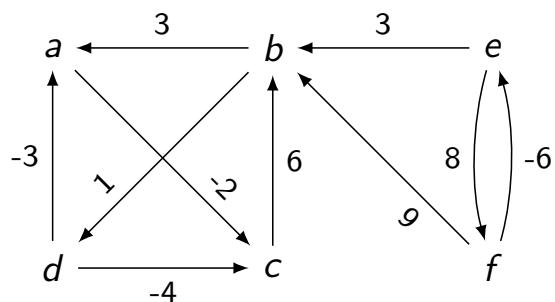
$$d[x_i][x_j] = d[n][i][j] \text{ și } P[x_i][x_j] = P[n][i][j].$$

Cursul 10/11: Grafuli ponderate

- ① Inițializarea valorilor lui $d[0][i][j]$ și $P[0][i][j]$ pentru $1 \leq i, j \leq n$ durează $O(n^2)$.
- ② Calculul valorilor lui $d[k][i][j]$ și $P[k][i][j]$ din valorile pentru $k - 1$ durează $O(n^2)$.
- ③ Acest calcul trebuie repetat pentru k de la 1 la $n \Rightarrow$ complexitatea temporală $n \cdot O(n^2) = O(n^3)$.

Algoritmul lui Floyd-Warshall

Exemplu ilustrat

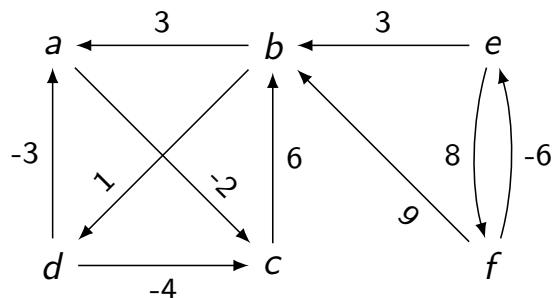


Nodurile sunt enumerate în ordinea $[a, b, c, d, e, f]$

k	$d[k]$	$P[k]$
0	$\begin{pmatrix} 0 & +\infty & -2 & +\infty & +\infty & +\infty \\ 3 & 0 & +\infty & 1 & +\infty & +\infty \\ +\infty & 6 & 0 & +\infty & +\infty & +\infty \\ -3 & +\infty & -4 & 0 & +\infty & +\infty \\ +\infty & 3 & +\infty & +\infty & 0 & 8 \\ +\infty & 9 & +\infty & +\infty & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & \bullet & a & \bullet & \bullet & \bullet \\ b & \bullet & \bullet & b & \bullet & \bullet \\ \bullet & c & \bullet & \bullet & \bullet & \bullet \\ d & \bullet & d & \bullet & \bullet & \bullet \\ \bullet & e & \bullet & \bullet & \bullet & e \\ \bullet & f & \bullet & \bullet & f & \bullet \end{pmatrix}$

Algoritmul lui Floyd-Warshall

Exemplu ilustrat



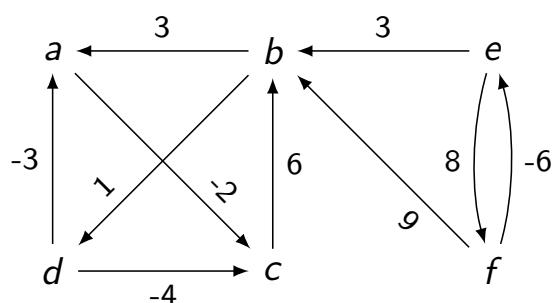
Nodurile sunt enumerate în ordinea
[a, b, c, d, e, f]



Cursul 10/11: Grafuri ponderate

Algoritmul lui Floyd-Warshall

Exemplu ilustrat



Nodurile sunt enumerate în ordinea
[a, b, c, d, e, f]

k	$d[k]$	$P[k]$
2	$\begin{pmatrix} 0 & +\infty & -2 & +\infty & +\infty & +\infty \\ 3 & 0 & 1 & 1 & +\infty & +\infty \\ 9 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & +\infty & -5 & 0 & +\infty & +\infty \\ 6 & 3 & 4 & 4 & 0 & 8 \\ 12 & 9 & 10 & 10 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & \bullet & a & \bullet & \bullet & \bullet \\ b & \bullet & a & b & \bullet & \bullet \\ b & c & \bullet & b & \bullet & \bullet \\ d & \bullet & a & \bullet & \bullet & \bullet \\ b & e & a & b & \bullet & e \\ b & f & a & b & f & \bullet \end{pmatrix}$

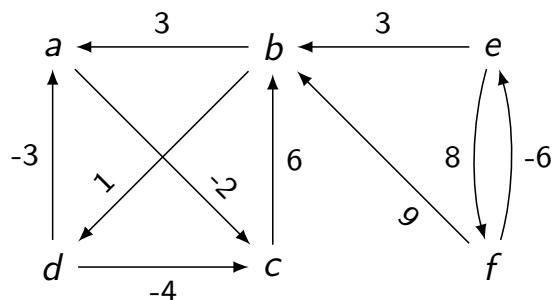


Cursul 10/11: Grafuri ponderate

Algoritmul lui Floyd-Warshall

Exemplu ilustrat

[Jump to Table of contents](#)



Nodurile sunt enumerate în ordinea
[a, b, c, d, e, f]

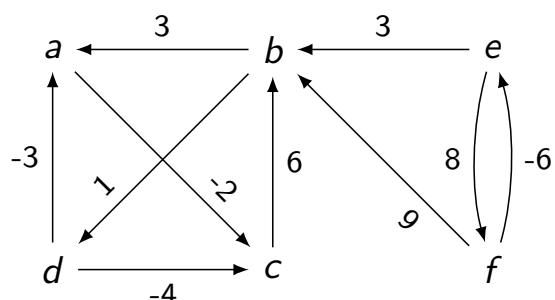
k	d[k]	P[k]
3	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ 3 & 0 & 1 & 1 & +\infty & +\infty \\ 9 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 6 & 3 & 4 & 4 & 0 & 8 \\ 12 & 9 & 10 & 10 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ b & \bullet & a & b & \bullet & \bullet \\ b & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ b & e & a & b & \bullet & e \\ b & f & a & b & f & \bullet \end{pmatrix}$



Cursul 10/11: Grafuri ponderate

Algoritmul lui Floyd-Warshall

Exemplu ilustrat



Nodurile sunt enumerate în ordinea
[a, b, c, d, e, f]

k	d[k]	P[k]
4	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ -2 & 0 & -4 & 1 & +\infty & +\infty \\ 4 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 1 & 3 & -1 & 4 & 0 & 8 \\ 7 & 9 & 5 & 10 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & f & a & b & f & \bullet \end{pmatrix}$

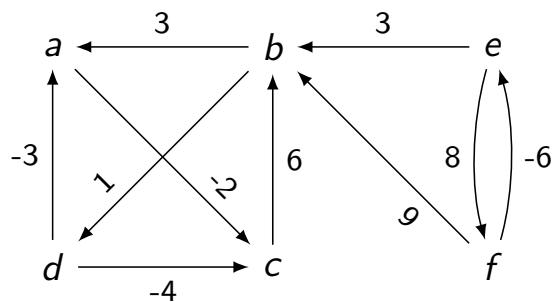


Cursul 10/11: Grafuri ponderate

Algoritmul lui Floyd-Warshall

Exemplu ilustrat

Jump to Table of contents



Nodurile sunt enumerate în ordinea
[a, b, c, d, e, f]

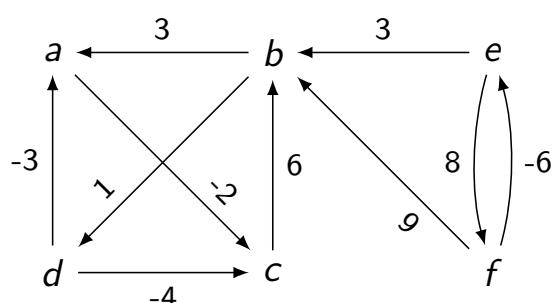
k	d[k]	P[k]
5	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ -2 & 0 & -4 & 1 & +\infty & +\infty \\ 4 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 1 & 3 & -1 & 4 & 0 & 8 \\ -5 & -3 & -7 & -2 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & e & a & b & f & \bullet \end{pmatrix}$



Cursul 10/11: Grafuri ponderate

Algoritmul lui Floyd-Warshall

Exemplu ilustrat



Nodurile sunt enumerate în ordinea
[a, b, c, d, e, f]

k	d[k]	P[k]
6	$\begin{pmatrix} 0 & 4 & -2 & 5 & +\infty & +\infty \\ -2 & 0 & -4 & 1 & +\infty & +\infty \\ 4 & 6 & 0 & 7 & +\infty & +\infty \\ -3 & 1 & -5 & 0 & +\infty & +\infty \\ 1 & 3 & -1 & 4 & 0 & 8 \\ -5 & -3 & -7 & -2 & -6 & 0 \end{pmatrix}$	$\begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ d & e & a & b & f & \bullet \end{pmatrix}$

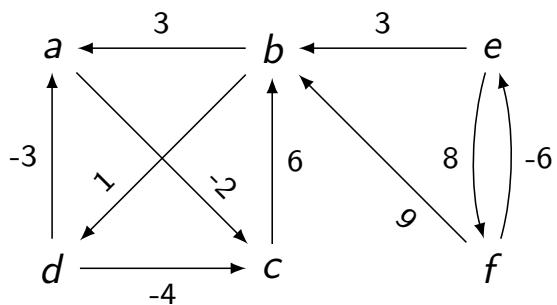


Cursul 10/11: Grafuri ponderate

Algoritmul lui Floyd-Warshall

Exemplu ilustrat (continuare)

[Jump to Table of contents](#)



Nodurile sunt enumerate în ordinea [a, b, c, d, e, f]

În final, elementele tablourilor $P[] []$ și $d[] []$ sunt

	a	b	c	d	e	f		a	b	c	d	e	f
a	•	c	a	b	•	•		0	4	-2	5	$+\infty$	$+\infty$
b	d	•	a	b	•	•		-2	0	-4	1	$+\infty$	$+\infty$
c	d	c	•	b	•	•		4	6	0	7	$+\infty$	$+\infty$
d	d	c	a	•	•	•		-3	1	-5	0	$+\infty$	$+\infty$
e	d	e	a	b	•	e		1	3	-1	4	0	8
f	d	e	a	b	f	•		-5	-3	-7	-2	-6	0



Cursul 10/11: Grafuri ponderate

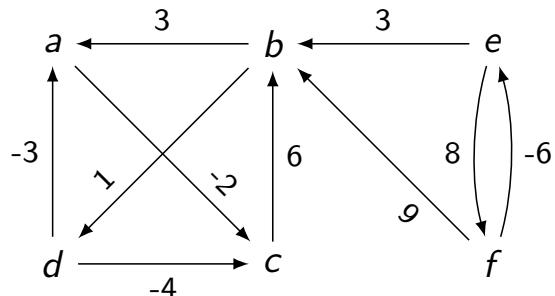
Algoritmul lui Floyd-Warshall

Proprietăți ale tabloului P

- Tabloul P se numește **matrice predecesor**.
- Pentru orice nod $x \in G$ putem defini arborele T_x cu rădăcina x și
 - mulțimea de noduri $\{x\} \cup \{y \in V \mid P[x][y] \neq \text{null}\}$
 - mulțimea de muchii $\{P[x][y] \rightarrow y \mid y \in V - \{x\}\}$.
- T_x este un arbore de căi cu lungimi ponderate minime de la x în G , și poate fi extras din linia corespunzătoare nodului x în tabloul $P[][]$.

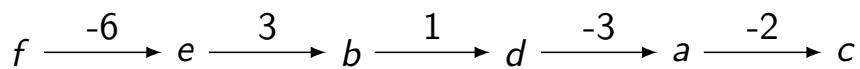
Algoritmul lui Floyd-Warshall

Să se determine un arbore de căi cu lungimi ponderate minime de la f în



$$P[n] = \begin{pmatrix} \bullet & c & a & b & \bullet & \bullet \\ d & \bullet & a & b & \bullet & \bullet \\ d & c & \bullet & b & \bullet & \bullet \\ d & c & a & \bullet & \bullet & \bullet \\ d & e & a & b & \bullet & e \\ \textcolor{red}{d} & \textcolor{red}{e} & \textcolor{red}{a} & \textcolor{red}{b} & \textcolor{red}{f} & \bullet \end{pmatrix}$$

f este al 6-lea element în enumerarea de noduri $[a, b, c, d, e, f]$, deci T_f se extrage din linia 6 a matricii P[6]:



5 CURS 5: Probleme de conexiune în (di)grafuri (teoremele lui Menger, Konig și Hall). Arbori partiali.

acoperire cu noduri - ștergi noduri astfel încât să nu mai rămână în graf nicio muchie

Algoritmica grafurilor - Cursul 5

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

30 octombrie 2020

Cuprins

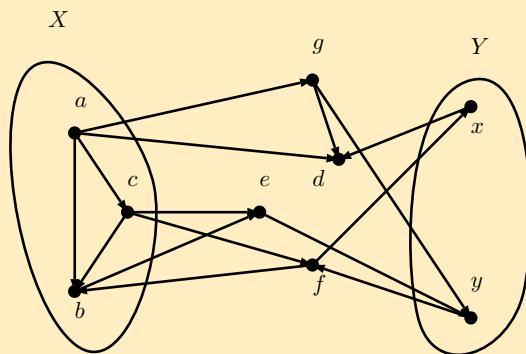
- 1 **Conexiune**
 - **Teorema lui Menger**
 - **p-conexiune**
 - **Teorema lui König**
 - **Teorema lui Hall**
 - **Teorema lui Dirac**
- 2 **Arbore**
 - Elemente de bază
 - Generarea tuturor arborilor parțiali
 - Numărarea arborilor parțiali: **Teorema lui Kirchhoff**
- 3 **Exerciții pentru seminarul de săptămâna viitoare**
- 4 **Exerciții rezolvate (parțial)**

Definiția 1

Fie $G = (V, E)$ un (di)graf și $X, Y \subseteq V$. Un **XY -drum** este orice drum P din G de la un nod $x \in X$ la un nod $y \in Y$ astfel încât $V(P) \cap X = \{x\}$ și $V(P) \cap Y = \{y\}$.

Notăm cu $\mathcal{P}(X, Y, G)$ familia tuturor XY -drumurilor din G . Observăm că dacă $x \in X \cap Y$ atunci $P = \{x\}$, de lungime 0, este un XY -drum.

Exemplu



XY -paths: (b, e, y) , (c, f, x) , and (a, g, y) ; an YX -path: (y, f, b)

Conexiune - Teorema lui Menger și aplicații

- Spunem că drumurile P_1 și P_2 sunt **disjuncte (pe noduri)** dacă $V(P_1) \cap V(P_2) = \emptyset$.
 - Motivată de problemele practice din rețelele de comunicații și de asemenei de studiile teoretice asupra conexiunii în (di)grafuri, este de interes determinarea mulțimilor de cardinal maxim de XY -drumuri disjuncte.
 - Notăm cu $p(X, Y; G)$ numărul maxim de XY -drumuri disjuncte în G .
 - Teorema care determină acest număr este datorată lui **Menger (1927)** și reprezintă unul dintre rezultatele fundamentale din Teoria grafurilor.

Definiția 2

Fie $G = (V, E)$ un (di)graf și $X, Y \subseteq V$. O mulțime XY -separatoare în G este orice submulțime $Z \subseteq V$ astfel încât

$$V(P) \cap Z \neq \emptyset, \text{ pentru fiecare } P \in \mathcal{P}(X, Y; G).$$

Notăm cu

$$\mathbf{S}(X, Y; G) = \{Z : Z \text{ este mulțime } XY\text{-separatoare din } G\} \text{ și}$$

$$k(X, Y; G) = \min \{|Z| : Z \in \mathbf{S}(X, Y; G)\}$$

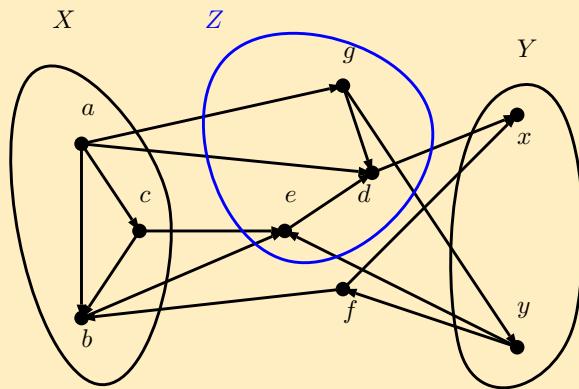
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Din definiție urmează că:

- Dacă $Z \in \mathbf{S}(X, Y; G)$, atunci $\mathcal{P}(X, Y; G \setminus Z) = \emptyset$.
- $X, Y \in \mathbf{S}(X, Y; G)$.

Conexiune - Teorema lui Menger și 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

A XY -separating set: $Z = \{g, e, d\}$

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

- Dacă $Z \in \mathbf{S}(X, Y; G)$, atunci $A \in \mathbf{S}(X, Y; G)$, $\forall A$ astfel încât $Z \subseteq A \subseteq V$.
- Dacă $Z \in \mathbf{S}(X, Y; G)$ și $T \in \mathbf{S}(Z, Y; G)$, atunci $T \in \mathbf{S}(X, Y; G)$.

Teorema 1

Teorema lui Menger. Fie $G = (V, E)$ un (di)graf și $X, Y \subseteq V$. Atunci $p(X, Y; G) = k(X, Y; G)$.

(I. e., numărul maxim de XY -drumuri disjuncte = cardinalul minim al unei multimi XY -separatoare.)

Demonstrație:

$k(X, Y; G) \geq p(X, Y; G) = r$. Fie P_1, \dots, P_r XY -drumuri disjuncte din G ; $Z \cap V(P_i) \neq \emptyset, \forall Z \in S(X, Y; G)$. Deoarece P_i sunt disjuncte ($i = \overline{1, r}$):

$$|Z| \geq \left| Z \cap \left(\bigcup_{i=1}^r V(P_i) \right) \right| = \sum_{i=1}^r |Z \cap V(P_i)| \geq \sum_{i=1}^r 1 = r.$$

Astfel, $|Z| \geq r, \forall Z \in S(X, Y; G)$; urmează că $k(X, Y; G) \geq r$.

Conexiune - Teorema lui Menger și aplicații

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

$k(X, Y; G) \leq p(X, Y; G)$. Omisă. (Vom arăta mai târziu că $\forall G = (V, E)$ și $\forall X, Y \subseteq V, \exists k(X, Y; G)$ XY -drumuri disjuncte în G folosind fluxuri în anumite rețele.) \square

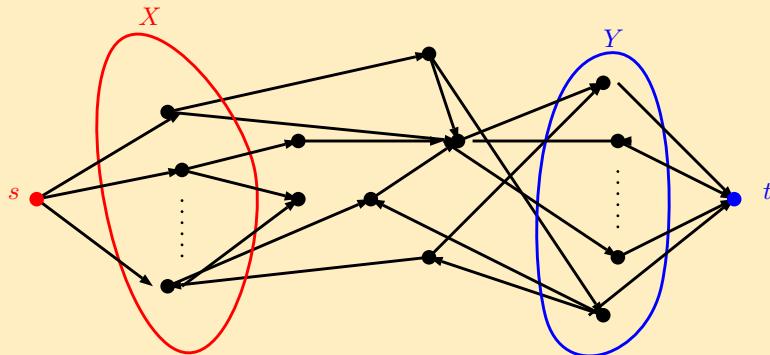
Menger (1927) a enunțat echivalent teorema de mai sus, utilizând drumuri intern-disjuncte: $P_1, P_2 \in \mathcal{P}_{st}$ astfel încât $V(P_1) \cap V(P_2) = \{s, t\}$:

Teorema 2

Fie $G = (V, E)$ un (di)graf și $s, t \in V$, astfel încât $s \neq t$, $st \notin E$. Există k drumuri intern-disjuncte de la s la t în G dacă și numai dacă există cel puțin un drum de la s la t în (di)graful obținut din G prin stergerea oricărei multimi de $< k$ noduri diferite de s și t .

Demonstrația echivalenței:

Teorema 1 \Rightarrow Teorema 2: luăm $X = N_G^+(s)$ ($N_G(s)$) și $Y = N_G^-(t)$ ($N_G(t)$).



Teorema 2 \Rightarrow Teorema 1: adăugăm două noi noduri s și t (di)grafului G , și toate muchiile (orientate) de la s la orice nod din X și de la orice nod din Y la t . \square

Conexiune - Teorema lui Menger și aplicații

Aplicații: p -conexiune

- Un graf G este p -conex ($p \in \mathbb{N}^*$) dacă fie $G = K_p$, fie $|G| > p$ și $G \setminus A$ este conex pentru orice $A \subseteq V(G)$ cu $|A| < p$.
- Din Teorema 2, o caracterizare echivalentă a p -conexiunii este:
Un graf G este p -conex ($p \in \mathbb{N}^*$) dacă fie $G = K_p$, fie $\forall st \in E(\overline{G})$ există p drumuri intern-disjuncte de la s la t în G .
- numărul de conexiune pe noduri al grafului G , $k(G)$, este cel mai mare p pentru care G este p -conex.
- Urmează că, pentru a calcula $k(G)$, trebuie aflat

$$\min_{st \notin E(G)} p(\{s\}, \{t\}; G)$$

care poate fi determinat în timp polinomial folosind fluxuri în rețelele.

Aplicații: Teorema lui König

- O acoperire cu noduri a grafului G este o mulțime $X \subseteq V(G)$ de noduri astfel încât $G - X$ este un graf nul (orice muchie din G are cel puțin o extremitate în X).
- Un caz special al Teoremei 1 se obține când G este bipartit iar $X = S$ și $Y = T$ sunt cele două clase ale bipartiției lui G :

Teorema 3

(König, 1931) Fie $G = (S, T; E)$ un graf bipartit. Atunci, cardinalul maxim al unui cuplaj din G este egal cu cardinalul minim al unei acoperiri cu noduri a lui G .

Demonstrație: Cardinalul maxim al unui cuplaj în G este $p(S, T; G) = k(S, T; G)$, din Teorema 1. Deoarece o mulțime de noduri este o mulțime ST -separatoare dacă și numai dacă este o acoperire cu noduri, Teorema 3 este dovedită. \square

Conexiune - Teorema lui Menger și aplicații

Aplicații: Teorema lui Hall

- Fie I și S mulțimi finite nevide. O familie submulțimi ale lui S (indexată după I) este o funcție $A : I \rightarrow 2^S$. Notăm $\mathcal{A} = (A_i)_{i \in I}$ și (folosind notația funcțională) $\mathcal{A}(J) = \bigcup_{j \in J} A_j$ (pentru $J \subseteq I$).
- O funcție de reprezentare pentru familia $\mathcal{A} = (A_i)_{i \in I}$ este orice funcție $r_{\mathcal{A}} : I \rightarrow S$ cu proprietatea $r_{\mathcal{A}}(i) \in A_i$, $\forall i \in I$; atunci, $(r_{\mathcal{A}}(i))_{i \in I}$ este numit un sistem de reprezentanți pentru \mathcal{A} .
- Dacă funcția de reprezentare, $r_{\mathcal{A}}$, este injectivă, atunci $r_{\mathcal{A}}(I)$ este o submulțime a lui S și este numită sistem de reprezentanți distincți pentru \mathcal{A} , sau o transversală a lui \mathcal{A} .
- Problema centrală în Teoria Transversalelor este de a caracteriza familiile care admit o transversală (cu anumite proprietăți). Teorema lui Hall (1935) este primul rezultat de acest tip.

Teorema 4

Hall, 1935 Familia $\mathcal{A} = (A_i)_{i \in I}$ de submulțimi ale lui S are o transversală dacă și numai dacă

$$(H) \quad |\mathcal{A}(J)| \geq |J|, \forall J \subseteq I.$$

Demonstrație: " \Rightarrow " Dacă $r_{\mathcal{A}}$ este o funcție de reprezentare injectivă pentru \mathcal{A} , atunci $r_{\mathcal{A}}(J) \subseteq \mathcal{A}(J)$, $\forall J \subseteq I$. Astfel, $r_{\mathcal{A}}$ fiind injectivă, $|\mathcal{A}(J)| \geq |r_{\mathcal{A}}(J)| = |J|$.

" \Leftarrow " Fie $G_{\mathcal{A}} = (I, S; E)$ graful bipartit asociat familiei \mathcal{A} (dacă $I \cap S \neq \emptyset$, putem considera copii izomorfe disjuncte): $E = \{(is | i \in I, s \in S \cap A_i)\}$. Se observă că $N_{G_{\mathcal{A}}}(i) = A_i$. Mai mult, \mathcal{A} are o transversală dacă și numai dacă $G_{\mathcal{A}}$ are un cuplaj de cardinal $|I|$.

Conexiune - Teorema lui Menger și aplicații

Demonstrația Teoremei lui Hall (continuare): Arătăm că dacă relația (H) are loc, atunci orice acoperire cu noduri a lui $G_{\mathcal{A}}$ are cel puțin $|I|$ noduri, și - din Teorema lui Konig - $G_{\mathcal{A}}$ are un cuplaj de cardinal $|I|$. Fie $X = I' \cup S' \subseteq I \cup S$ o acoperire cu noduri a lui $G_{\mathcal{A}}$: urmează că $N_{G_{\mathcal{A}}}(I \setminus I') \subseteq S'$, adică, $\mathcal{A}(I \setminus I') \subseteq S'$. Atunci,

$$|X| = |I'| + |S'| \geq |I'| + |\mathcal{A}(I \setminus I')|.$$

Deoarece are loc (H), obținem

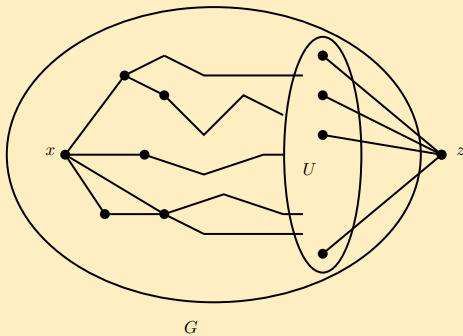
$$|X| \geq |I'| + |\mathcal{A}(I \setminus I')| \geq |I'| + |I \setminus I'| = |I|. \square$$

Aplicații: Teorema lui Dirac (structura grafurilor p -conexe)

Lemă

Fie $G = (V, E)$ un graf p -conex de ordin $|G| \geq p + 1$, $U \subseteq V$, $|U| = p$ și $x \in V \setminus U$. Atunci există p xU -drumuri astfel încât oricare două dintre ele îl au numai pe x drept nod comun.

Demonstrație: Fie $G' = (V \cup \{z\}, E')$, unde $E' = E \cup \{zu : u \in U\}$.



Conexiune - Teorema lui Menger și aplicații

Aplicații: Teorema lui Dirac (structura grafurilor p -conexe)

Demonstrație (continuare). Atunci, G' este un graf p -conex. Întradevăr, fie $A \subseteq V(G')$ cu $|A| \leq p - 1$. Dacă $A \subseteq V(G)$, atunci $G' - A$ este conex (din p -conexiunea lui G , $G - A$ este conex; cum $|A| < p$, $\exists u \in U \setminus A$ și, astfel, muchia zu este în $E(G' - A)$). Dacă $z \in A$, atunci $G' - A = G - A$ care este conex.

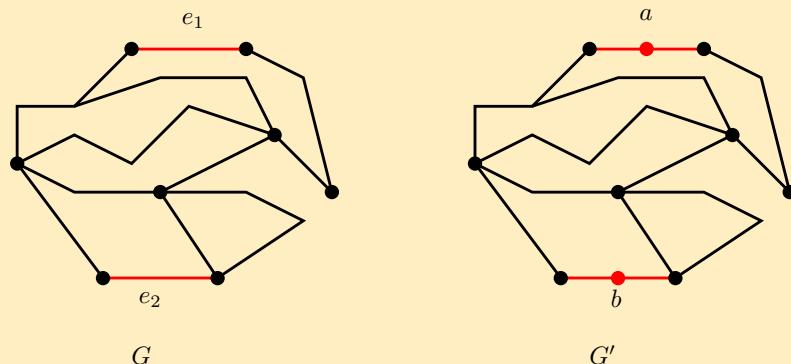
Lema urmează aplicând Teorema 2 grafului G' și perechii x, z . \square

Propoziție

Fie $G = (V, E)$ un graf p -conex, $p \geq 2$. Atunci, pentru orice două muchii e_1 și e_2 ale lui G și pentru orice, x_1, \dots, x_{p-2} , $p - 2$ noduri ale lui G , există un circuit în G care conține toate aceste muchii și noduri.

Demonstrație: Inducție după p .

Pentru $p = 2$, trebuie să arătăm că într-un graf 2-conex, G , orice două muchii e_1 și e_2 aparțin unui circuit. Fie G' graful obținut din G prin inserarea unui nod a pe e_1 și a unui nod b pe e_2 :



G' este 2-conex (orice graf de tipul $G' - v$ este conex). Astfel, există două drumuri intern-disjuncte de la a la b , care dau circuitul din G conținând e_1 și e_2 .

Conexiune - Teorema lui Menger și aplicații

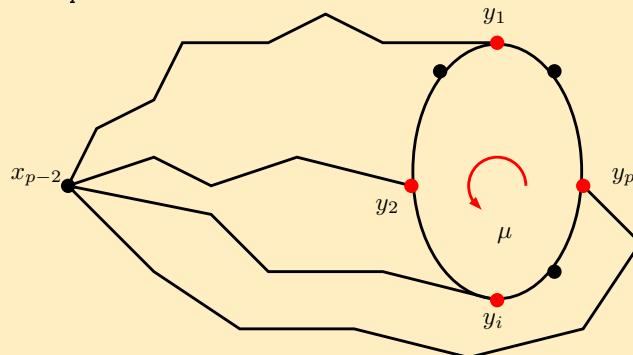
Demonstrație (continuare). În pasul inductiv, fie $p \geq 3$, presupunem că Propoziția este adevărată pentru orice graf p' -conex cu $2 \leq p' < p$, și considerăm un graf p -conex G , două dintre muchiile, sale e_1 și e_2 și o mulțime de $p - 2$ noduri $\{x_1, x_2, \dots, x_{p-2}\}$.

Putem presupune că nicio extremitate v a lui e_1 sau e_2 nu aparține mulțimii $\{x_1, x_2, \dots, x_{p-2}\}$ (altfel, aplicăm ipoteza inductivă și obținem că în graful $(p - 1)$ -conex, G , există un circuit C conținând e_1, e_2 și mulțimea de noduri $\{x_1, x_2, \dots, x_{p-2}\} \setminus \{v\}$; iar v este un nod al lui C deoarece e_1 și e_2 sunt muchii ale lui C).

Graful $G - x_{p-2}$ este $(p - 1)$ -conex. Din ipoteza inductivă, există un circuit μ care conține $x_1, x_2, \dots, x_{p-3}, e_1$ și e_2 . Fie Y mulțimea nodurilor lui μ . Evident, $|Y| \geq p$ (mulțimii de $p - 3$ noduri x_1, x_2, \dots, x_{p-3} , îi adăugăm cel puțin trei extremități ale muchiilor e_1 și e_2). Din Lema de mai sus, există p x_{p-2} - Y -drumuri astfel încât oricare două dintre ele au în comun doar un singur nod, x_{p-2} .²⁰⁴

Demonstrație (continuare). Fie $P_{x_{p-2}y_1}, P_{x_{p-2}y_2}, \dots, P_{x_{p-2}y_p}$ aceste drumeuri, unde ordinea y_1, \dots, y_p se obține în urma unei parcurgeri a lui μ .

Nodurile y_1, \dots, y_p împart circuitul μ în drumurile $P_{y_1y_2}, P_{y_2y_3}, \dots, P_{y_{p-1}y_p}, P_{y_py_1}$:



Cel puțin unul dintre drumeurile de mai sus nu conține niciun element din mulțimea $x_1, x_2, \dots, x_{p-3}, e_1$ și e_2 (pigeon hole principle).

Fie $P_{y_1y_2}$ acest drum (altfel, renumerotăm nodurile y_i).

Conexiune - Teorema lui Menger și aplicații

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

Demonstrație (continuare). Atunci,

$$P_{x_{p-2}y_2}, P_{y_2y_3}, \dots, P_{y_py_1}, P_{y_1x_{p-2}}$$

este circuitul din G care conține $x_1, x_2, \dots, x_{p-2}, e_1$ și e_2 . \square

Teorema 5

(Dirac, 1953) Prin orice $p \geq 2$ noduri ale unui graf p -conex trece un circuit.

Demonstrație: Fie $G = (V, E)$ un graf p -conex, $p \geq 2$. Fie x_1, x_2, \dots, x_p p noduri ale lui G . Deoarece G este conex, există muchiile $e_1 = xx_{p-1}$ și $e_2 = yx_p$. Atunci, teorema urmează din Propoziția de mai sus. \square

O aplicație interesantă a acestei teoreme (și a demonstrației propoziției) este următoarea condiție suficientă pentru ca un graf să fie Hamiltonian dată de Erdős și Chvatal.

Teorema 6

(Erdős-Chvatal, 1972) Fie $G = (V, E)$ un graf p -conex. Dacă $\alpha(G) \leq p$ atunci G este graf Hamiltonian.

Demonstrație: Să presupunem, prin contradicție că G nu este Hamiltonian. Fie C un cel mai lung circuit din G .

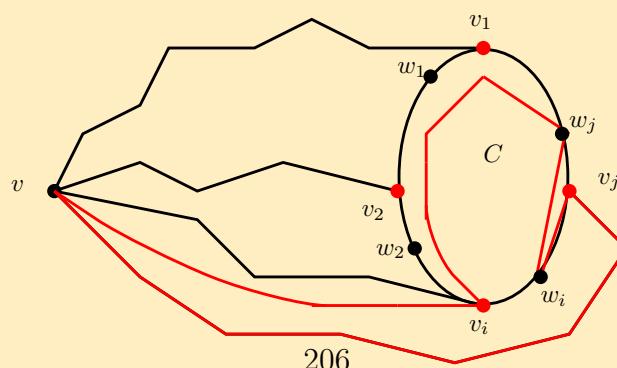
Din Teorema lui Dirac $|C| \geq p$ și din presupunerea noastră, există un nod $v \in V(G) \setminus V(C) \neq \emptyset$.

Conexiune - Teorema lui Menger și aplicații

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

Demonstrație (continuare). Cum $|C| \geq p$, putem repeta argumentul din demonstrația Propoziției de mai sus pentru a arăta că există P_{vv_1} , $P_{vv_2}, \dots, P_{vv_p}$, p vC -drumuri care se intersectează două câte două doar în v și cu extremități v_i etichetate în ordinea în care apar la o parcurgere a circuitului.

Fie w_i succesorul nodului v_i pe circuit.



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

Demonstrație (continuare). Observăm că $vw_i \notin E$ (în caz contrar, circuitul
 $vw_i, w_i, C \setminus \{w_i v_i\}, P_{v_i v}$ este mai lung decât C , contradicție).

Deoarece $\alpha(G) \leq p$, mulțimea $\{v, w_1, w_2, \dots, w_p\}$ nu este stabilă, și din
remarca de mai sus, urmează că există o muchie $w_i w_j \in E$.

Dar atunci, P_{vv_i} , inversul drumului de la v_i la w_j de pe circuit, muchia
 $w_j w_i$, drumul de la w_i la v_j de pe circuit, și drumul $P_{v_j v}$ oferă un circuit
mai lung decât C , contradicție). \square

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

Arbore - Elemente de bază

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

Un **arbore** este un graf conex fără circuite.

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

Teorema 7

Fie $G = (V, E)$ un graf. Atunci următoarele afirmații sunt echivalente:

- (i) G este un arbore (este conex și nu are circuite).
- (ii) G este conex și este minimal cu această proprietate.
- (iii) G nu are circuite și este maximal cu această proprietate.

Demonstrație: Omisă. \square

Minimalitatea și maximalitatea din afirmațiile de mai sus sunt relativ
la relația de ordine parțială dată de incluziune pe submulțimile muchii.
Mai precis afirmațiile (ii) și (iii) înseamnă:

- (ii) G este conex și $\forall e \in E$, $G - e$ nu este conex.
 - (iii) G nu are circuite și $\forall e \notin E$, $G + e$ are un circuit.

Fie $G = (V, E)$ un (multi)graf. Un **arbore parțial** G este un graf parțial al lui G , $T = (V, E')$ ($E' \subseteq E$), care este arbore. Notăm cu \mathcal{T}_G mulțimea tuturor arborilor parțiali ai lui G .

Remarci

1. $\mathcal{T}_G \neq \emptyset$ dacă și numai dacă G este conex. Într-adevăr, dacă $\mathcal{T}_G \neq \emptyset$, atunci există un arbore parțial $T = (V, E')$ al lui G . T este conex, deci între orice două noduri ale lui G există este un drum P în T . Deoarece $E' \subset E$, P este un drum și în G , deci G este conex.

Arbore - Elemente de bază

Reciproc, dacă G este conex, atunci considerăm următorul algoritm:

```

 $T \leftarrow G;$ 
while ( $\exists e \in E(T)$  astfel încât  $T - e$  este conex) do
     $T \leftarrow T - e;$ 

```

Din construcție, T este graf parțial al lui G , și are loc afirmația (ii) din Teorema 7, deci T este un arbore.

2. O altă demonstrație constructivă (dacă G este conex atunci $\mathcal{T}_G \neq \emptyset$) se bazează pe observația că există o muchie în cross între cele două clase ale oricărei bipartitii a lui V : $\exists e = v_1v_2 \in E$ cu $v_i \in V_i$, $i = \overline{1, 2}$.

Dacă $|V| = n > 0$ atunci următorul algoritm construiește un arbore parțial al grafului conex $G = (V, E)$:

```

 $k \leftarrow 1; T_1 \leftarrow (\{v\}, \emptyset); // v \in V$ 
while ( $k < n$ ) do
  fie  $xy \in E$  cu  $x \in V(T_k)$ ,  $y \in V \setminus V(T_k)$ ;
  // o astfel de muchie există din conexiunea lui  $G$ 
   $V(T_{k+1}) \leftarrow V(T_k) \cup \{y\}$ ;
   $E(T_{k+1}) \leftarrow E(T_k) \cup \{xy\}$ ;
   $k++$ ;
  
```

Evident, T_k este un arbore $\forall k = \overline{1, n}$ (inductiv, dacă T_k este un arbore atunci, din construcție, T_{k+1} este conex și nu are circuite). Mai mult, avem $|V(T_k)| = k$ și $|E(T_k)| = k - 1$, $\forall k = \overline{1, n}$.

3. Dacă această construcție este aplicată unui arbore G cu n noduri, vom obține că G are $n - 1$ muchii. Această proprietate poate fi folosită pentru a extinde Teorema 7 cu alte caracterizări ale arborilor:

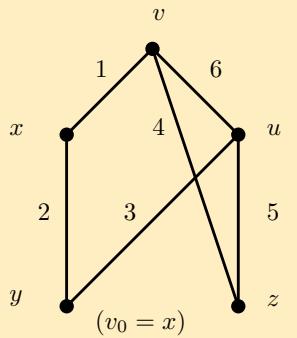
Teorema 8

Următoarele afirmații sunt echivalente pentru un graf $G = (V, E)$ cu n noduri:

- (i) G este un arbore.
- (ii) G este conex și are $n - 1$ muchii.
- (iii) G nu are circuite și are $n - 1$ muchii.
- (iv) $G = K_n$ pentru $n \in \{1, 2\}$, iar pentru $n \geq 3$ $G \neq K_n$ și $G + e$ are exact un circuit, pentru orice muchie $e \in E$.

Demonstrație: Omisă. \square

- Descriem o metodă simplă de tip **backtracking** pentru a genera toți arborii parțiali ai unui graf conex $G = (V, E)$, unde $V = \{1, \dots, n\}$, $|E| = m$.
- Multimea de muchii, E , va fi reprezentată cu un tablou $E[1..2, 1..m]$ cu elemente din V , cu semnificația: dacă $v = E[1, i]$ și $w = E[2, i]$, atunci vw este muchia i a lui G . Mai mult, vom presupune că primele $d_G(v_0)$ coloane din tabloul E au v_0 în linia 1 ($E[1, i] = v_0$, $\forall i = 1, d_G(v_0)$), pentru $v_0 \in V$.



1	2	3	4	5	6
x	x	y	z	z	u
v	y	u	v	u	v

- Un arbore parțial $T \in \mathcal{T}_G$ va fi reprezentat ca o mulțime de $n - 1$ indecsi (în ordine crescătoare) ai coloanelor din tabloul E (desemnându-i muchiile).
- Întimpul generării, menținem un vector $T[1..n - 1]$ cu elemente din $\{1, \dots, m\}$ și o variabilă flag $i \in \{1, \dots, n\}$ cu următoarele semnificații:
Căutăm toți arborii parțiali ai lui G , cu proprietatea că cele mai mici $i - 1$ muchii sunt: $T[1] < T[2] < \dots < T[i - 1]$.
- Pentru exemplul de mai sus, dacă $i = 2$, $T[1] = 1$, și $T[2] = 2$, atunci arborii care vor fi găsiți sunt $\{1, 2, 3\}$, $\{1, 2, 5\}$, și $\{1, 2, 6\}$. Dacă, $i = 2$, $T[1] = 3$, și $T[2] = 5$ atunci arborele care trebuie găsit este $\{3, 5, 6\}$. Dar dacă $i = 2$, $T[1] = 1$, și $T[2] = 6$, niciun arbore nu va fi găsit.

ALL-ST-Gen(i)

```
// sunt generați toți arborii parțiali  $G$ , cu cele mai mici  $i - 1$  muchii:  $T[1], \dots, T[i - 1]$ 
if ( $i = n$ ) then
    //  $\{T[1], \dots, T[n - 1]\}$  este arbore parțial
    process( $T$ ); // printează, memorează etc
else
    if ( $i = 1$ ) then
        for ( $j = \overline{1, d_G(v_0)}$ ) do
             $T[i] \leftarrow j$ ; A All-ST-Gen( $i + 1$ ) B
    else
        for ( $j = \overline{T[i - 1] + 1, m - (n - 1) + i}$ ) do
            if ( $\langle\{T[1], \dots, T[i - 1]\} \cup \{j\}\rangle_G$  has no circuit) then
                 $T[i] \leftarrow j$ ; A All-ST-Gen( $i + 1$ ) B
```

Arbore - Generarea \mathcal{T}_G

- Prin apelul All-ST-Gen(1) obținem \mathcal{T}_G .
- Pentru a testa dacă graful $\langle\{T[1], \dots, T[i - 1]\} \cup \{j\}\rangle_G$ nu are circuite, observăm că, din construcție,

$$\langle\{T[1], \dots, T[i - 1]\}\rangle_G$$

nu are circuite, deci este o pădure (fiecare componentă conexă este un arbore).

- Fie $root[1..n]$ un vector (global) cu elemente din V și semnificația: $root[v] =$ rădăcina componentei conexe care conține v (unul dintre nodurile sale).
- Înaintea apelului All-ST-Gen(1), vectorul $root$ este inițializat pentru a satisface proprietatea: $root[v] \leftarrow v$ ($\forall v \in V$) (deoarece atunci, $\{T[1], \dots, T[i - 1]\} =_{2\varnothing}$).

- În timpul apelurilor recursive, când se testează dacă muchia j poate fi adăugată mulțimii $\{T[1], \dots, T[i-1]\}$ fără a crea vreun circuit, fie $v = E[1, j]$ și $w = E[2, j]$. Atunci, $\langle\{T[1], \dots, T[i-1]\} \cup \{j\}\rangle_G$ nu are circuite dacă și numai dacă v și w sunt în componente conexe diferite ale pădurii, i.e., $root[v] \neq root[w]$.
- Pentru a actualiza vectorul $root$, în locurile marcate cu **A** și **B** din algoritm, trebuie făcute următoarele modificări.
- În loc de **A**:

```
 $S \leftarrow \emptyset; x \leftarrow root[v];$ 
for ( $u \in V$ ) do
    if ( $root[u] = x$ ) then
         $S \leftarrow S \cup \{u\}; root[u] \leftarrow root[w];$ 
```

- Cu alte cuvinte toate nodurile din arborele cu rădăcina x sunt adăugate arborelui cu rădăcina $root[w]$; aceste noduri sunt salvate în mulțimea S .
- După apelul All-ST-Gen($i + 1$), vector $root$ trebuie setat din nou la valoarea dinaintea apelului, aceasta poate fi făcută **B** prin:


```
for ( $u \in S$ ) do
         $root[u] = x;$ 
```

Fie $G = (V, E)$ un multi-graf cu $V = \{1, 2, \dots, n\}$, și matricea de adiacență $A = (a_{ij})_{n \times n}$ (a_{ij} = multiplicitatea muchiei ij dacă $ij \in E$, 0 altfel). Fie

$$D = \text{diag}(d_G(1), d_G(2), \dots, d_G(n)) = \begin{pmatrix} d_G(1) & 0 & \dots & 0 \\ 0 & d_G(2) & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & d_G(n) \end{pmatrix}.$$

Matricea Laplaciană a lui G (sau Laplacianul) este definită ca fiind:

$$L[G] = D - A.$$

Observăm că suma tuturor elementelor din fiecare linie sau din fiecare coloană a lui $L[G]$ este 0. Notăm cu $L[G]_{ij}$ minorul matricii $L[G]$ obținut prin stergerea liniei i și a coloanei j .

Arbore - Numărarea arborilor parțiali

Teorema 9

(Kirchoff-Trent). Fie G un (multi)graf cu mulțimea nodurilor $\{1, \dots, n\}$ și Laplacianul $L[G]$. Atunci, numărul arborilor parțiali ai lui G este: $|\mathcal{T}_G| = \det(L[G]_{ii})$, $\forall 1 \leq i \leq n$.

Demonstrație: Omisă. \square

Corolar

(Formula lui Cayley). $|\mathcal{T}_{K_n}| = n^{n-2}$.

Demonstrație:

$$L[K_n] = \begin{pmatrix} n-1 & -1 & \dots & -1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{pmatrix}.$$

Astfel:

$$\det(L[K_n]_{11}) = \begin{vmatrix} n-1 & -1 & \dots & -1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{vmatrix}$$

Dacă adunăm toate liniile la prima obținem

$$\begin{vmatrix} 1 & 1 & \dots & 1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{vmatrix} = n^{n-2}. \text{ (De ce?)}$$

□

Exerciții pentru seminarul de săptămâna viitoare

Exercițiul 1. Fie $G = (V, E)$ un graf conex și $v \in V$ astfel încât $N_G(v) \neq V \setminus \{v\}$. Pentru $X \subseteq V$ notăm $N_G(X) = \left(\bigcup_{v \in X} N_G(v) \right) \setminus X$.

Evident, mulțimea $A = \{v\}$ satisface următoarele proprietăți:

- (i) $v \in A$ și $[A]_G$ este conex.
- (ii) $N = N_G(A) \neq \emptyset$.
- (iii) $R = V \setminus (A \cup N) \neq \emptyset$.
 - (a) Arătați că, dacă $A \subseteq V$ este orice mulțime noduri care satisface (i) - (iii) și maximală (relativ la " \subseteq ") cu aceste proprietăți, atunci $\forall x \in R$ și $\forall y \in N$ avem $xy \in E$.
 - (b) Dovediți că dacă A este ca la (a) și G este $\{C_k\}_{k \geq 4}$ -free, atunci N este o clică în G .
 - (c) Deducreți că K_n ($n \in \mathbb{N}^*$) sunt singurele grafuri regulate, triangulate și conexe.

Exercițiu 2. Un graf de ordin cel puțin trei este numit **confidențial conex** dacă, pentru orice trei noduri distincte a, b și c , există un drum de la a la b astfel încât c este diferit de și nu este adiacent cu niciun nod intern (dacă există) al acestui drum. (Un exemplu de graf confidențial conex este graful complet K_n , cu $n \geq 3$.)

Arătați că un graf conex, necomplet, $G = (V, E)$, cu cel puțin trei noduri este confidențial conex dacă și numai dacă:

- (i) pentru orice $v \in V$, $N_{\overline{G}}(v) \neq \emptyset$ și induce un subgraf conex;
- (ii) orice muchie a lui G face parte dintr-un C_4 indus sau este muchia mediană a unui P_4 indus.

Exercițiu 3. Dovediți că un graf conex, p -regulat și bipartit este 2-conex.

Exerciții pentru seminarul de săptămâna viitoare

Exercițiu 4. Fie $G = (V, E)$ un digraf. Demonstrați că:

- (a) G este tare conex dacă și numai dacă pentru orice $S \subsetneq V$, $S \neq \emptyset$, există măcar un arc care pleacă din S .
- (b) Dacă G este tare conex și poate fi deconectat prin stergerea a cel mult p arce (i. e., $\exists A \subseteq E$, $|A| \leq p$ astfel încât $G - A$ nu este tare conex), atunci G poate fi deconectat prin inversarea a cel mult p arce (adică $\exists B \subseteq E$, $|B| \leq p$ astfel încât $G' = (V, (E \setminus B) \cup \{uv : vu \in B\})$ nu este tare conex).

Exercițiu 5. Fie G un graf 2-muchie-conex ($G - e$ este conex, $\forall e \in E(G)$). Definim următoarea relație binară $e \asymp f$ dacă $e = f$ or $G - \{e, f\}$ nu este conex.

- (a) Arătați că $e \asymp f$ dacă și numai dacă e și f aparțin acelorași circuite.
- (b) Arătați că o clasă de echivalență $[e]_{\asymp}$ este inclusă într-un circuit.
- (c) Ștergând toate muchiile dintr-o clasă de echivalență $[e]_{\asymp}$, componentele conexe ale grafului rămas sunt grafuri 2-muchie-conexe.

Exercițiu 6.

- (a) Fie G un graf cu cel puțin 3 noduri. Dacă G este 2-conex, atunci putem să-i orientăm muchiile aşa încât graful orientat rezultat să fie tare conex.
- (b) Reciproca afirmației de mai sus este adevărată?

Exercițiu 7.

- (a) Fie G un graf 2-conex, necomplet și $xy \in E(G)$. Arătați că $G - xy$ sau $G|xy$ este 2-conex.
- (b) Dați câte un exemplu de un graf G și o muchie $xy \in E(G)$ astfel ca: (b1) $G - xy$ și $G|xy$ sunt 2-conexe; (b2) $G - xy$ nu este 2-conex dar $G|xy$ este 2-conex; (b3) $G - xy$ este 2-conex dar $G|xy$ nu este 2-conex;

Exerciții pentru seminarul de săptămâna viitoare

Exercițiu 9. Fie $G = (V, E)$ un graf conex și T_1, T_2 doi arbori parțiali ai lui G ($T_1, T_2 \in \mathcal{T}_G$).

- (a) Dovediți că T_1 poate fi transformat în T_2 prin aplicarea repetată a următoarei proceduri: șterge o muchie și adaugă o altă muchie arborelui curent.
- (b) Dacă, în plus, G este 2-conex arătați că T_1 poate fi transformat în T_2 prin aplicarea repetată a următoarei proceduri: șterge o muchie uv și adaugă o altă muchie uw arborelui curent.

Exercițiu 10. Demonstrați că multimea de muchii a unui graf complet K_n ($n \geq 2$) poate fi partionată în $\lceil n/2 \rceil$ submulțimi fiecare reprezentând multimea de muchii ale unui arbore (subgraf al lui K_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 -

Exercițiu 11. Fie n un întreg pozitiv și $G_n = (V, E)$ un graf definit astfel:

- $V = \{(i, j) : 1 \leq i, j \leq n\}$;
- $(i, j)(k, l) \in E$ (pentru $(i, j) \neq (k, l)$ din V) dacă și numai dacă $i = l$ sau $j = k$.

Arătați că G_n este universal pentru familia arborilor de ordin n : pentru orice arbore T de ordin n , $\exists A \subseteq V$ astfel încât $T \cong [A]_{G_n}$.

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

Exercițiu 12. Arătați că un turneu este tare conex dacă și numai dacă are un circuit hamiltonian.

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

Exercițiu 13. Se consideră rețeaua stradală a unui oraș. Se știe că putem îndepărta posibilitatea de a merge în cerc în oraș pe această rețea prin blocarea a cel mult p sensuri de străzi (prin blocarea unui singur sens se înțelege obstrucționarea sensului vizat de pe stradă). Arătați că putem îndepărta posibilitatea de a merge în cerc în oraș pe rețeaua sa stradală prin inversarea a cel mult p sensuri de străzi.

(Inversarea unui sens pe o stradă cu două sensuri implică păstrarea doar a celuilalt sens; inversarea sensului unei străzi cu un singur sens implică introducerea celuilalt sens.)

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

Exercițiu 14. Câți arbori parțiali are un graf bipartit complet $K_{n,n}$? Aceeași întrebare pentru $K_{p,q}$.

- Graph Algorithms * C. Croitoru - Graph Algo217ns * C. Croitoru - Graph Algorithms *

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

Exercițiu 1. Soluție.

(a) Presupunem prin reducere la absurd că există două noduri ne-adiacente $x \in N$ și $y \in R$.

- fie $A' = A \cup \{x\}$; $[A']_G$ este un subgraf conex (de ce?);
- $N_G(A') \neq \emptyset$ deoarece G is conex și $N_G(A') \neq V$ deoarece $y \notin N_G(A')$;
- $y \in V \setminus (A' \cup N_G(A'))$;

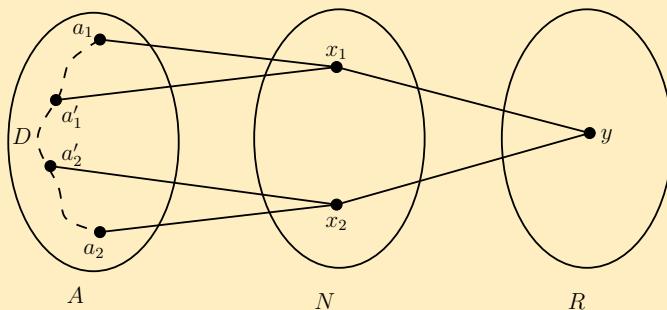
A' are proprietățile (i)-(iii) și conține strict A - contradicție (de ce?).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

(b) Presupunem prin reducere la absurd că N nu este o clică: există două noduri ne-adiacente $x_1, x_2 \in N$.



- Un circuit indus de lungime cel puțin 4 (where?) poate fi observat în figura de mai jos - contradicție.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo218ns * C. Croitoru - Graph Algorithms *

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

(c) Presupunem prin reducere la absurd că G nu este graf complet, atunci va exista un nod $v \in V$ cu $N_G(v) \neq V \setminus \{v\}$, și o submulțime $A \subset V$ maximală cu proprietățile (i)-(iii) (de ce?).

- $d_G(y) \leq |N| + |R| - 1$ (de ce?).
 - $d_G(x) \geq |N| + |R|$ (de ce?).
 - $|N| + |R| - 1 \geq d_G(y) = d_G(x) \geq |N| + |R|$ - contradicție.

Exercitii rezolvate (parțial)

Exercitiul 3. Solutie

- Fie $G = (S, T, E)$ un graf p -regulat bipartit conex și $u \in S$.
 - Presupunem prin reducere la absurd că $G - u$ nu este conex, atunci există o bipartiție (V_1, V_2) a lui $V(G) \setminus \{u\}$ astfel că nicio muchie nu conectează V_1 și V_2 (**de ce?**).
 - Fie $G_1 = [V_1 \cup \{u\}]_G$ care este un graf bipartit (**de ce?**),
 - Fie $S_1 = S \cap V(G_1)$ și $T_1 = T \cap V(G_1)$, presupunem că $u \in S_1$, atunci

$$\sum_{x \in S_1} d_{G_1}(x) = \sum_{y \in T_1} d_{G_1}(y) \Rightarrow p \cdot (|S_1| - 1) + d_{G_1}(u) =$$

$p \cdot |T_1| \Rightarrow p | d_{G_1}(u) \Rightarrow d_{G_1}(u) = p$ - de ce?

- Aceasta e o contradicție. De ce?

Exercițiu 4. Soluție.

- (a) " \implies " Dacă G nu este tare conex nu putem ajunge la un nod din $V(G) \setminus S \neq \emptyset$ plecând dintr-un nod din $S \neq \emptyset$ (de ce?) - contradicție.
- " \impliedby " Fie $u, v \in V(G)$; este suficient să arătăm că exist un uv -drum în G .
 - Fie $u = u_1$, există un arc de la $\{v_1\}$ la un nod $v_2 \in V(G) \setminus \{v_1\}$ (de ce?). Dacă am definit deja sirul $v_1, v_2, \dots, v_p \in V(G)$ astfel că există $v_1 v_i$ -drumuri, $\forall 2 \leq i \leq p$, atunci există un arc $v_j w$ cu $w \notin \{v_1, v_2, \dots, v_p\}$ și $1 \leq j \leq p$ (de ce?).
 - Putem defini $v_{p+1} = w$.
 - Repetăm această procedură până când $v_p = v$.

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

Exerciții rezolvate (partial)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $A \subseteq E$ o mulțime de cardinal minim așa încât $G - A$ nu este tare conex ($|A| \leq p$).
- Există $S \subsetneq V(G) \setminus A$, $S \neq \emptyset$ așa încât nu există arce de la S la $V(G - A) \setminus S$ (de ce?).
- Cum A este minimală (de ce?), orice arc $e \in A$ leagă un nod din S cu unul din $V(G - A) \setminus S$ - altfel prin ștergerea din G a arcelor din $A \setminus \{e\}$ obținem un digraf care nu este tare conex - contradicție.
- Fie A' mulțime arcelor inverse ale celor din A . Evident că $G - A'$ nu este tare conex (de ce?) și $|A'| = |A| \leq p$.

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo220ns * C. Croitoru - Graph Algorithms *

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

(a) " \implies " Fie $e \asymp f$, $e \neq f$ și C un circuit care conține e .

- Dacă $f \notin E(C)$, atunci $G - e$ are un circuit care trece prin f (**de ce?**), astfel $G - \{e, f\}$ este conex, contradicție.
- " \Leftarrow " Fie $e \neq f$; $G - e$ este conex.
- Dacă presupunem că $G - \{e, f\}$ este conex, atunci trebuie să existe un circuit C în $G - e$ care să conțină f (**de ce?**), astfel C nu conține simultan și pe e și pe f - contradicție.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

(b) Se folosește (a). **Cum?**

(c) Fie $e \in E(G)$, $F = [e]_{\asymp}$ și H o componentă conexă a lui $G - F$.

- Fie $f \in E(H) \subseteq E(G) \setminus F$.
- $G - e$ și $G - \{e, f\}$ sunt conexe (**de ce?**), astfel există un circuit C în $G - e$ care conține f .
- $E(C) \cap F = \emptyset$ (**de ce?**), astfel C este un circuit în $G - F$ și în H , i. e., $H - f$ este conex.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo221ns * C. Croitoru - Graph Algorithms *

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

Exercițiu 6. Soluție.

(a) Fie $G = (V, E)$ un graf 2-conex.

- Parcurgem dfs graful G reținând ordinea de vizitare a nodurilor; muchiile traversate sunt orientate de la nodul de indice mai mic la cel de indice mai mare.
- Muchiile rămase netraversate se orientează invers.
- Procedura 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
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții rezolvate (partial)

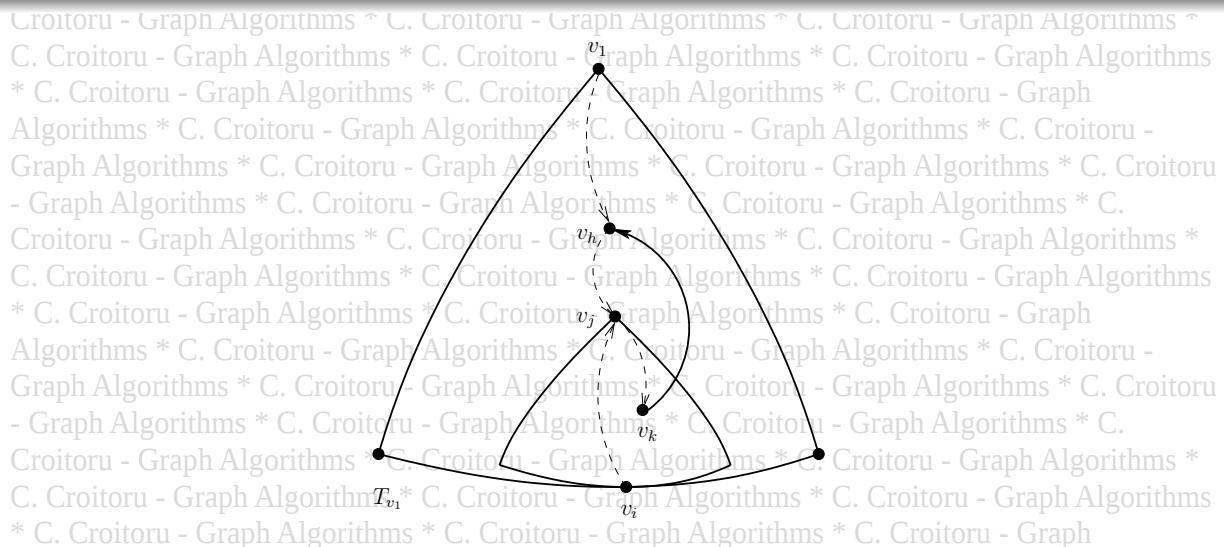
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

```
dfs(v) {  
    k++; v_k ← v; order[v] ← k;  
    for (u ∈ A(v))  
        E' ← E' ∪ {vu};  
        if (order[u] == 0) {  
            dfs(u);  
        }  
        E' ← E' ∪ {uv};  
}
```

- Algoritmul principal:

```
k ← 0; E' ← ∅;  
for (v ∈ V)  
    order[v] ← 0;  
dfs(r);
```

- Nu există muchii în cross după parcurgerea dfs , i. e. orice arc $v_i v_j$, cu $i > j$, merge către un strămoș al lui v_i . Notăm cu T_v subarborele dfs cu rădăcina în v .



- Se poate demonstra că de la fiecare frunză v_i există un drum către v_1 în $\vec{G} = (V, E')$. Cum?

Exerciții rezolvate (partial)

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

- (b) Nu. $G = (\{x, y, z, t, u\}, \{xy, yz, zx, zt, tu, uz\})$ este tare conex (de ce?), dar $G - z$ nu este conex.

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

Exercițiul 7. Soluție.

- (a) Presupunem prin reducere la absurd că $G - xy$ și $G|xy$ nu sunt 2-conexe.
- Cum $G - x$ este conex și $G - x \subseteq G - xy$, $G|xy$, urmează că $G - xy$ și $G|xy$ sunt conexe (de ce?).
 - Fie a nodul care înlocuiește x și y în G după contracție.
 - Fie z un nod astfel încât $G|xy - z$ nu este conex, atunci $z = a$, altfel $G - z$ fiind conex, $G|xy - z = (G - z)|xy$ va fi conex de asemenea (de ce?).
 - Astfel $G|xy$ poate fi deconectat doar prin stergerea lui 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 -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

- Fie $u \in V(G)$ un nod astfel încât $G - xy - u$ nu este conex.
- Urmează că $u \notin \{x, y\}$, și dacă G_1, G_2 sunt două componente
conexe din $G - xy - u$, atunci xy este singura muchie dintre G_1 și
 G_2 din $G - u$ (de ce?).
- Dacă contractăm xy și apoi îl stergem, u asigură conexiunea grafului
rămas.
- Astfel $G|xy - a$ nu poate fi deconectat..

(b) (b1) K_4 ; (b2) $C_4 + e$ și $xy \neq e$; (b3) $C_4 + e$ și $xy = e$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 9. Soluție.

- (a) Fie $G = (V, E)$, $|G| = n$ și $T_1, T_2 \in \mathcal{T}_G$, $T_1 \neq T_2$, $T_i = (V, E_i)$.
- Demonstrăm prin inducție după $2k = |E_1 \Delta E_2|$ că are loc proprietatea din enunț..
 - Este ușor de verificat proprietatea când $|E_1 \Delta E_2| = 2$ ($\Leftrightarrow T_1 = T_2$).
 - Pasul inductiv: presupunem că pentru orice doi arbori parțiali T'_1 and T'_2 cu $|E'_1 \Delta E'_2| = 2k \geq 2$, proprietatea are loc.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo224ns * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $T_1, T_2 \in \mathcal{T}_G$, $|E_1 \Delta E_2| = 2k + 2$. alegem o muchie $e_1 \in E_1 \setminus E_2$.
- $T_2 + e_1$ conține un singur circuit C (de ce?).
- Cum $E(C) \not\subseteq E_1$, va exista o muchie $e_2 \in E(C) \cap (E_2 \setminus E_1)$.
- $T' = T_2 + e_1 - e_2 \in \mathcal{T}_G$, $|E_2 \Delta E(T')| = 2$ (de ce?) și $|E_1 \Delta E(T')| = 2k$.
- Din ipoteza inductivă știm că putem transforma T_1 în T' folosind procedura de mai sus de un număr de ori după care putem transforma T' în T_2 folosind procedura încă o dată.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiul 11. Soluție.

- Fie T = un arbore de ordin n ; parcurgem bfs arborele T și etichetăm nodurile astfel:
 - primul nod (rădăcina) primește eticheta $(1, 1)$;
 - orice alt nod v primește eticheta (i, j) , unde j este numărul de ordine bfs al lui v iar i este numărul de ordine bfs al tatălui său.

Evident $T \simeq [\Lambda]_{G_n}$, unde Λ este mulțimea etichetelor construite mai sus. De ce?

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo225ns * C. Croitoru - Graph Algorithms *

Exercițiul 13. Soluție. Reformulare: Fie $G = (V, E)$ un digraf fără bucle. Dacă putem transforma G într-un dag ștergând cel mult p arce (adică $\exists A \subseteq E$, $|A| \leq p$ astfel încât $G - A$ nu are circuite), atunci putem transforma G într-un dag inversând cel mult p arce (i. e., $\exists B \subseteq E$, $|B| \leq p$ astfel încât $G' = (V, (E \setminus B) \cup \{uv : vu \in B\})$ nu are circuite).

- Fie $A \subseteq E$, o mulțime de cardinal minim astfel încât $G - A$ este un dag ($|A| \leq p$ - **de ce?**).
- $G - A$ fiind un dag există o ordonare topologică a $V(G - A) = V(G)$: v_1, v_2, \dots, v_n astfel încât $v_i v_j \in E(G - A)$ implica $i < j$.
- Pentru fiecare arc $v_k v_l \in A$ digraful $G - A + v_k v_l$ conține circuite, i. e., $k > l$ (**de ce?**).
- Astfel toate arcele din A sunt de acest tip, dar inversându-le se păstrează ordonarea topologică în $(G - A) \cup \{uv : vu \in A\}$.

6 CURS 6: Arbori partiali de cost minim. Cuplaje maxime si acoperiri minime.

cuplaje - multimi de muchii care nu au niciun nod in comun (muchii izolate)

numar de cuplaj - cardinalul maxim al unui cuplaj (cate muchii are cel mai mare cuplaj pe care poti sa l faci)

acoperire cu muchii - o multime de muchii care sa acopere(atinga) toate nodurile (vezi in exemplul de mai sus)

arbori partiali de cost minim - tre sa scoti din cicluri muchiile de cost maxim

Algoritmica grafurilor - Cursul 6

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

6 noiembrie 2020

Cuprins

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

1 Problema arborelui parțial de cost minim

• **Metoda generală MST**

• **Algoritmul lui Prim**

• **Algoritmul lui Kruskal**

2 Cuplaje

• **Cuplaje maxime – Acoperire minimă cu muchii**

3 Exerciții pentru seminarul din săptămâna următoare

4 Exerciții rezolvate (parțial)

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

Problema MST. Dat $G = (V, E)$ un graf și $c : E \rightarrow \mathbb{R}$ ($c(e)$ este costul
 muchiei e) găsiți $T^* \in \mathcal{T}_G$ astfel încât

$$c(T^*) = \min_{T \in \mathcal{T}_G} c(T),$$

unde $c(T) = \sum_{e \in E(T)} c(e)$.

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

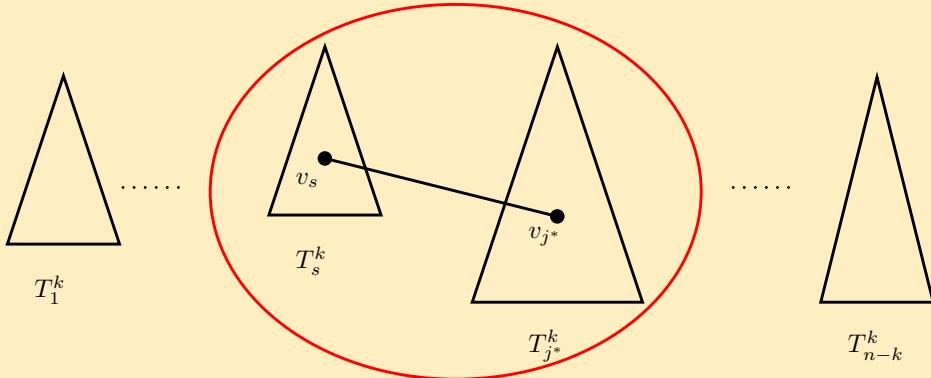
Metoda generală MST

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

- Se pornește cu familia $\mathcal{T}^0 = (T_1^0, T_2^0, \dots, T_n^0)$ de arbori disjuncți:
 $T_i^0 = (\{i\}, \emptyset)$, $i = \overline{1, n}$.
- La fiecare pas k ($0 \leq k \leq n - 2$), din familia $\mathcal{T}^k = (T_1^k, T_2^k, \dots, T_{n-k}^k)$ de $n - k$ arbori disjuncți astfel încât $V = \bigcup_{i=1}^{n-k} V(T_i^k)$ și $\bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E$, construiește \mathcal{T}^{k+1} după cum urmează:
 - alege $T_s^k \in \mathcal{T}^k$;
 - determină o muchie de cost minim $e^* = v_s v_{j^*}$ din mulțimea de muchii ale lui G cu o extremitate $v_s \in V(T_s^k)$ și cealaltă în $V \setminus V(T_s^k)$ ($v_{j^*} \in V(T_{j^*}^k)$);
 - $\mathcal{T}^{k+1} = (\mathcal{T}^k \setminus \{T_s^k, T_{j^*}^k\}) \cup T$, unde T este arborele obținut din T_s^k și $T_{j^*}^k$ prin adăugarea muchiei e^* .

Remarci

- Observăm că, dacă, la un anumit pas, nu există nicio muchie cu o extremitate în $V(T_s^k)$ și cealaltă în $V \setminus V(T_s^k)$, atunci G nu este conex și nu există vreun MST în G .
- Construcția de mai sus este sugerată în imaginea de mai jos:



- Familia \mathcal{T}^{n-1} are doar un arbore, T_1^{n-1} .

Metoda generală MST

Teorema 1

Dacă $G = (V, E)$ este un graf conex cu $V = \{1, 2, \dots, n\}$, atunci T_1^{n-1} construit de algoritmul anterior este un MST al lui G .

Demonstrație: Demonstrăm (prin inducție) că $(*) \forall k \in \{0, \dots, n-1\}$ există un arbore parțial T_k^* , MST al lui G , astfel încât

$$E(\mathcal{T}^k) = \bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E(T_k^*).$$

În particular, pentru $k = n-1$, $E(\mathcal{T}^{n-1}) = E(T_1^{n-1}) \subseteq E(T_{n-1}^*)$ implică $T_1^{n-1} = T_{n-1}^*$ și teorema este demonstrată.

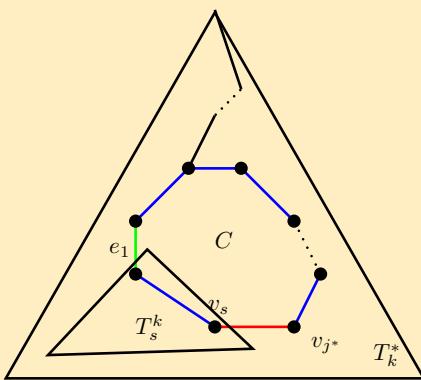
Pentru $k = 0$, avem $E(\mathcal{T}^0) = \emptyset$ și, deoarece G este conex, există un MST T_0^* ; astfel, proprietatea $(*)$ este adevărată.

Demonstrație (continuare). Dacă proprietatea (*) are loc pentru $0 \leq k \leq n - 2$, atunci există un MST al lui G , T_k^* , astfel încât $E(\mathcal{T}^k) \subseteq E(T_k^*)$. Din construcție, $E(\mathcal{T}^{k+1}) = E(\mathcal{T}^k) \cup \{e^*\}$. Dacă $e^* \in E(T_k^*)$, atunci luăm $T_{k+1}^* = T_k^*$ și proprietatea are loc și pentru $k + 1$.

Să presupunem că $e^* \notin E(T_k^*)$. Atunci, $T_k^* + e^*$ are exact un circuit C , conținând $e^* = v_s v_{j^*}$. Deoarece $v_{j^*} \notin V(T_s^k)$, urmează că există o muchie $e_1 \neq e^*$ în C cu o extremitate în $V(T_s^k)$ și cealaltă în $V \setminus V(T_s^k)$. Din modul de alegere al e^* avem $c(e^*) \leq c(e_1)$ și $e_1 \in E(T_k^*) \setminus E(\mathcal{T}^k)$. Fie $T^1 = T_k^* + e^* - e_1$; evident, $T^1 \in \mathcal{T}_G$ (fiind conex cu $n - 1$ muchii). Deoarece $e_1 \in E(T_k^*) \setminus E(\mathcal{T}^k)$, avem $E(\mathcal{T}^{k+1}) \subseteq E(T^1)$.

Metoda generală MST

Demonstrație (continuare).



Pe de altă parte, deoarece $c(e^*) \leq c(e_1)$, avem $c(T^1) = c(T_k^*) + c(e^*) - c(e_1) \leq c(T_k^*)$.

Deoarece T_k^* este un MST al lui G , urmează că $c(T^1) = c(T_k^*)$, i.e., T^1 este un MST al lui G conținând toate muchiile din $E(\mathcal{T}^{k+1})$. Luând $T_{k+1}^* = T^1$ încheiem demonstrația teoremei. \square

Remarci

- Demonstrația de mai sus rămâne adevărată și pentru funcții de cost $c : \mathcal{T}_G \rightarrow \mathbb{R}$ care satisfac: $\forall T \in \mathcal{T}_G, \forall e \in E(T), \forall e' \notin E(T)$

$$c(e') \leq c(e) \Rightarrow c(T + e' - e) \leq c(T).$$

- În metoda generală prezentată, modul de alegere a arborelui T_s^k nu este precizat în amănunt. Vom discuta două strategii foarte cunoscute.
 - Prima strategie alege T_s^k ca fiind arborele de ordin maxim din familia \mathcal{T}^k .
 - În cea de-a doua strategie T_s^k este unul dintre cei doi arbori din familia \mathcal{T}^k , legați printr-o muchie de cost minim printre toate muchiile cu extremități în arbori diferenți ai familiei.

Algoritmul lui Prim

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

 - În strategia lui Prim T_s^k este arborele de ordin maxim din familia \mathcal{T}^k .
 - Urmează că la fiecare pas $k > 0$ al metodei generale, \mathcal{T}^k are un arbore, $T_s^k = (V_s, E_s)$, cu $k + 1$ noduri și $n - k - 1$ arbori fiecare cu câte un nod.
 - **Implementarea lui Dijkstra:** Fie α și β doi vectori de dimensiune n ; elementele lui α sunt noduri din $V(G)$ iar elementele lui β sunt numere reale, cu următoarea semnificație:

$$(S) \quad \forall j \in V \setminus V_s, \beta[j] = c(\alpha[j]j) = \min_{i \in V_s, ij \in E} c(ij)$$

Algoritmul lui Prim

```

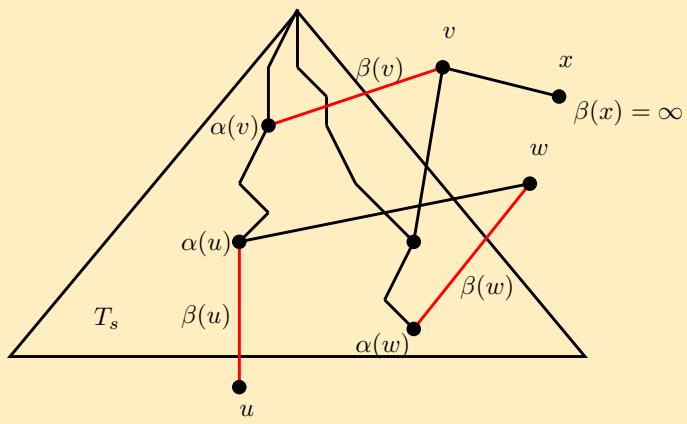
 $V_s \leftarrow \{s\}; E_s \leftarrow \emptyset; //$  pentru un  $s \in V$ .
for ( $v \in V \setminus \{s\}$ ) do
     $\alpha[v] \leftarrow s; \beta[v] \leftarrow c(sv); //$  dacă  $ij \notin E$ , atunci  $c(ij) = \infty$ .
while ( $V_s \neq V$ ) do
    find  $j^* \in V \setminus V_s$  a. î.  $\beta[j^*] = \min_{j \in V \setminus V_s} \beta[j];$ 
     $V_s \leftarrow V_s \cup \{j^*\}; E_s \leftarrow E_s \cup \{\alpha[j^*]j^*\};$ 
    for ( $j \in V \setminus V_s$ ) do
        if ( $\beta[j] > c[j^*j]$ ) then
             $\beta[j] \leftarrow c[j^*j]; \alpha[j] \leftarrow j^*;$ 

```

Algoritmul lui Prim

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

Remarci



- Se observă că (S) este satisfăcută după initializare. În bucla while, strategia metodei generale este respectată și de asemenei semnificația lui (S) este păstrată de testul din bucla for.
 - **Complexitatea timp:** $\mathcal{O}(n - 1) + \mathcal{O}(n - 2) + \dots + \mathcal{O}(1) = \mathcal{O}(n^2)$ care este bună pentru grafuri de dimensiune $\mathcal{O}(n^2)$.

- În algoritmul lui Kruskal T_s^k este unul dintre cei doi arbori din familia \mathcal{T}^k , legați printr-o muchie de cost minim printre toate muchiile cu extremități în arbori diferiți ai familiei.
- Această alegere poate fi făcută prin sortarea inițială a muchiilor crescător după cost și, după aceea, prin parcurgerea listei astfel obținute. Dacă notăm cu T arborele T_s^k , algoritmul poate fi descris astfel.

```

sort  $E = \{e_1, e_2, \dots, e_m\}$  a. î.  $c(e_1) \leq \dots \leq c(e_m)$ ;
 $T \leftarrow \emptyset$ ;  $i \leftarrow 1$ ;
while ( $i \leq m$ ) do
    if ( $\langle T \cup \{e_i\} \rangle_G$  nu are circuite) then
         $T \leftarrow T \cup \{e_i\}$ ;
     $i \leftarrow i + 1$ ;

```

Algoritmul lui Kruskal

- Sortarea poate fi făcută în $\mathcal{O}(m \log m) = \mathcal{O}(m \log n)$.
- Pentru implementarea eficientă a testului din bucla while, este necesar să reprezentăm mulțimile de noduri ale arbori, $V(T_1^k), V(T_2^k), \dots, V(T_{n-k}^k)$ (la fiecare pas k al metodei generale), și să testăm dacă muchia curentă are ambele extremități în aceeași mulțime.
- Aceste mulțimi vor fi reprezentate folosind arbori (care nu sunt, în general, subarbori ai grafului G). Fiecare astfel de arbore are o rădăcină care va fi folosită pentru a desemna mulțimea de noduri ale grafului G din acel arbore.
- Mai precis, avem o funcție $find(v)$ care determină cărei mulțimi îi aparține nodul v , adică, returnează rădăcina arborelui care reține mulțimea lui v .

- În metoda generală este necesară o reuniune (disjunctă) a multimilor de noduri a doi arbori (pentru a obține \mathcal{T}^{k+1}).
- Folosim procedura $union(u, w)$ cu următoarea semnificație: realizează reuniunea a două mulțimi de noduri, una căreia îi aparține v și una căreia îi aparține w .
- Putem rescrie bucla while a algoritmului, folosind aceste două proceduri, după cum urmează:

```

while ( $i \leq m$ ) do
    let  $e_i = vw$ ;
    if ( $find(v) \neq find(w)$ ) then
         $union(v, w)$ ;
         $T \leftarrow T \cup \{e_i\}$ ;
     $i++$ ;

```

Algoritmul lui Kruskal – Union-Find – Prima soluție

- Tabloul $root[1..n]$ cu elemente din V are semnificația: $root[v] =$ rădăcina arborelui care reține mulțimea căreia îi aparține v .
- Adaugă la pasul de inițializare (correspunzând familiei \mathcal{T}^0):

$$\text{for } (v \in V) \text{ do}$$

$$root[v] \leftarrow v;$$
- Funcția $find$ (cu complexitatea timp $\mathcal{O}(1)$):

$$\text{function } find(v : V);$$

$$\text{return } root[v];$$
- Procedura $union$ (cu complexitatea timp $\mathcal{O}(n)$):

$$\text{procedure } union(v, w : V);$$

$$\text{for } (i \in V) \text{ do}$$

$$\text{if } (root[i] = root[v]) \text{ then}$$

$$root[i] = root[w];$$

Analiza complexității timp:

- Sunt $\mathcal{O}(m)$ apeluri ale funcției *find* în timpul buclei *while*.
- În sirul de apeluri *find*, se intercalează exact $n-1$ apeluri *union* (un apel pentru fiecare muchie din *MST-ul final*).
- Astfel, time necesar buclei *while* este $\mathcal{O}(m\mathcal{O}(1)+(n-1)\mathcal{O}(n)) = \mathcal{O}(n^2)$.

Complexitatea timp a algoritmului este $\mathcal{O}(\max(m \log n, n^2))$.

Dacă G are multe muchii, $m = \mathcal{O}(n^2)$, algoritmul lui Prim este mai eficient.

Algoritmul lui Kruskal – Union-Find – A doua soluție

- Tabloul $pred[1..n]$ cu elemente din $V \cup \{0\}$ are semnificația $pred[v] =$ nodul dinaintea lui v pe drumul unic către v de la rădăcina arborelui care reține mulțimea căruia îi aparține v .
- Adaugă la pasul de inițializare (correspunzând familiei \mathcal{T}^0):

```
for ( $v \in V$ ) do
     $pred[v] \leftarrow 0$ ;
```
- Funcția *find(v)* are complexitatea în $\mathcal{O}(h(v))$, unde $h(v)$ este lungimea drumului din arbore de la v la rădăcina acestui arbore:

```
function find( $v : V$ );
     $i \leftarrow v$ ; // o variabilă locală.
    while ( $pred[i] > 0$ ) do
         $i \leftarrow pred[i]$ ;
    return  $i$ ;
```

Algoritmul lui Kruskal – Union-Find – A două soluție

[Jump to Table of contents](#)

- C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms
- Procedura **union** (de complexitate timp $\mathcal{O}(1)$) este apelată doar pentru noduri rădăcină:

```
procedure union( $root_1, root_2 : V$ )
   $pred[root_1] \leftarrow root_2;$ 
```

- Bucla **while** a algoritmului este modificată astfel:

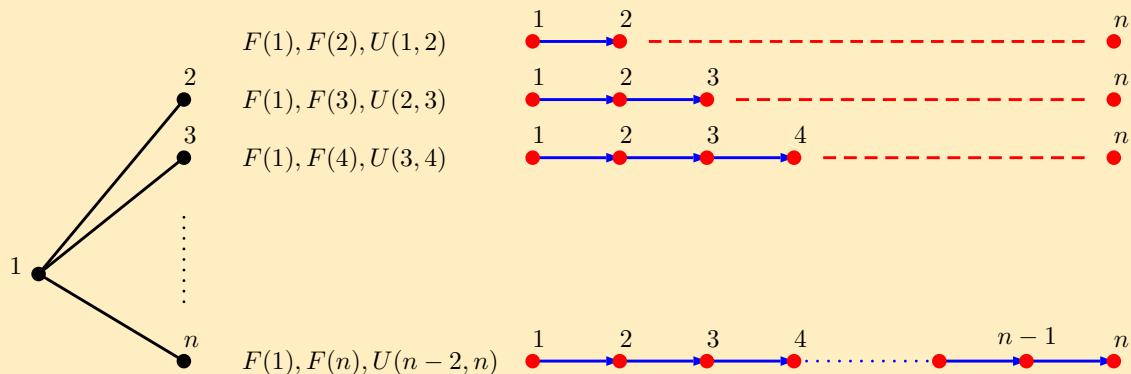
```
while ( $i \leq m$ ) do
  let  $e_i = vw; x \leftarrow find(v); y \leftarrow find(w);$ 
  if ( $x \neq y$ ) then
    union( $x, y$ );
     $T \leftarrow T \cup \{e_i\};$ 
   $i++;$ 
```

Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – A două soluție

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 executăm bucla **while** în această formă pentru graful $G = K_{1,n-1}$ cu lista sortată a muchiilor, $E = \{12, 13, \dots, 1n\}$, atunci sirul de apeluri ale celor două proceduri este (F și U abreviază **find** și **union**):



Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algo237ns * C. Croitoru - Graph Algorithms *

- Astfel, această formă a algoritmului are complexitatea timp $\Omega(n^2)$ (chiar dacă graful este rar).
- Neajunsul acestei implementări este dat de faptul că, în procedura **union**, rădăcină a noului arbore devine rădăcina aceluia arbore care reține un număr mai mic de noduri, ceea ce implică o mărire a lui $h(v)$ la $\mathcal{O}(n)$ în timpul algoritmului.
- Putem evita acest neajuns ținând în rădăcina fiecărui arbore cardinalul mulțimii pe care arborele o reține. Mai precis, semnificația $pred[v]$, unde v este o rădăcină, este:

$pred[v] < 0 \Leftrightarrow v$ este rădăcina unui arbore

care reține o mulțime cu $-pred[v]$ noduri

Algoritmul lui Kruskal – Union-Find – A doua soluție

- Pasul de inițializare

```
for ( $v \in V$ ) do  
     $pred[v] \leftarrow -1$ ;
```

- Procedura **union** are complexitatea $\mathcal{O}(1)$ pentru a întreține noua semnificație:

```
procedure union( $root_1, root_2 : V$ )  
     $t \leftarrow pred[root_1] + pred[root_2]$ ;  
    if ( $-pred[root_1] \geq -pred[root_2]$ ) then  
         $pred[root_2] \leftarrow root_1$ ;  $pred[root_1] \leftarrow t$ ;  
    else  
         $pred[root_1] \leftarrow root_2$ ;  $pred[root_2] \leftarrow t$ ;
```

Afirmăție. Cu această implementare a procedurilor `find` și `union` algoritmul are următorul invariant:

$$(*) \forall v \in V, -\text{pred}[\text{find}(v)] \geq 2^{h(v)}$$

Cu alte cuvinte, numărul de noduri din arborele căruia îi aparține v este cel puțin 2 la puterea "distanța de la v la rădăcină".

Demonstrația afirmației. După pasul de inițializare avem $h(v) = 0$, $\text{find}(v) = v$, și $-\text{pred}[v] = 1$, $\forall v \in V$, deci $(*)$ are loc cu egalitate.

Să presupunem că $(*)$ are loc înaintea unei iterării din bucla `while`. Sunt posibile două cazuri:

- În această iterăție `while` nu este apelată `union`. Tabloul `pred` nu este actualizat, deci $(*)$ are loc și după această iterăție.

Algoritmul lui Kruskal – Union-Find – A doua soluție

- În această iterăție `while` avem un apel al procedurii `union`. Fie `union(x, y)` acest apel, și să presupunem că în procedura `union` se execută atribuirea $\text{pred}[y] \leftarrow x$. Aceasta înseamnă că înaintea aceastei iterării avem $-\text{pred}[x] \geq -\text{pred}[y]$.

Nodurile v pentru care $h(v)$ se modifică în această iterăție sunt aceleia pentru care, înaintea iterăției aveam $\text{find}(v) = y$ și $-\text{pred}[y] \geq 2^{h(v)}$.

După iterăția `while`, avem $h'(v) = h(v) + 1$ și $\text{find}'(v) = x$. Astfel, trebuie să verificăm că $-\text{pred}'[x] \geq 2^{h'(v)}$. Într-adevăr, $-\text{pred}'[x] = -\text{pred}[x] - \text{pred}[y] \geq 2 \cdot (-\text{pred}[y]) \geq 2 \cdot 2^{h(v)} = 2^{h(v)+1} = 2^{h'(v)}$.

Urmează că $(*)$ este un invariant al algoritmului. \square

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

Aplicând logaritmul în (*) obținem

$$h(v) \leq \log(-\text{pred}[\text{find}(v)]) \leq \log n, \forall v \in V.$$

Complexitatea timp a buclei **while** este

$$\mathcal{O}(n - 1 + 2m \log n) = \mathcal{O}(m \log n).$$

Astfel, această a doua implementare a procedurilor **union-find** dă o complexitate timp a algoritmului Kruskal de $\mathcal{O}(m \log n)$.

* 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 *

Algoritmul lui Kruskal – Union-Find – A treia soluție

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

Complexitatea timp a buclei **while** din soluția de mai sus este datată sirului de apeluri **find**. Tarjan (1976) a observat că un apel cu $h(v) > 1$ poate să "colapseze" drumul din arbore de la v la rădăcină, fără a modifica timpul $\mathcal{O}(h(v))$, făcând $h(x) = 1$ pentru toate nodurile x de pe drum. În acest fel, viitoarele apeluri **find** pentru aceste noduri vor lua mai puțin timp. Mai precis, funcția **find** devine:

```
function find(v : V); // i, j, aux sunt variabile locale.  
    i ← v;  
    while (pred[i] > 0) do  
        i ← pred[i];  
    j ← v;  
    while (pred[j] > 0) do  
        aux ← pred[j]; pred[j] ← i; j ← aux;  
    return i;
```

Dacă $A : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$ este funcția lui Ackermann dată prin:

$$(1) \quad A(m, n) = \begin{cases} n + 1, & \text{dacă } m = 0 \\ A(m - 1, 1), & \text{dacă } m > 0 \text{ și } n = 0 \\ A(m - 1, A(m, n - 1)), & \text{dacă } m > 0 \text{ și } n > 0 \end{cases}$$

și dacă notăm, $\forall m \geq n > 0$,

$$\alpha(m, n) = \min \{z : A(z, 4\lceil m/n \rceil) \geq \log n, z \geq 1\}$$

obținem că complexitatea timp a buclei while folosind union din cea de-a doua soluție și find de mai sus, devine $\mathcal{O}(m \cdot \alpha(m, n))$.

Să notăm că $\alpha(m, n)$ este o funcție care crește foarte încet (pentru valori practice ale lui n , $\alpha(m, n) \leq 3$); astfel cea de-a treia soluție este practic o implementare liniară ($\mathcal{O}(m)$) a algoritmului lui Kruskal.

Cuplaje

Fie $G = (V, E)$ un (multi)graf. Dacă $A \subseteq E$ și $v \in V$, notăm $d_A(v) = |\{e : e \in A, e \text{ incidentă cu } v\}|$, i. e., gradul lui v în subgraful generat de A , $\langle A \rangle_G$.

Definiție

Un cuplaj (multime independentă de muchii) în G este o multime de muchii $M \subseteq E$ astfel încât

$$d_M(v) \leq 1, \forall v \in V.$$

Familia tuturor cuplajelor din graful G este notată cu \mathcal{M}_G :

$$\mathcal{M}_G = \{M : M \subseteq E, M \text{ cuplaj în } G\}.$$

Se observă că \mathcal{M}_G satisface:

- (i) $\emptyset \in \mathcal{M}_G$;
- (ii) $M \in \mathcal{M}_G, M' \subseteq M \Rightarrow M' \in \mathcal{M}_G$.

Fie $M \in \mathcal{M}_G$ un cuplaj.

- Un nod $v \in V$ cu $d_M(v) = 1$ este numit **saturat** de către M , iar mulțimea tuturor nodurilor lui G saturate de M este notată cu $S(M)$. Evident,

$$S(M) = \bigcup_{e \in M} e, \text{ și } |S(M)| = 2 \cdot |M|.$$

- Un nod $v \in V$ cu $d_M(v) = 0$ este numit **expus** (relativ) la M , iar mulțimea tuturor nodurilor lui G expuse relativ la M este notată cu $E(M)$. Evident că $E(M) = V \setminus S(M)$, și $|E(M)| = |V| - 2 \cdot |M|$.

Cuplaje maxime

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

Problema cuplajului maxim:

P₁ Dat un graf $G = (V, E)$, să se determine $M^* \in \mathcal{M}_G$ astfel încât

$$|M^*| = \max_{M \in \mathcal{M}_G} |M|.$$

Notăm cu $\nu(G) = \max_{M \in \mathcal{M}_G} |M|$.

Problema cuplajului maxim este strâns legată de **problema acoperirii minime cu muchii**.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algo242ns * C. Croitoru - Graph Algorithms *

Definiție

O acoperire cu muchii a lui G este o mulțime de muchii $F \subseteq E$ astfel încât

$$d_F(v) \geq 1, \forall v \in V(G).$$

Familia acoperirilor cu muchii ale grafului G este notată cu \mathcal{F}_G :

$$\mathcal{F}_G = \{F : F \subseteq E, F \text{ acoperire cu muchii a lui } G\}.$$

\mathcal{F}_G are următoarele proprietăți

- (i) $\mathcal{F}_G \neq \emptyset \Leftrightarrow G$ nu are noduri izolate (caz în care $E \in \mathcal{F}_G$);
- (ii) $F \in \mathcal{F}_G, F' \supseteq F \Rightarrow F' \in \mathcal{F}_G$.

Problemă acoperirii minime cu muchii:

P₂ dat un graf $G = (V, E)$, găsiți $F^* \in \mathcal{F}_G$ astfel încât

$$|F^*| = \min_{F \in \mathcal{F}_G} |F|.$$

Cuplaje maxime – Acoperire minime cu muchii

Teorema 2

(Norman-Rabin, 1959) Fie $G = (V, E)$ un graf de ordin n , fără noduri izolate. Dacă M^* este un cuplaj de cardinal maxim în G și F^* este o acoperire minimă cu muchii a lui G , atunci

$$|M^*| + |F^*| = n.$$

Demonstrație: " \leqslant " Fie M^* un cuplaj de cardinal maxim în G ; considerăm următorul algoritm:

```

 $F \leftarrow M^*;$ 
for ( $v \in E(M^*)$ ) do
    find  $v' \in S(M^*)$  a. i.  $vv' \in E$ ;
     $F \leftarrow F \cup \{vv'\};$ 

```

Demonstrație (continuare).

Să notăm că, $\forall v \in E(M^*)$, deoarece G nu are noduri izolate, există o muchie incidentă cu v , și, cum M^* este maximal relativ la relația de incluziune, această muchie are celălalt capăt în $S(M^*)$.

Mulțimea F' de muchii construită este o acoperire cu muchii și $|F'| = |M^*| + |E(M^*)| = |M^*| + n - 2 \cdot |M^*| = n - |M^*|$. Astfel

$$(2) \quad |F^*| \leq |F'| = n - |M^*|.$$

" \geq " Fie F^* o acoperire minimă cu muchii a lui G ; considerăm următorul algoritm:

```

 $M \leftarrow F^*;$ 
for ( $\exists v \in V : d_M(v) > 1$ ) do
    find  $e \in M$  incidentă cu  $v$ ;
     $M \leftarrow M \setminus \{e\};$ 

```

Cuplaje maxime – Acoperire minime cu muchii

Demonstrație (continuare).

Evident că algoritmul construiește un cuplaj M în G . Dacă muchia e incidentă cu v (eliminată din M într-o iterare **while**) este $e = vv'$, atunci $d_M(v') = 1$ și în următoarea iterare vom avea $d_M(v') = 0$, astfel la fiecare iterare **while** este creat un nod expus relativ la cuplajul final, M (dacă ar exista o altă muchie e' , în mulțimea curentă M , incidentă cu v' , atunci deoarece $e \in F^*$, $F^* \setminus \{e\}$ ar fi o acoperire cu muchii, în contradicție cu alegerea lui F^*).

Astfel, dacă M este cuplajul construit de algoritm, avem: $|F^*| - |M| = |E(M)| = n - 2 \cdot |M|$, i. e.,

$$(3) \quad |F^*| = n - |M| \geq n - |M^*|.$$

Din (2) și (3) derivă concluzia teoremei. \square

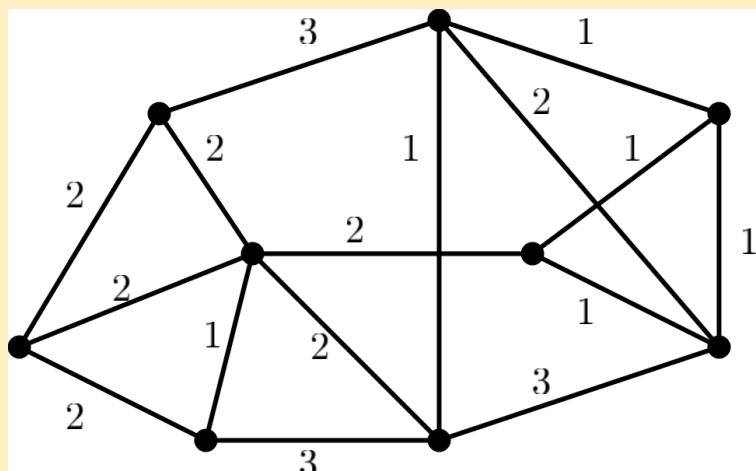
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C.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

Remarcă

Să observăm că tocmai am demonstrat că două probleme P_1 și P_2 sunt polinomial echivalente deoarece cuplajul M și acoperirea cu muchii F construite sunt soluții optime pentru cele două probleme, respectiv.

Exercitii pentru seminarul din săptămâna următoare

Exercițiu 1'. Determinați un arbore parțial de cost minim în graful de mai jos.



Exerciții pentru seminarul din săptămâna următoare

Jump to Table of contents

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

Exercițiu 1. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost pe muchiile sale. O submulțime $A \subseteq E$ este numită *tăietură* dacă există o bipartitie (S, T) a lui V astfel încât $A = \{uv \in E : u \in S, v \in T\}$ ($G \setminus A$ nu mai este conex).

- (a) Dacă în orice tăietură există o singură muchie de cost minim, atunci G conține un singur arbore parțial de cost minim.
- (b) Arătați că, dacă c este funcție injectivă, atunci G conține un singur arbore parțial de cost minim.
- (c) Reciprocele afirmațiilor de mai sus sunt adevărate?

Exercițiu 2. Fie $G = (V, E)$ un graf conex de ordin n , $c : E \rightarrow \mathbb{R}$, și \mathcal{T}_G^{\min} familia arborilor săi parțiali de cost (c) minim. Definim $H = (\mathcal{T}_G^{\min}, E(H))$ unde $T_1 T_2 \in E(H) \iff |E(T_1) \Delta E(T_2)| = 2$. Arătați că H este conex și că diametrul său este cel mult $n - 1$.

Exerciții pentru seminarul din 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

Exercițiu 3. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$. Pentru un arbore parțial $T = (V, E') \in \mathcal{T}_G$, și $v \neq w \in V$ notăm cu P_{vw}^T singurul vw -drum din T . Arătați că un arbore parțial $T^* = (V, E^*)$ este de cost minim dacă și numai dacă

$$\forall e = vw \in E \setminus E^*, \forall e' \in E(P_{vw}^{T^*}), \text{ avem } c(e) \geq c(e').$$

Exercițiu 4. Fie $G = (V, E)$ un graf 2-muchie-conex și $c : E \rightarrow \mathbb{R}$. Dacă $T = (V, E')$ este arbore parțial de cost minim al lui G și $e \in E'$, $T - e$ are exact două componente conex T'_1 și T'_2 , respectiv. Notăm cu $e_T \neq e$ o muchie de cost minim în tăietura generată de $(V(T'_1), V(T'_2))$ în $G - e$. Arătați că, dacă T^* este un arbore parțial de cost minim al lui G , și $e \in E(T^*)$, atunci $T^* - e + e_T$ este un arbore parțial de cost minim $G - e$.

Exerciții pentru seminarul din săptămâna următoare

[Jump to Table of contents](#)

Exercițiu 5. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost injectivă pe muchiile sale. Considerăm următorul algoritm

```
for ( $e \in E$ ) do
     $\gamma(e) \leftarrow r$ ; // toate muchiile sunt colorate cu roșu; în timpul execuției
    vor fi roșii, albastre sau verzi
    while (( $\exists A \subseteq E$ , tăietură cu  $\gamma(e') \neq v$ , unde  $c(e') = \min_{e \in A} c(e)$ ) sau
           ( $\exists C$ , un circuit cu  $\gamma(e') \neq a$ , unde  $c(e') = \max_{e \in C} c(e)$ )) do
        pentru o tăietură,  $A$ ,  $\gamma(e') \leftarrow v$ ;
        pentru un circuit,  $C$ ,  $\gamma(e') \leftarrow a$ ;
    return  $H = (V, \{e \in E : \gamma(e) = v\})$ ;
```

Arătați că

- (a) o muchie aparține unui arbore parțial de cost minim dacă și numai dacă este de cost minim într-o tăietură;
- (b) o muchie nu aparține nici unui arbore parțial de cost minim dacă și numai dacă este de cost maxim într-un circuit;

Exerciții pentru seminarul din săptămâna următoare

Exercițiu 5 (continuare).

- (c) algoritmul nu se oprește câtă vreme mai există muchii roșii în graf;
- (d) algoritmul se oprește pentru orice alegere a muchiilor e' , iar H este singurul arbore parțial de cost minim din G .

Exercițiu 6. Fie H un graf conex, $\emptyset \neq A \subseteq V(H)$, și $w : E(H) \rightarrow \mathbb{R}_+$. Un arbore Steiner pentru (H, A, w) este un arbore $T(H, A, w) = (V_T, E_T) \subseteq H$ cu $A \subseteq V_T$ care are costul minim printre toți arborii care conțin A , și care sunt subgrafuri ale lui H :

$$s[T(H, A, w)] = \sum_{e \in E_T} w(e) =$$

$$= \min \left\{ \sum_{e \in E_{T'}} w(e) : T' = (V_{T'}, E_{T'}) \text{ arbore în } H, A \subseteq V_{T'} \right\}$$

- (a) Arătați că un arbore Steiner poate fi determinat în timp polinomial dacă $A = V(H)$ sau $|A| = 2$.

Exercițiu 7. Considerăm o ordonare $E = \{e_1, e_2, \dots, e_m\}$ a muchiilor unui graf conex $G = (V, E)$ de ordin n . Pentru orice submulțime $A \subseteq E$ definim $\mathbf{x}^A \in GF^m$ vectorul caracteristic m -dimensional al mulțimii A : $x_i^A = 1 \Leftrightarrow e_i \in A$. GF^m este spațiul m -dimensional peste \mathbb{Z}_2 .

- Arătați că submulțimea vectorilor caracteristici corespunzători tuturor tăieturilor din G împreună cu vectorul nul este un subspațiu X al lui GF^m .
- Arătați că submulțimea vectorilor caracteristici corespunzători tuturor circuitelor din G generează un subspațiu U al lui GF^m care este ortogonal pe X .
- Arătați că $\dim(X) \geq n - 1$.
- Arătați că $\dim(U) \geq m - n + 1$.
- În final, dovediți că inegalitățile de mai sus sunt de fapt egalități.

Exerciții pentru seminarul din săptămâna următoare

Exercițiu 9. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost injectivă. Fie T^* un arbore parțial de cost minim al lui G și T_0 un arbore parțial cu cel de-al doilea cel mai mic cost în G .

- T_0 este unicul arbore parțial cu cel de-al doilea cel mai mic cost în G ?
- Arătați că $|E(T^*) \Delta E(T_0)| = 2$.
- Descrieți un algoritm pentru a determina un arbore parțial cu cel de-al doilea cel mai mic cost în G .

Exercițiu 10. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost. Adevărat sau fals? (Justificați răspunsurile!)

- (a) Orice muchie de cost minim din G este conținută într-un anume arbore parțial de cost minim din G .
 - (b) Dacă G are un circuit, C , cu o singură muchie de cost minim, atunci acea muchie este conținută în orice arbore parțial de cost minim din G .
 - (c) Dacă o muchie este conținută într-un arbore parțial de cost minim din G , atunci acea muchie este de cost minim într-o anumită tăietură a lui G .

Exercitii pentru seminarul din săptămâna următoare

Exercițiu 11. Determinați numărul de cuplaje maxime ale următorului graf:



Exercițiu 12. Doi copii se joacă pe un graf dat, G , astfel: fiecare alege alternativ un nod nou v_0, v_1, \dots așa încât, pentru orice $i > 0$, v_i este adjacent cu v_{i-1} . Jucătorul care nu mai poate alege un nod nou pierde jocul. Arătați că jucătorul care începe jocul are întotdeauna o strategie de câștig dacă și numai dacă G nu are un cuplaj perfect.

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

Exercițiu 13. Fie S o mulțime nevidă și finită, $k \in \mathbb{N}^*$, iar $\mathcal{A} = (A_i)_{1 \leq i \leq k}$ și $\mathcal{B} = (B_i)_{1 \leq i \leq k}$ două partiții ale lui S . Arătați că \mathcal{A} și \mathcal{B} admit un sistem comun de reprezentanți, i. e., există $r_{\mathcal{A}}, r_{\mathcal{B}} : \{1, 2, \dots, k\} \rightarrow S$ așa încât pentru orice $1 \leq i \leq k$, $r_{\mathcal{A}}(i) \in A_i$ și $r_{\mathcal{B}}(i) \in B_i$, iar cele două funcții au aceeași imagine.

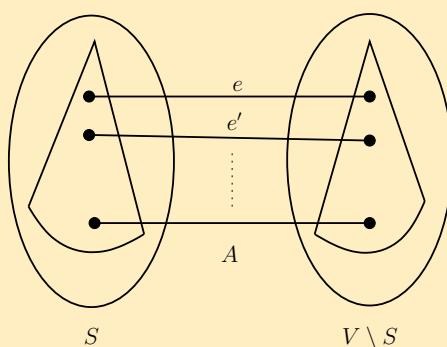
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

Exercițiu 1. Soluție.

- (a) Ștergând o muchie e dintr-un arbore parțial T al lui G , obținem exact două componente conexe (subarbori ai lui T - **de ce?**): unul cu mulțimea nodurilor S iar celălalt cu $V \setminus S$; e aparține tăieturii generate de bipartiția $(S, V \setminus S)$:



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

- Dacă T este un arbore parțial de cost minim, atunci e este muchia de cost minim în tăietură corespunzătoare de mai sus (**de ce?**).
- Presupunem prin reducere la absurd că există doi arbori parțiali de cost minim $T_1 = (V, E_1) \neq T_2 = (V, E_2)$. Fie $e_1 \in E_1 \setminus E_2$.
- $T_1 - e_1$ are două componente conexe cu multimile de noduri S și $V \setminus S$.
- În tăietura A generată de $(S, V \setminus S)$, e_1 este muchia de cost minim.
- Pe de altă parte $T_2 + e_1$ conține doar un circuit C (**de ce?**).
- C intersectează tăietură în cel puțin o altă muchie (**de ce?**), să zicem $e_2 \in E_2 \setminus E_1$.
- $T_2 + e_1 - e_2$ este un arbore parțial (**de ce?**) cu $c(T_2 + e_1 - e_2) < c(T_2)$ - contradicție.

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

Exerciții rezolvate (partial)

(b) and (c) are consequences of (a). **De ce?**

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

Exercițiul 4. Soluție. Fie T_0 un arbore parțial de cost minim în $G - e$.

- $c(T_0) \leq c(T^* - e + e_{T^*})$ (**de ce?**).
- $T_0 + e$ conține exact un circuit C care intersectează tăietură generată de $(V(T'_1), V(T'_2))$ în cel puțin o altă muchie $e_0 \neq e$ (**de ce?**).
- $c(e_0) \geq c(e_{T^*})$ (**de ce?**).
- $T_0 + e - e_0 \in \mathcal{T}_G$, deci $c(T_0 + e - e_0) \geq c(T^*)$ și
$$c(T^* - e + e_{T^*}) \geq c(T_0) \geq c(T^*) - c(e) + c(e_0) \geq c(T^*) - c(e) + c(e_{T^*})$$
- Astfel, inegalitățile de mai sus devin egalități iar $(T^* - e + e_{T^*})$ trebuie să fie arbore parțial de cost minim în $G - e$.

Exercițiu 5. Soluție. Fie $T^* = (V, E^*)$ un arbore parțial de cost minim în G .

(a) " \Rightarrow " Fie $e \in E^*$; $T^* - e$ este o pădure cu doi subarbori T_1^* and T_2^* .

- În tăietură generată de $V(T_1^*)$ și $V(T_2^*)$ considerăm o muchie de cost minim e' .
- Presupunem prin reducere că $e' \neq e$, atunci $c(e) > c(e')$, dar $T = T^* - e + e' \in \mathcal{T}_G$ și $c(T) < C(T^*)$ (**de ce?**) - contradicție.

" \Leftarrow " Fie A o tăietură și $e' \in A$ cu $c(e') = \min_{e \in A} c(e)$.

- Presupunem prin reducere că $e' \notin E^*$.
- Atunci $T^* + e'$ conține un circuit C și fie $e'' \in E(C) \cap A$ o muchie diferită de e' (**de ce există o astfel de muchie?**).
- $c(e') < c(e'')$ și $T^* + e' - e'' \in \mathcal{T}_G$ cu $c(T^* + e' - e'') < c(T^*)$ (**de ce?**) - contradicție.

Exerciții rezolvate (partial)

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

(b) " \Rightarrow " Dacă $e \notin E^*$, atunci pe circuitul C din $T^* + e'$, e' este de cost maxim. **De ce?**

" \Leftarrow " Dacă e' este de cost maxim pe circuitul C iar $e' \in E^*$, atunci $T^* - e'$ este o pădure cu doi subarbori T_1^* și T_2^* .

- În tăietura generată de $V(T_1^*)$ și $V(T_2^*)$ C trebuie să conțină și o altă muchie e'' (**de ce?**).
- Cum $c(e') > c(e'')$ obținem $T = T^* - e' + e'' \in \mathcal{T}_G$ și $c(T) < c(T^*)$ - contradicție.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algo252ns * C. Croitoru - Graph Algorithms *

(c) Fie $e = uv$ o muchie roșie, definim

$$P = \{x \in V : \text{există un drum verde de la } u \text{ la } x \text{ în } G\}.$$

- Dacă $v \in P$, atunci $C = P_{uv} + e$ este un circuit pe care e este de cost maxim (de ce?) și va fi colorată în albastru de către algoritm.
 - Dacă $v \notin P$, atunci tăietura generată de P și $V \setminus P$ nu conține muchii verzi (de ce?), deci algoritmul va colora cu verde o muchie de pe el.
- (d) Muchiile nu se recolorează (de ce?), deci bucla while se termină după cel mult $|E|$ iterații când nu vor mai fi muchii roșii.
- At the end of the algorithm we have only blue and green edges, the blue ones doesn't belong to T^* , hence all the blue edges are in T^* .
 - Ca rezultat secundar: T^* este singurul arbore parțial de cost minim în G (de ce?).

Exerciții rezolvate (partial)

Exercițiul 7. Soluție.

- (a) Trebuie demonstrat că pentru orice două tăieturi A_1, A_2 și $\alpha_1, \alpha_2 \in \{0, 1\}$, avem $\alpha_1 x^{A_1} + \alpha_2 x^{A_2} = x^A$, unde A este o tăietură sau mulțimea vidă.
- Evident că $A = \emptyset \Leftrightarrow A_1 = A_2$ (de ce?).
 - Deoarece $\alpha_1, \alpha_2 \in \mathbb{Z}_2$, va fi suficient să arătăm că, dacă $A_i = \{uv \in E : u \in S_i, v \in T_i\}$, $i = \overline{1, 2}$, sunt tăieturi distințe, atunci $x^{A_1} + x^{A_2} = x^A$, unde A este o tăietură în G ((S_i, T_i) sunt bipartitii ale lui V).
 - Este ușor de văzut că $A = A_1 \Delta A_2$ (de ce?).
 - Se poate demonstra că A este tăietura $\{uv \in E : u \in S, v \in T\}$, unde $S = S_1 \Delta S_2$, $T = V \setminus S$. Cum?

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

- (b) Deoarece $U = \text{span}(\{\mathbf{x}^{E(C)} : C \text{ circuit în } G\})$, este suficient să arătăm că pentru orice circuit C și orice tăietură A , $\mathbf{x}^A \perp \mathbf{x}^{E(C)}$.
- Dacă alegem o tăietură A și un circuit C , este ușor de văzut că $|A \cap E(C)| \equiv 0 \pmod{2}$ (de ce?).
 - Astfel,

$$\langle \mathbf{x}^A, \mathbf{x}^{E(C)} \rangle \equiv \sum_{i=1}^m \mathbf{x}_i^A \mathbf{x}_i^{E(C)} \stackrel{\text{de ce?}}{=} |A \cap E(C)| \equiv 0 \pmod{2}.$$

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 * C. Croitoru - Graph Algorithms *

Exerciții rezolvate (partial)

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

- (c) Vom pune în evidență $(n - 1)$ vectori liniar independenti din X .
- Pentru orice $v \in V$, let $A_v = \{vw \in E : w \neq v\} \neq \emptyset$.
 - Dacă $V = \{v_1, v_2, \dots, v_n\}$, atunci $\mathbf{x}^{A_1}, \mathbf{x}^{A_2}, \dots, \mathbf{x}^{A_{n-1}}$ sunt independenți. Fie $\{i_1, i_2, \dots, i_k\} \subseteq \{1, 2, \dots, n - 1\}$.
 - Cum G este conex, există o muchie $e_h = v_{i_l}v_q \in E$, unde $q \notin \{i_1, i_2, \dots, i_k\}$ (de ce?).
 - Avem

$$\left(\sum_{j=1}^k \mathbf{x}^{A_{i_j}} \right)_h = 1, \text{ thus } \sum_{j=1}^k \mathbf{x}^{A_{i_j}} \neq 0 \text{ (de ce?).}$$

- O bază în X va avea cel puțin $(n - 1)$ vectori, deci $\dim(X) \geq n - 1$.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algo 254ns * C. Croitoru - Graph Algorithms *

(d) Vom pune în evidență $(m - n + 1)$ vectori liniar independenți în U .

- Fie T un arbore parțial al lui G ; pentru orice $e \in E \setminus E(T)$, $T + e$ conține exact un circuit C_e .
- Fie $\{e_{i_1}, e_{i_2}, \dots, e_{i_p}\} \subseteq E \setminus E(T)$.
- Evident că $e_{i_1} \notin E(C_{e_{i_j}})$, pentru $2 \leq j \leq p$ deci

$$\left(\sum_{j=1}^p x^{E(C_{e_{i_j}})} \right)_{i_1} = 1, \text{ thus } \sum_{j=1}^p x^{E(C_{e_{i_j}})} \neq 0 \text{ (de ce?).}$$

- Astfel $\dim(U) \geq m - n + 1$.

Exerciții rezolvate (partial)

(e) Am arătat că

$$\dim(X) + \dim(U) \geq m = \dim(GF^m)$$

- Mai stim că $U \subseteq X^\perp$ (de ce?).
- Deci $\dim(U) \leq \dim(X^\perp) = \dim(GF^m) - \dim(X) \geq \dim(U)$.
- Adică $\dim(X) + \dim(U) = m$ - de unde se obțin egalitățile dorite.

Exercițiul 9. Soluție. Dacă c este injectivă, există un singur arbore parțial de cost minim în G (vezi ex. 1).

(a) Nu. $G \equiv K_4$, $V(G) = \{x, y, z, t\}$; $c(xy) = 2$, $c(yz) = 4$, $c(zt) = 3$, $c(tx) = 1$, $c(xz) = 5$, și $c(yt) = 6 \dots$

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

- (b) Evident că $p = |E(T^*) \setminus E(T_0)| = |E(T_0) \setminus E(T^*)| = |E(T^*) \Delta E(T_0)|/2$.

 - Presupunem prin reducere la absurd că $k \geq 2$ și fie e_1 o muchie de cost minim din $E(T^*) \setminus E(T_0)$.
 - $T_0 + e_1$ conține un circuit C și există o muchie $e_0 \in E(C) \cap (E(T_0) \setminus E(T^*))$.
 - Vom arăta că $c(e_0) > c(e_1)$. Altfel $T^* + e_0$ conține un circuit C' și există o muchie $e_2 \in E(C') \cap (E(T^*) \setminus E(T_0))$.
 - $T' = T^* + e_0 - e_2 \in \mathcal{T}_G$, deci $c(T') < c(T^*)$ și $c(e_2) < c(e_0) < c(e_1)$ - contradicție (de ce?).

Exerciții rezolvate (partial)

- Astfel, $c(e_0) > c(e_1)$. Arboarele $T'' = T_0 + e_1 - e_0 \in \mathcal{T}_G$ are costul $c(T'') < c(T_0)$.
 - Urmează că T'' este arbore parțial de cost minim, dar $T'' \neq T^*$ - contradicție (de ce?).
 - (c) Aplicăm un algoritm pentru determinarea unui arbore parțial de cost minim pentru orice graf $G - e$, $\forall e \in E(T^*)$ și păstrăm arborele parțial de cost minim obținut.
 - Cât este complexitatea timp?
 - Remarcă: o parte dintre grafurile de mai sus pot să nu fie conexe.

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

Exercițiu 10. Soluție.

- (a) Adevărat, folosind algoritmul lui Kruskal (**cum?**). Adevărat (**de ce?**).
 - (b) Fals (**de ce?**).
 - (c) Adevărat (**de ce?**).

Exercitii rezolvate (partial)

Exercițiu 11. Soluție.

- Fie $\#_{max}(G_n)$ numărul de cuplaje maxime din G_n .
 - Se poate folosi inducția după n pentru a arăta că numărul de cuplaje perfecte din graful de mai sus este 2^n .

Exercițiul 13. Solutie. Fie $G = (S, T; E)$ următorul graf bipartit:

$T = \mathcal{A}$, $S = \mathcal{B}$, și $A_i, B_i \in E$ dacă $A_i \cap B_i \neq \emptyset$.

- Se folosește teorema lui Hall ([cum?](#)).

7 CURS 7: Problema cuplajului maxim. Teoremele lui Berge si Tutte. Fluxuri in retele.

Teorema lui tutte (eliminare muchii):

- se elimina toate muchiile adiacente cu nodurile ...
- se verifica cate componente conexe impare sunt
- se verifica daca numarul este mai mic decat cardinalul multimii S
- *daca da atunci* : graful admite cuplaj perfect
- *daca nu atunci* : graful *nu* admite cuplaj perfect

Drum de crestere (muchii colorate):

- trebuie sa fie 2 noduri expuse distincte (*noduri expuse la cuplaj* sunt acelea care NU fac parte din muchiile marcate)
- trebuie realizat un drum de la unul din ele la celalalt astfel incat sa alterneze (o muchie marcata, una nemarcata)
 - in cazul in care nu se poate face spunem ca sunt *drumuri alternate*, nu sunt si de crestere deoarece nu se poate incepe de la un nod expus la altul cu alternare \Rightarrow nu exista cuplaj de cardinal maxim

Teorema lui Hall (bipartit):

- *conditia* : $N_a \geq a$
- se gaseste multimea A, se scrie, se cauta vecinii, se noteaza si dupa se verifica
- trebuie gasita o submultime care sa incalce proprietatea ca $|n(a)| \geq |a|$, adica multimea vecinilor lui A sa fie mai mica decat cardinalul multimii A
- deci \Rightarrow nu exista cuplaj care sa satureze T

Algoritmica Grafurilor - Cursul 7

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

Cuprins

- 1 **Cuplaje**
 - Formularea analitică a problemei cuplajului maxim
 - Cuplaje perfecte - Teorema lui Tutte
 - Cuplaje decardinal maxim - Teorema lui Berge
 - Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp
- 2 **Fluxuri în rețele**
 - Rețele de transport
 - Flux, valoarea fluxului, problema fluxului maxim
- 3 **Exerciții pentru seminarul următor**
- 4 **Exerciții rezolvate (parțial)**

Fie $G = (V, E)$ un graf și \mathcal{M}_G familia cuplajelor sale.

Problem cuplajului maximum.

P₁ dat un graf $G = (V, E)$, determinați $M^* \in \mathcal{M}_G$ astfel încât

$$|M^*| = \max_{M \in \mathcal{M}_G} |M|.$$

Fie $B = (b_{ij})_{n \times m}$ matricea de incidentă a lui G (se consideră căte o ordonare fixată a celor n noduri al lui G și a celor m muchii ale lui G) cu

$$b_{ij} = \begin{cases} 1, & \text{dacă muchia } j \text{ este incidentă cu nodul } i \\ 0, & \text{altfel} \end{cases}$$

Dacă 1_p este vectorul p -dimensional cu toate componentele 1, atunci problema **P₁** poate fi descrisă echivalent astfel:

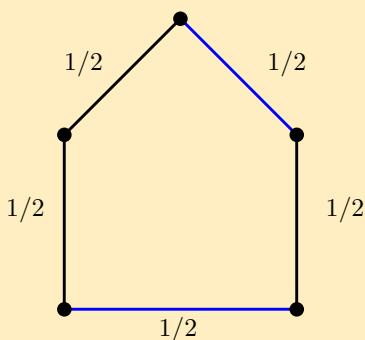
Formularea analitică a problemei cuplajului maxim

$$\mathbf{P}'_1 \max \left\{ 1_m^T x : Bx \leqslant 1_n, x \geqslant 0, x_i \in \{0, 1\}, i = \overline{1, m} \right\}.$$

P'₁ este o problemă ILP (Integer Linear Programming), care este în general NP-hard. Vom încerca să rezolvăm **P'₁**, folosind problema (relaxată) asociată LP (Linear Programming)

$$\mathbf{LP}'_1 \max \left\{ 1_m^T x : Bx \leqslant 1_n, x \geqslant 0 \right\}.$$

Soluțiile optime ale lui **LP'₁** pot avea componente ne-întregi și de asemenei valorile lor optime pot fi mai mari decât $\nu(G)$. De exemplu, dacă $G = C_{2n+1}$, atunci $\nu(G) = n$, dar soluția $x_i = 1/2, \forall 1 \leqslant i \leqslant 2n + 1$ este optimă pentru problema corespunzătoare **LP'₁** cu valoarea optimă $n + 1/2 > n$.



Urmează că dacă graful G are circuite impare, problema P_1 nu poate fi rezolvată folosind LP'_1 . Mai precis,

Teorema 1

(Balinski, 1971) Punctele extreme ale politopului $Bx \leq 1_n$, $x \geq 0$, $x \in \mathbb{R}^m$, au componente din $\{0, 1/2, 1\}$. Componentele $1/2$ apar dacă și numai dacă G are circuite impare.

Formularea analitică a problemei cuplajului maxim

- Astfel, problema P_1 este ușoară dacă graful este bipartit: se rezolvă problema LP'_1 și soluția (întreagă) optimă găsită este o soluție optimă și pentru P_1 .
- Adaptări combinatoriale ale algoritmului simplex pentru rezolvarea problemei de programare liniară au condus la aşa numita “metodă ungară” pentru rezolvarea P_1 în grafuri bipartite.
- Vom discuta o soluție mai rapidă găsită de Hopcroft și Karp (1973).
- Cu toate acestea, teorema de dualitate din programarea liniară și integralitatea soluției optime pot fi folosite pentru a obține și explica rezultatele (deja dovedite) despre cuplaje în grafuri bipartite:

Teorema 2

(Hall, 1935) Fie $G = (S, T; E)$ un graf bipartit. Există un cuplaj în G care saturează toate nodurile din S dacă și numai dacă

$$|N_G(A)| \geq |A|, \forall A \subseteq S.$$

Teorema 3

(Konig, 1930) Fie $G = (S, T; E)$ un graf bipartit. Cardinalul maxim al unui cuplaj în G este egal cu cardinalul minim al unei acoperiri cu noduri a lui G , $\nu(G) = n - \alpha(G)$.

Cuplaje perfecte - Teorema lui Tutte

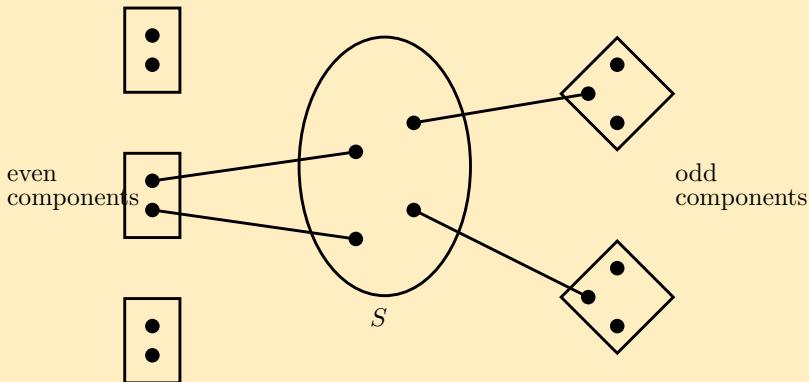
Fie G be un graf. Evident, $|M| \leq |G|/2$, $\forall M \in \mathcal{M}_G$. Un **cuplaj perfect** (sau **1-factor**) în G este un cuplaj M cu proprietatea $|M| = |G|/2$ (i.e., $S(M) = V(G)$).

O componentă conexă a grafului G este pară (impară) dacă numărul nodurilor sale este par (impar). Notăm cu $q(G)$ numărul componentelor conexe impare ale lui G .

Remarcă

Fie G un graf care are un cuplaj perfect. Evident, fiecare componentă conexă a lui G este pară. Astfel, $q(G) = 0$. Mai mult, dacă $S \subseteq V(G)$ atunci pentru fiecare componentă conexă impară din graful $G - S$ trebuie să existe o muchie în cuplajul perfect al grafului cu o extremitate în această componentă conexă și cealaltă în S . Deoarece extremitățile multiiilor diferite sunt distințte, urmează că $|S| \geq q(G - S)$.

Pentru $S = \emptyset$ în (T), obținem $q(G - \emptyset) \leq 0$, deci $q(G) = 0$.

**Teorema 4**

(Tutte, 1947) Un graf G admite cuplaj perfect dacă și numai dacă

$$(T) \quad q(G - S) \leq |S|, \forall S \subseteq V(G).$$

Cuplaje perfecte - Teorema lui Tutte

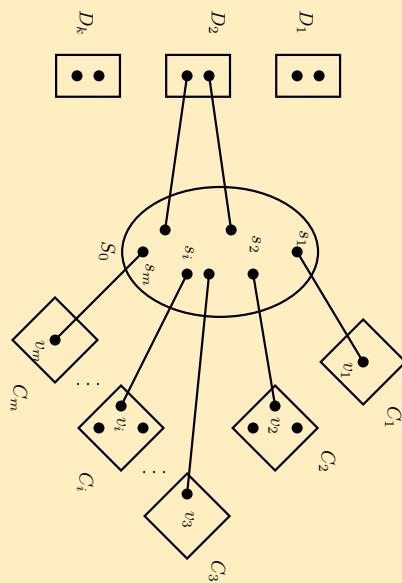
Demonstrație. Necesitatea condiției (T) a fost dovedită în discuția de mai sus. Demonstrăm prin inducție după $n = |G|$ că dacă un graf $G = (V, E)$ satisfacă (T), atunci G are un cuplaj perfect.

Pentru $n = 1, 2$ teorema are loc evident. În pasul inductiv, fie G un graf cu $n \geq 3$ noduri care satisfacă (T) și să presupunem că orice graf G' cu $|G'| < n$ și care satisfacă (T) are un cuplaj perfect.

Fie $S_0 \subseteq V(G)$ astfel încât $q(G - S_0) = |S_0|$ și maximală cu proprietatea că avem egalitate în (T) (i. e., pentru orice supramulțime S a lui S_0 avem $q(G - S) < |S|$).

Observăm că familia de submulțimi ale lui $V(G)$ pentru care (T) este satisfăcută cu egalitate este nevidă, și, deci, există un S_0 (pentru fiecare nod v_0 care nu este punct de articulație în componenta lui din G , avem $q(G - v_0) = 1 = |\{v_0\}|$, deoarece fiecare componentă conexă este pară și în fiecare componentă conexă cu cel puțin un nod, există un astfel de nod v_0).

Fie $m = |S_0| > 0$, C_1, C_2, \dots, C_m componente impare ale lui $G - S_0$ și D_1, D_2, \dots, D_k componente pare ale lui $G - S_0$ ($k \geq 0$):



Vom construi un cuplaj perfect compus din

Cuplaje perfecte - Teorema lui Tutte

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

- a) câte un cuplaj perfect în fiecare componentă conexă pară D_i ;
 - b) un cuplaj cu m muchii, $\{e_1, \dots, e_m\}$, muchia e_i având un capăt $s_i \in S_0$ și celălalt $v_i \in C_i$ ($i = 1, \dots, m$);
 - c) cuplaj perfect în fiecare subgraph $C_i - v_i$ ($i = 1, \dots, m$).
- a) **Pentru orice $1 \leq i \leq k$ graful $[D_i]_G$ admite cuplaj perfect.** Într-adevăr, deoarece $m > 0$, urmărează că $|D_i| < n$ și din ipoteza inducțivă este suficient să arătăm că $G' = [D_i]_G$ satisfacă (T).
Fie $S' \subseteq D_i$. Dacă $q(G' - S') > |S'|$, atunci obținem următoarea contradicție:

$$q(G - (S_0 \cup S')) = q(G - S_0) + q(G' - S') = |S_0| + q(G' - S') > |S_0 \cup S'|.$$

Așa că, $q(G' - S') \leq |S'|$, $\forall S' \subseteq D_i$, i.e., G' satisfacă (T).

b) Fie $H = (S_0, \{C_1, \dots, C_m\}; E')$ graful bipartit cu o clasă a bipartiției S_0 , cealaltă clasă mulțimea componentelor conexe impare ale lui $G - S_0$, și cu muchiile de forma $\{s, C_i\}$, unde $s \in S_0$ astfel că există $v \in C_i$ cu $sv \in E(G)$.

H are un cuplaj perfect. Într-adevăr, arătăm că H satisface condiția din teorema lui Hall pentru existența unui cuplaj M_0 care saturează $\{C_1, \dots, C_m\}$.

Fie $A \subseteq \{C_1, \dots, C_m\}$. Atunci $B = N_H(A) \subseteq S_0$, și din construcția lui H , în graful G nu avem nicio muchie de la un nod $v \in S_0 - B$ la un nod $w \in C_i \in A$. Astfel componentele conexe impare din A rămân componente conexe impare și în $G - B$; astfel $q(G - B) \geq |A|$. Deoarece G satisface condiția lui Tutte (T), avem $|B| \geq q(G - B)$. Am obținut că $|N_H(A)| = |B| \geq |A|$.

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

Cuplaje perfecte - Teorema lui Tutte

Din teorema lui Hall H are un cuplaj M_0 , care saturează $\{C_1, \dots, C_m\}$; deoarece $|S_0| = m$, M_0 este perfect.

$$M_0 = \{s_1 v_1, s_2 v_2, \dots, s_m v_m\}, S_0 = \{s_1, \dots, s_m\}, v_i \in C_i, \forall i = \overline{1, m}.$$

c) $\forall i \in \{1, \dots, m\}$ graful $G' = [C_i - v_i]_G$ admite cuplaj perfect. Folosind ipoteza inducțivă, este suficient să dovedim că G' satisface (T).

Fie $S' \subseteq C_i - v_i$. Dacă $q(G' - S') > |S'|$, atunci, deoarece $q(G' - S') + |S'| \equiv 0 \pmod{2}$ (pentru că $|G'|$ este par), urmează că $q(G' - S') \geq |S'| + 2$.

Dacă $S'' = S_0 \cup \{v_i\} \cup S'$, avem

$$|S''| \geq q(G - S'') = q(G - S_0) - 1 + q(G' - S') = |S_0| - 1 + q(G' - S') \geq$$

$$\geq |S_0| - 1 + |S'| + 2 = |S''|,$$

i. e., $q(G - S'') = |S''|$, în contradicție cu alegerea lui S_0 (deoarece $S_0 \subsetneq S''$).

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

Astfel $\forall S' \subseteq C_i - v_i$, $q(G' - S') \leq |S'|$ și G' are cuplaj perfect (din ipoteza inducțivă).

Evident cuplajul lui G obținut reunind cuplajele de la a), b), și c) de mai sus saturează toate nodurile lui G , și demonstrația prin inducție a teoremei se încheie

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

Cuplaje de cardinal maxim - Teorema lui Berge

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 graf și $M \in \mathcal{M}_G$ un cuplaj al lui G .

Definiție

Un drum alternat în G relativ la cuplajul M este un drum

$$P : v_0, v_0 v_1, v_1, \dots, v_{k-1}, v_{k-1} v_k, v_k$$

astfel încât $\{v_{i-1} v_i, v_i v_{i+1}\} \cap M \neq \emptyset$, $\forall i = \overline{1, k-1}$.

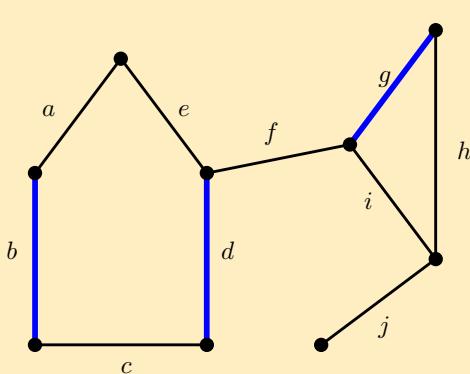
Observăm că, deoarece M este un cuplaj, dacă P este un drum alternat relativ la M , atunci dintre orice două muchii consecutive ale lui P exact una aparține lui M (muchile aparțin alternativ lui M și $E \setminus M$).

În cele ce urmează, când ne vom referi la un drum P vom înțelege multimea muchiilor sale.

Definiție

Un **drum de creștere** al lui G relativ la cuplajul M este un drum alternat care unește două noduri distincte expuse relativ la M .

Observăm că, din definiția de mai sus, urmează că dacă P este un drum de creștere relativ la M , atunci $|P \setminus M| = |P \cap M| + 1$.



a, b, c, d - alternating even path

f - alternating odd path

j - augmenting path

g, f, d - alternating odd path

a, b, c, d, e - closed alternating path

a, b, c, d, f, g, h, j - augmenting path

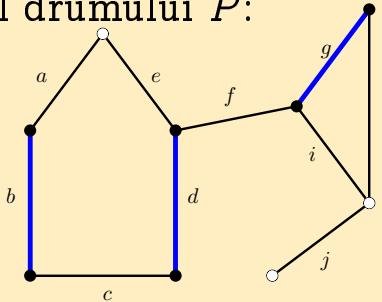
Cuplaje de cardinal maxim - Teorema lui Berge

Teorema 5

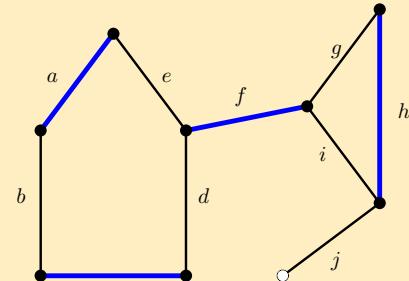
(Berge, 1959) M este un cuplaj de cardinal maxim în graful G dacă și numai dacă nu există drum de creștere în G relativ la M .

Demonstrație. " \Rightarrow " Fie M un cuplaj de cardinal maxim în G . Să presupunem că P este un drum de creștere în G relativ la M .

Atunci, $M' = M \Delta P = (P \setminus M) \cup (M \setminus P) \in \mathcal{M}_G$. Într-adevăr, M' poate fi obținut prin interschimbarea muchiilor din M cu cele din afara lui M de-a lungul drumului P :



$P = a, b, c, d, f, g, h$ - augmenting path



$M \Delta P$

Mai mult, $|M'| = |P \cap M| + 1 + |M \setminus P| = |M| + 1$, în contradicție cu alegerea lui M .

" \Leftarrow " Fie M un cuplaj în G cu proprietatea că nu există drumuri de creștere în G relativ la M .

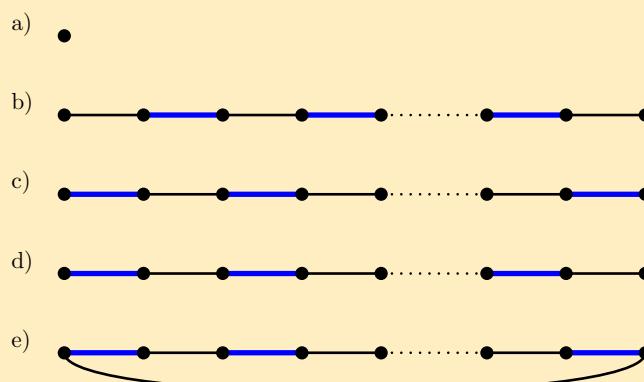
Dacă M^* este un cuplaj de cardinal maxim în G , vom arăta că $|M^*| = |M|$. Fie G' subgraful generat de $M \Delta M^*$ în G ($G' = (V, M \Delta M^*)$).

Să observăm că $d_{G'}(v) \leq 2$, $\forall v \in V$ și deci componentele conexe ale lui G' pot fi noduri izolate, drumuri de lungime cel puțin unu, sau circuite. Avem cinci posibilități (se văd mai jos; muchiile albastre sunt din M^* , muchiile negre sunt din M).

Cazul b) nu apare deoarece este un drum de creștere relativ la M^* , care este un cuplaj de cardinal maxim. Cazul c) nu apare deoarece este un drum de creștere relativ la M .

[C. Croitoru - Graph Algorithms](#) * [C. Croitoru - Graph Algorithms](#) * [C. Croitoru - Graph Algorithms](#)

Cuplaje de cardinal maxim - Teorema lui Berge



Dacă notăm cu $m_M(C)$ numărul de muchii din M din componenta conexă C a lui G' și cu $m_{M^*}(C)$ numărul de muchii din M^* din aceeași componentă conexă a lui G' , obținem că $m_M(C) = m_{M^*}(C)$. Astfel,

$$\begin{aligned}
 |M \setminus M^*| &= \sum_{C \text{ comp. a lui } G'} m_M(C) = \\
 &= \sum_{C \text{ comp. a lui } G'} m_{M^*}(C) = |M^* \setminus M|
 \end{aligned}$$

Deci $|M| = |M^*|$. \square

Obținem o strategie pentru a determina un cuplaj de cardinal maxim:

```
fie  $M$  un cuplaj în  $G$  (e. g.,  $M = \emptyset$ );
while ( $\exists P$  drum de creștere relativ la  $M$ ) do
     $M \leftarrow M \Delta P$ ;
end while
```

La fiecare iterareție **while** cardinalul lui M crește cu 1, astfel, în cel mult $n/2$ iterări obținem un cuplaj fără drumuri de creștere, adică de cardinal maxim. Neajunsul acestui algoritm este următorul: condiția din **while** trebuie implementată în timp polinomial. Acest lucru a fost făcut pentru prima oară de către **Edmonds (1965)**.

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

Lema 1

Fie $M, N \in \mathcal{M}_G$, $|M| = r$, $|N| = s$ și $s > r$. Atunci în $M \Delta N$ există cel puțin $s - r$ drumuri de creștere relativ la M disjuncte pe noduri.

Demonstrație. Fie $G' = (V, M \Delta N)$ și C_i ($i = \overline{1, p}$) componentele conexe ale lui G' . Pentru fiecare $1 \leq i \leq p$, notăm cu $\delta(C_i)$ diferența dintre numărul de muchii ale lui N din C_i și numărul de muchii ale lui M din C_i :

$$\delta(C_i) = |E(C_i) \cap N| - |E(C_i) \cap M|.$$

Se observă că, deoarece M și N sunt cuplaje, C_i sunt drumuri sau circuite. Astfel $\delta(C_i) \in \{-1, 0, 1\}$. $\delta(C_i) = 1$ dacă și numai dacă C_i este un drum de creștere relativ la M .

Demonstrație (continuare). Deoarece

$$\sum_{i=1}^p \delta(C_i) = |N \setminus M| - |M \setminus N| = s - r,$$

urmează că există cel puțin $s - r$ componente conexe ale lui G' cu $\delta(C_i) = 1$, adică, există cel puțin $s - r$ drumuri de creștere disjuncte pe noduri conținute în $M \Delta N$. \square

Lema 2

Dacă $\nu(G) = s$ și $M \in \mathcal{M}_G$ cu $|M| = r < s$, atunci există în G un drum de creștere relativ la M de lungime cel mult $2\lceil r/(s-r) \rceil + 1$.

Demonstrație. Fie $N \in \mathcal{M}_G$ cu $|N| = s = \nu(G)$.

Demonstrație (continuare). Din Lema 1, există $s - r$ drumuri de creștere disjuncte pe muchii (drumurile disjuncte pe noduri sunt și disjuncte pe muchii) conținute în $M \Delta N$. Urmează că cel puțin unul dintre ele are cel mult $\lceil r/(s-r) \rceil$ muchii din M . Lungimea acestui drum de creștere este cel mult $2\lceil r/(s-r) \rceil + 1$. \square

Definiție

Fie $M \in \mathcal{M}_G$. Un **drum de creștere minim relativ la M în G** este un drum de creștere de lungime minimă printre toate drumurile de creștere relativ la M din G .

Lema 3

Fie $M \in \mathcal{M}_G$, P un drum de creștere minim relativ la M , și P' un drum de creștere relativ la $M \Delta P$. Atunci²⁷⁰ $|P'| \geq |P| + 2|P \cap P'|$.

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

Demonstrație. Fie $N = (M \Delta P) \Delta P'$. Atunci $M \Delta N = P \Delta P'$ și $|N| = |M| + 2$. Din Lema 1, există două drumuri de creștere relativ la M disjuncte pe muchii, P_1 și P_2 , conținute în $M \Delta N$. Deoarece P este un drum de creștere minim relativ la M , avem $|P \Delta P'| \geq |P_1| + |P_2| \geq 2|P|$ și, deci, $|P| + |P'| - 2|P \cap P'| \geq 2|P|$. \square

Considerăm următorul algoritm:

```

 $M_0 \leftarrow \emptyset; i = 0;$ 
while ( $\exists$  drumuri de creștere relativ la  $M$ ) do
    fie  $P_i$  un drum de creștere minim relativ la  $M_i$ ;
     $M_{i+1} \leftarrow M_i \Delta P_i;$ 
     $i++;$ 
end while

```

Fie $P_0, P_1, \dots, P_{\nu(G)-1}$ secvența de drumuri de creștere minime construite de acest algoritm.

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

Lema 4

- a) $|P_i| \leq |P_{i+1}|$ și $|P_i| = |P_{i+1}|$ dacă și numai dacă P_i și P_{i+1} sunt disjuncte pe noduri, $\forall 1 \leq i \leq \nu(G) - 2$.

b) $\forall i < j < \nu(G) - 1$, dacă $|P_i| = |P_j|$, atunci P_i și P_j sunt disjuncte pe noduri.

Demonstrație. a) Luând $P = P_i$ și $P' = P_{i+1}$ în Lema 3, obținem $|P_{i+1}| \geq |P_i| + 2|P_i \cap P_{i+1}| \geq |P_i|$. Egalitatea are loc dacă și numai dacă P_i și P_{i+1} sunt disjuncte pe muchii, de unde rezultă că sunt disjuncte și pe noduri (fiind drumuri alternate).

b) rezultă aplicând succesiv a) \square

Teorema 6

(Hopcroft, Karp, 1973) Fie G un graf cu $\nu(G) = s$. Numărul de întregi distincți din secvența $|P_0|, |P_1|, \dots, |P_{s-1}|$ (P_i sunt drumuri de creștere minime construite de algoritmul de mai sus) nu este mai mare decât $2\lfloor\sqrt{s}\rfloor + 2$.

Demonstrație. Fie $r = \lceil s - \sqrt{s} \rceil$. Atunci $|M_r| = r$ și

$$|P_r| \leq 2\lceil r/(s-r) \rceil + 1 = 2\lceil \lceil s - \sqrt{s} \rceil / (s - \lceil s - \sqrt{s} \rceil) \rceil + 1 < 2\lfloor \sqrt{s} \rfloor + 3.$$

Astfel, pentru orice $i < r$, $|P_i|$ este unul din cele $\lfloor \sqrt{s} \rfloor + 1$ numere întregi impare nu mai mari decât $2\lfloor \sqrt{s} \rfloor + 1$.

În sub-secvența $|P_r|, \dots, |P_{s-1}|$ există cel mult $s - r \leq \lfloor \sqrt{s} \rfloor + 1$ întregi distincți. Urmează că în secvența $|P_0|, |P_1|, \dots, |P_{s-1}|$ nu există mai mult de $2\lfloor \sqrt{s} \rfloor + 2$ întregi distincți. \square

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

$$2\lceil \lceil s - \sqrt{s} \rceil / (s - \lceil s - \sqrt{s} \rceil) \rceil + 1 < 2\lfloor \sqrt{s} \rfloor + 3 \Leftrightarrow$$

$$2\lceil \lceil s - \sqrt{s} \rceil / (s - \lceil s - \sqrt{s} \rceil) \rceil < 2\lfloor \sqrt{s} \rfloor + 2 \Leftrightarrow$$

$$\lceil s - \sqrt{s} \rceil / (s - \lceil s - \sqrt{s} \rceil) \leq \lfloor \sqrt{s} \rfloor + 1 \Leftrightarrow$$

$$(s - \lfloor \sqrt{s} \rfloor) / \lfloor \sqrt{s} \rfloor \leq \lfloor \sqrt{s} \rfloor + 1 \Leftrightarrow s \leq \lfloor \sqrt{s} \rfloor^2 + 2\lfloor \sqrt{s} \rfloor$$

If $\lfloor \sqrt{s} \rfloor = k \in \mathbb{N}$, then $k \leq \sqrt{s} < k + 1$, deci $s = (\sqrt{s})^2 < k^2 + 2k + 1$ adică $s = (\sqrt{s})^2 \leq k^2 + 2k$ - care este inegalitatea de mai sus.

Dacă algoritmul de mai sus este descompus în faze astfel încât în fiecare fază se determină o familie maximală de drumuri de creștere minime disjuncte pe noduri, atunci - din Lema 4 - lungimea drumurilor de creștere minime din fază următoare nu va descrește (altfel, mulțimea drumurilor de creștere minime construite în fază curentă nu este maximală). Din Teorema 6, obținem că numărul de faze nu este mai mare de $2\lfloor \sqrt{\nu(G)} \rfloor + 2$.

Astfel avem următorul algoritm pentru determinarea unui cuplaj de cardinal maxim într-un graf dat G :

$M \leftarrow \emptyset;$

repeat

determină \mathcal{P} o familie maximală de drumuri de creștere minime relativ la M disjuncte pe noduri;

for ($P \in \mathcal{P}$) do

$$M \leftarrow M \wedge P.$$

end for

until $(\mathcal{P} = \emptyset)$

Complexitatea timp a algoritmului de mai sus este $\mathcal{O}(\sqrt{n}A)$, unde A este complexitatea timp a determinării familiei \mathcal{P} .

În cazul grafurilor bipartite, Hopcroft și Karp au arătat că aceasta poate fi obținută în $\mathcal{O}(n+m)$ și, deci, întreg algoritmul are complexitatea timp $\mathcal{O}(m\sqrt{n})$.

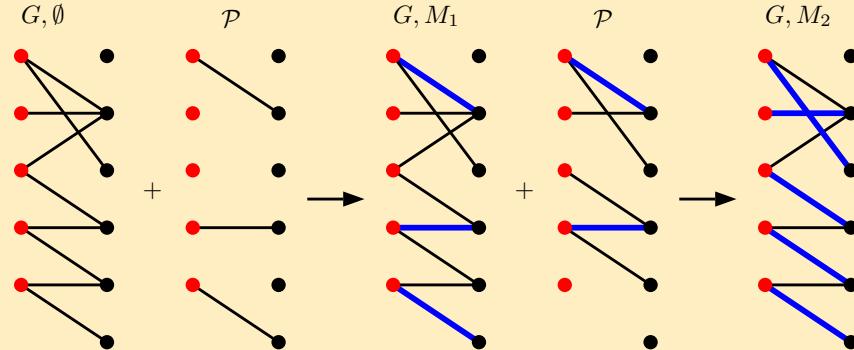
Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

Acest rezultat a fost extins la grafuri arbitrară de către Micali și Vazirani (1980) folosind o structură de date elaborată pentru a întreține etichetele asociate nodurilor pentru construcția drumurilor de creștere minime.

Să considerăm cazul grafurilor bipartite: $G = (S, T; E)$ și $M \in \mathcal{M}_G$. Pornind cu una dintre clase, de exemplu S , considerăm mulțimea extremităților inițiale ale unor drumuri de creștere $S \cap E(M)$. Din fiecare astfel de nod pornim, în paralel, construcția unor drumuri alternate într-o manieră **bfs**. Primul drum de creștere obținut oprește construcția, oferind lungimea minimă a unui drum de creștere. Familia \mathcal{P} este obținută folosind etichetele și listele de adiacență în $\mathcal{O}(n + m)$.

Detalii sunt omise; un exemplu este dat pe slide-ul următor.

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



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

Rețele de transport

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

O rețea (de transport) cu sursa s și destinația t este o tuplă $R = (G, s, t, c)$ unde:

- $G = (V, E)$ este un digraf,
- $s, t \in V$; $s \neq t$; $d_G^+(s) > 0$; $d_G^-(t) > 0$,
- $c : E \rightarrow \mathbb{R}_+$; $c(e)$ este capacitatea arcului e .

Vom presupune că $V = \{1, 2, \dots, n\}$ ($n \in \mathbb{N}^*$) și $|E| = m$. Extindem funcția c la $c : V \times V \rightarrow \mathbb{R}_+$ prin

$$c((i, j)) = \begin{cases} c(ij), & \text{dacă } ij \in E \\ 0, & \text{altfel} \end{cases}$$

și notăm $c((i, j)) = c_{ij}$.

Definiție

Un **flux** în $R = (G, s, t, c)$ este o funcție $x : V \times V \rightarrow \mathbb{R}$ a. î.

- (i) $0 \leq x_{ij} \leq c_{ij}$, $\forall (i, j) \in V \times V$,
- (ii) $\sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} = 0$, $\forall i \in V \setminus \{s, t\}$.

Remarci

- Dacă $ij \in E$ atunci x_{ij} este **flux (transportat)** de-a lungul arcului ij .
- Constraințele (i) cer ca **fluxul pe fiecare arc să fie ne-negativ și să nu depășească capacitatea**.
- Constraințele (ii), (**legea de conservare**), cer ca **suma fluxurilor de pe arcele care intră într-un nod i să fie egală cu suma fluxurilor de pe arcele care ies din i (fluxul care intră este egal cu fluxul careiese).**

Valoarea fluxului

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

- Să observăm că, din modul în care am extins funcția c la $V \times V$, constraințele (i) implică $x_{ij} = 0$ când $ij \notin E$. Astfel, un flux este de fapt o funcție definită pe E . Preferăm extensia sa pe $V \times V$ pentru a simplifica notațiile.

Fie x un flux în $R = (G, s, t, c)$. Dacă adunăm toate constraințele (ii) (pentru $i \in V \setminus \{s, t\}$) obținem

$$\begin{aligned} 0 &= \sum_{i \neq s, t} \left(\sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} \right) = \sum_{i \neq s, t} \sum_{j \neq s, t} x_{ji} - \sum_{i \neq s, t} \sum_{j \neq s, t} x_{ij} + \\ &\quad + \sum_{i \neq s, t} x_{si} + \sum_{i \neq s, t} x_{ti} - \sum_{i \neq s, t} x_{is} - \sum_{i \neq s, t} x_{it} = \end{aligned}$$

Graph Algorithms * C. Croitoru - Graph Algo 275ns * C. Croitoru - Graph Algorithms *

$$= \left(\sum_{i \in V} x_{si} - \sum_{i \in V} x_{is} \right) - \left(\sum_{i \in V} x_{it} - \sum_{i \in V} x_{ti} \right),$$

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

Definiție

Valoarea fluxului x în $R = (G, s, t, c)$ este

$$v(x) = \sum_{i \in V} x_{it} - \sum_{i \in V} x_{ti}.$$

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

În cuvinte, $v(x)$ este **fluxul net care ajunge în destinația rețelei sau, după cum am demonstrat mai sus, fluxul net care părăsește sursa rețelei**. Să observăm că în orice rețea $R = (G, s, t, c)$ există un flux: x^0 , **fluxul nul**, cu $x_{ij}^0 = 0, \forall ij \in E$ și $v(x^0) = 0$.

Problema fluxului maxim

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

Problema fluxului maxim: Dată $R = (G, s, t, c)$ o rețea, să se determine un flux de valoare maximă.

Problema fluxului maxim poate fi văzută ca o problemă LP:

$$\begin{aligned} \max \quad & v \\ \text{s.t.} \quad & \sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} = 0, \forall i \neq s, t \\ & \sum_{j \in V} x_{js} - \sum_{j \in V} x_{sj} = -v \\ & \sum_{j \in V} x_{jt} - \sum_{j \in V} x_{tj} = v \\ & 0 \leq x_{ij} \leq c_{ij}, \forall ij \in E \end{aligned}$$

Cu toate acestea, vom considera o abordare combinatorială directă care este importantă când există constrângeri de integralitate a unor variabile (de exemplu fluxul de pe arce). 276

Exercițiu 1. Fie X o mulțime finită, $X_1, \dots, X_n \subseteq X$, și $d_1, d_2, \dots, d_n \in \mathbb{N}$. Arătați că există n submulțimi disjuncte $Y_i \subseteq X_i$, $|Y_i| = d_i$, $\forall i = 1, \dots, n$ dacă și numai dacă

$$\left| \bigcup_{i \in I} X_i \right| \geq \sum_{i \in I} d_i,$$

pentru orice $I \subseteq \{1, \dots, n\}$.

Exercițiu 2. Orice graf p -regulat bipartit admite cuplaj perfect ($p \geq 1$).

Exercițiu 3. Fie $G = (S, T; E)$ un graf bipartit. Folosind teorema lui Hall demonstrați că, pentru fiecare $0 \leq k \leq |S|$, G are un cuplaj de cardinal cel puțin $|S| - k$ dacă și numai dacă $|N_G(A)| \geq |A| - k$, $\forall A \subseteq S$.

Exerciții pentru seminarul următor

Exercițiu 4. Utilizând teorema lui Tutte arătați că un graf 2-muchie conex și 3-regulat admite cuplaj perfect.

Exercițiu 5. Fie $G = (V, E)$ un graf. O submulțime $A \subseteq V$ este numită m -independentă dacă G are un cuplaj M care saturează toate nodurile din A . Arătați că, pentru orice două mulțimi m -independente A și B cu $|A| < |B|$, se poate determina un nod $b \in B \setminus A$ astfel încât $A \cup \{b\}$ este de asemenea m -independentă (\Rightarrow toate mulțimile maximal m -independente au același cardinal).

Exercițiu 6. Fie $T = (V, E)$ un arbore cu rădăcină; îi notăm cu r rădăcina și cu $parent(v)$ strămoșul direct al oricărui nod $v \neq r$. Un cuplaj M al lui T este numit *propriu* dacă orice nod nesaturat de M , $v \neq r$, are un frate w astfel încât $w parent(v) \in M$.

- (a) Arătați că orice cuplaj *propriu* este un cuplaj de cardinal maxim.
- (b) Găsiți în $\mathcal{O}(n)$ un cuplaj *propriu* pentru un arbore de ordin n .

Exercițiu 7. Fie $G = (V, E)$ un graf p -regulat bipartit ($p \geq 1$). Considerăm următorul algoritm:

```

for ( $e \in E$ ) do
     $a(e) \leftarrow 1$ ;
end for
 $E^+ \leftarrow \{e \in E : a(e) > 0\}$ ;
while ( $G^+ = (V, E^+)$  conține un circuit  $C$ ) do
    fie  $C = M_1 \cup M_2$ , unde  $M_1$  și  $M_2$  sunt cuplaje a. î.  $a(M_1) \geq a(M_2)$ ; //
    pentru orice  $F \subseteq E$ ,  $a(F) = \sum_{e \in F} a(e)$ ;
    for ( $e \in E(C)$ ) do
        if ( $e \in M_1$ ) then
             $a(e) ++$ ;
        else
             $a(e) --$ ;
        end if
    end for
     $E^+ \leftarrow \{e \in E : a(e) > 0\}$ ;
end while
return  $E^+$ ;
```

Exerciții pentru seminarul următor

Exercițiu 7. (cont.) Fie $f(E^+) = \sum_{e \in E^+} a^2(e)$. Arătați că

- (a) după fiecare iterareție **while** $f(E^+)$ este un număr întreg care crește cu cel puțin $|C|$ față de valoarea anterioară;
- (b) după fiecare iterareție **while**, $\sum_{uv \in E^+} a(uv) = p$, $\forall u \in V$;
- (c) cât timp există muchii e cu $0 < a(e) < p$, algoritmul continuă; la final $a(e) = p$, $\forall e \in E^+$ și în E^+ se găsesc muchiile unui cuplaj perfect al lui G ;
- (d) numărul de iterareții **while** este finit, la final $f(E^+) = np^2/2 = pm$, iar suma lungimilor tuturor circuitelor procesate este cel mult pm ;
- (e) toate circuitele pot fi găsite în complexitatea timp $\mathcal{O}(\sum_C |C|)$ folosind parcurgeri **dfs**;
- (f) complexitatea timp a algoritmului în ansamblu este $\mathcal{O}(pm)$.

Exercițiu 8. Fie $G = (S, T; E)$ un graf bipartit nenul. Arătați că următoarele afirmații sunt echivalente:

- $G - \{x, y\}$ are un cuplaj perfect, $\forall x \in S, \forall y \in T$.
- G este conex și orice muchie a lui G aparține unui cuplaj perfect.
- $|S| = |T|$ și $\emptyset \neq A \subsetneq S, |N_G(A)| > |A|$.

Exercițiu 9. Fie $G = (V, E)$ un graf conex care are un cuplaj perfect. Descrieți (și demonstrați corectitudinea) unui algoritm de complexitate timp $\mathcal{O}(|V| + |E|)$ care să construiască un arbore parțial T al lui G astfel ca $V(T)$ să admită o bipartitie în două mulțimi stabile de cardinal maxim ale lui T .

Exerciții rezolvate (parțial)

Exercițiu 1. Soluție.

- Fie $G = (S, X, E)$ următorul graf bipartit: S este compusă din câte d_i copii ale submulțimilor X_i , $\forall i = \overline{1, n}$; pentru fiecare $x \in X$ și pentru fiecare $X'_i \in S$ (o copie a lui X_i) adăugăm xX'_i la E dacă și numai dacă $x \in X_i$.
- Există mulțimile cerute $(Y_i)_{1 \leqslant i \leqslant n}$ dacă și numai dacă există un cuplaj care saturează S (**puteți explica de ce?**), ceea ce, din teorema lui Hall, este echivalent cu

$$\forall A \subseteq S, |A| \leqslant |N_G(A)| \iff$$

$$\iff \forall (d'_i)_{1 \leqslant i \leqslant n} \subseteq \mathbb{N}, d'_i \leqslant d_i, \forall i = \overline{1, n} \text{ and}$$

$$\sum_{i=1}^n d'_i \leqslant \left| \bigcup_{i=\overline{1, n}, d'_i \neq 0} X_i \right|.$$

279

Luând $I = \{i : d'_i \neq 0\}$ obținem inegalitatea dorită.

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

Exercițiu 2. Soluție. Fie $G = (S, T; E)$ un graf bipartit și p -regulat.

- Este ușor de arătat că $|S| = |T|$ (**cum?**).
- Astfel, va fi suficient de demonstrat că există un cuplaj care saturează S .
- Fie $A \subseteq S$; există $k|A|$ muchii între A și $N_G(A)$, deci $k|A| \leq k|N_G(A)|$ care este numărul total de muchii care pleacă din $N_G(A)$.
- Obținem $|A| \leq |N_G(A)|$ și concluzia dorită din teorema lui Hall.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

Exercițiu 3. Soluție.

- Fie $G' = (S, T \cup W; E')$ graful obținut astfel: adăugăm k noduri noi la T ($W = \{x_1, x_2, \dots, x_k\}$); toate aceste noduri vor fi adiacente cu toate nodurile din S : $E' = E \cup \{ux_i : u \in S, 1 \leq i \leq k\}$).
- Evident că G conține un cuplaj de cardinal cel puțin $|S| - k$ dacă și numai dacă G' conține un cuplaj care saturează toate nodurile din S (**de ce?**).
- În G' există un cuplaj M' , cu $|M'| = |S|$ dacă și numai dacă (**de ce?**):

$$|N_{G'}(A)| \geq |A|, \forall A \subseteq S$$

ceea ce este echivalent cu

$$|N_G(A)| + k \geq |A|, \forall A \subseteq S,$$

deoarece $N_{G'}(A) = N_G(A) \cup W$ și $|N_{G'}(A)| = |N_G(A)| + k$.

Exercițiu 4. Soluție. Fie G un graf 2-muchie conex și 3-regulat

- Vom arăta că $q(G - S) \leq |S|$, $\forall S \subseteq V$.
- G este de ordin par (de ce?); putem presupune că $S \neq \emptyset$.
- Fie H_i o componentă conexă impară a lui $G - S$; cum G este conex, există o muchie $e_1 = u_1 v_1$ cu $u_1 \in S$ și $v_1 \in V(H_i)$.
- $G - e_1$ este deasemenea conex (de ce?), deci există și o altă muchie $e_2 = u_2 v_2 \neq e_1$, cu $u_2 \in S$, $v_2 \in V(H_i)$.
- Să presupunem prin reducere la absurd că e_1 and e_2 sunt singurele muchii care leagă S de H_i , atunci:

$$2|E({}_i H)| = \sum_{v \in V(H_i)} d_{H_i}(v) \stackrel{\text{de ce?}}{=} 3|H_i| - 2 \equiv 1 \pmod{2}, \text{ contradicție.}$$

- Astfel, fiecare componentă conexă impară a lui $G - S$ este legată prin cel puțin trei muchii de S :

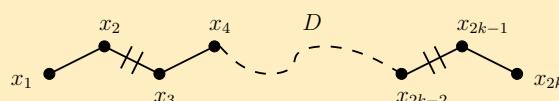
$$3|S| = \sum_{x \in S} d_G(x) \geq 3q(G - S).$$

Exerciții rezolvate (partial)

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

Exercițiu 6. Soluție.

- (a) Fie M un cuplaj propriu și să presupunem prin reducere la absurd că $|M| < \nu(T)$.
- Din teorema lui Berge T conține un M -drum de creștere D .
 - Un astfel de drum are lungime impară ≥ 3 (de ce?).
 - Presupunem că $V(D) = \{x_1, x_2, \dots, x_{2k}\}$, $x_1, x_{2k} \in E(M)$ și $x_2 = \text{parent}(x_1)$, $x_3 = \text{parent}(x_2)$.
 - Cum $x_1 \in S(M)$, există y un fată al lui x_1 cu $x_2 y \in M$ - contradicție (de ce?).



- (b) Folosim o parcurgere în adâncime (de ce?) a arborelui pentru a construi un cuplaj propriu:

```
 $M \leftarrow \emptyset;$ 
 $E(M) \leftarrow V(T);$ 
 $\text{ProperMatching}(r);$ 
```

- unde *ProperMatching* este descrisă recursiv mai jos:

```
 $\text{ProperMatching}(v); \{$ 
 $\quad \text{for}(x \in A(v)) \{$ 
 $\quad \quad \text{ProperMatching}(x);$ 
 $\quad \quad \text{if } (v, x \in E(M)) \{$ 
 $\quad \quad \quad M \leftarrow M \cup \{vx\};$ 
 $\quad \quad \quad E(M) \leftarrow E(M) \setminus \{x, v\};$ 
 $\quad \quad \}$ 
 $\quad \}$ 
 $\}$ 
```

Exerciții rezolvate (partial)

Exercițiul 7. Soluție.

- (a) Inițial $f(E^+) = m \in \mathbb{Z}_+$; după o iterare while $f(E^+)$ crește cu:

$$\begin{aligned} \sum_{e \in M_1} [(a(e) + 1)^2 - a^2(e)] + \sum_{e \in M_2} [(a(e) - 1)^2 - a^2(e)] &= \\ \sum_{e \in M_1} [2a(e) + 1] + \sum_{e \in M_2} [-2a(e) + 1] &= \\ = 2[a(M_1) - a(M_2)] + |C| &\geq |C|, \quad a(M_i) \in \mathbb{Z}. \end{aligned}$$

- (b) Evident. De ce?

- (c) Având o muchie e_1 cu $0 < a(e_1) < p$ putem aplica următoarea procedură: presupunem că $e_1 = u_1 u_2 \in E^+$; există o muchie $e_2 = u_2 u_3 \in E^+$ cu $0 < a(e_2) < p$ (de ce?), și tot aşa până când atingem un nod, u_i , deja vizitat și se închide un circuit numai cu muchii din E^+ .

Exerciții rezolvate (partial)

[Jump to Table of contents](#)

- (d) La sfârșitul execuției algoritmului $|E^+| = n/2$ și $f(E^+) = np^2/2 = pm$ (de ce?). În fiecare pas $f(E^+)$ crește cu cel puțin $|C|$ deci creșterea totală a lui $f(E^+)$ este $(pm - m)$ iar suma lungimilor tuturor circuitelor este cel mult pm .

(e)

```
for v ∈ V do
    parent(u) ← -1;
end for
fie e = uv ∈ E+ cu 0 < a(e) < p; parent(u) ← 0; parent(v) ← u;
crează o stivă S conținând v;
while S ≠ ∅ do
    x ← top(S);
    if ((y ← next[A(u)]) ≠ NULL) then
        if parent(y) < 0 then
            parent(v) ← u; push(S, v);
        else
            return y;
        end if
    else
        delete(S, x);
    end if
end while
```

- Folosind y returnat și tabloul $parent(\cdot)$ putem determina circuitul dorit. Nu există nicio muchie $e \in E^+$ cu $0 < a(e) < p$, deci $G^+ = (V, E^+)$ este aciclic (de ce?).

Exerciții rezolvate (partial)

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. Soluție.

- (i) \Rightarrow (ii) Dacă G nu este conex fie G' una dintre componentele sale conexe, $0 < |V(G') \cap S| \leq |V(G') \cap T|$. Luăm $u \in V(G') \cap S$, $v \in T \setminus V(G')$ și M un cuplaj perfect în $G - \{u, v\}$; $M \cap E(G')$ nu poate satura toate nodurile din G' (de ce?).
- (ii) \Rightarrow (iii) Cum G are un cuplaj perfect, $|S| = |T|$ (de ce?). Din teorema lui Hall $|N_G(A)| \geq |A|$.
- Să presupunem prin reducere la absurd că $|N_G(A)| = |A|$, pentru o mulțime $\emptyset \neq A \subsetneq S$.
- $S \cup T \neq A \cup N_G(A)$ (de ce?), deci există două noduri adacente $u \in N_G(A)$ și $v \in S \setminus A$.

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

- Fie M be a cuplaj perfect care conține uv ; M conține deasemeni $|A|$ muchii de la A la $N_G(A)$, deci v este saturat în M de o muchie diferită de uv (de ce?) - contradicție.
- (iii) \Rightarrow (i) Folosim teorema lui Hall pentru graful bipartit $G - \{x, y\}$. Pentru orice $A \subseteq S - \{x\}$

$$|N_{G - \{x, y\}}(A)| \geq |N_G(A)| - 1 \geq |A|.$$

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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. Soluție.

- Fie M un cuplaj perfect în $G = (V, E)$.
- Mai întâi reordonăm listele de adiacență astfel ca muchiile din M să apară la început. Fie următoarea procedură **dfs**

```
dfs(v, i)
visited[v] ← i;
for (u ∈ A(v)) do
    if (visited[u] = -1) then
        E' ← E' ∪ {uv}; dfs(u, 1 - i);
    end if
end for
```

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

- Algoritmul principal este:

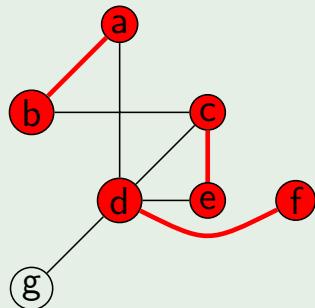
```
for ( $v \in V$ ) do  
    visited[ $v$ ]  $\leftarrow -1$ ;  
end for  
let  $v_0 \in V$ ;  
dfs( $v_0$ , 0);
```

- Definim $A = \{v \in V : \text{visited}[v] = 0\}$ și $B = \{v \in V : \text{visited}[v] = 1\}$; A și B sunt mulțimi stabile (în $T = (V, E')$) de aceeași cardinal (de ce?).
- (Cu alte cuvinte, orice drum de la rădăcină la frunze este alternat: unul dintre ele începe și se termină cu câte o muchie din M iar celelalte încep cu câte o muchie din afara lui M și se încheie cu o muchie din M .)

Se consideră dat un graf simplu neorientat $G = (V, E)$.

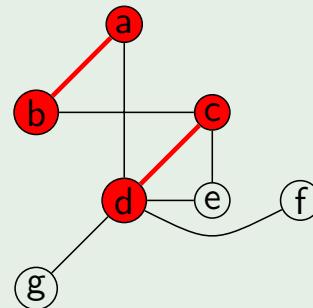
- Un **cuplaj** în G este o mulțime de muchii M în care nici o pereche de muchii nu are un nod comun. Nodurile adiacente la muchiile din M se numesc noduri **saturate de M** (sau **M -saturate**). Celelalte noduri se numesc **M -nesaturate**.

Exemplu



$$M_1 = \{(a,b), (c,e), (d,f)\}$$

Noduri M_1 -saturate: a, b, c, e, d, f



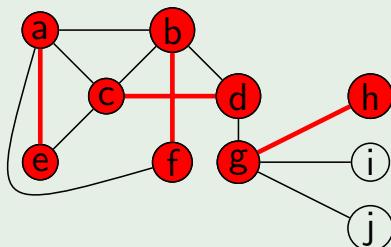
$$M_2 = \{(a,b), (c,d)\}$$

Noduri M_2 -saturate: a, b, c, d

Definiții (2)

- Un **cuplaj perfect al lui G** este un cuplaj care saturează toate nodurile lui G .
- Un **cuplaj maxim al lui G** este un cuplaj care are cel mai mare număr posibil de muchii.
- Un **cuplaj maximal al lui G** este un cuplaj care nu poate fi lărgit prin adăugarea unei muchii.

Exemplu (Cuplaje maxime și maximale)



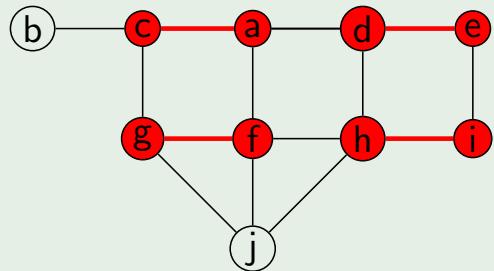
Cuplaje maxime?

$$M_1 = \{(a,e), (b,f), (c,d), (g,h)\}$$

Definiție (Cale M -alternantă, M -cale de creștere)

Dacă se dă un graf G și un cuplaj M , o **cale M -alternantă** este o cale în G în care toate muchiile alternează între M -muchii și non- M -muchii. O **M -cale de creștere** este o cale M -alternantă care are ambele capete M -nesaturate.

Exemplu



M -cale alternantă: (c,a,d,e,i)

M -cale de creștere: (j,g,f,a,c,b)

Teorema lui Berge

Teoremă

Un cuplaj M al unui graf G este maxim dacă și numai dacă G nu conține M -căi de creștere.

DEMONSTRAȚIA LUI “ \Rightarrow ”. Presupunem că M este un cuplaj maxim. Demonstrăm prin contradicție că G nu are M -căi de creștere. Dacă $P : (v_1, v_2, \dots, v_k)$ ar fi o M -cale de creștere atunci, conform definiției, k ar trebui să fie par astfel încât $(v_2, v_3), (v_4, v_5), \dots, (v_{k-2}, v_{k-1})$ sunt muchii din M , iar $(v_1, v_2), (v_3, v_4), \dots, (v_{k-1}, v_k)$ nu sunt muchii din M .



În acest caz putem defini următorul cuplaj M_1 al lui G :

$$M_1 = (M \setminus \{(v_2, v_3), \dots, (v_{k-2}, v_{k-1})\}) \cup \{(v_1, v_2), \dots, (v_{k-1}, v_k)\}.$$

Dar M_1 conține o muchie mai mult decât M , ceea ce contrazice ipoteza
287 că M este maxim.

“ \Leftarrow :” Dacă M nu este maxim, există un cuplaj M' al lui G cu $|M'| > |M|$. Fie H subgraful lui G definit astfel:

- $V(H) = V(G)$
- $E(H) =$ mulțimea muchiilor ce apar exact o dată în M și M' .

$|M'| > |M| \Rightarrow H$ are mai multe muchii în M' decât în M .

Orice nod v al lui H aparține la cel mult o muchie din M și la cel mult o muchie din $M' \Rightarrow \deg_H(v) \leq 2$ pentru toți $v \in V(H)$

\Rightarrow componentele conexe ale lui H cu mai multe M' -muchii decât M -muchii sunt căi sau cicluri. Dacă este ciclu, trebuie să fie ciclu par fiindcă muchiile alternează între M -muchii și M' -muchii
 \Rightarrow singurele componente conexe ale lui H care pot conține mai multe M' -muchii decât M -muchii sunt căile.

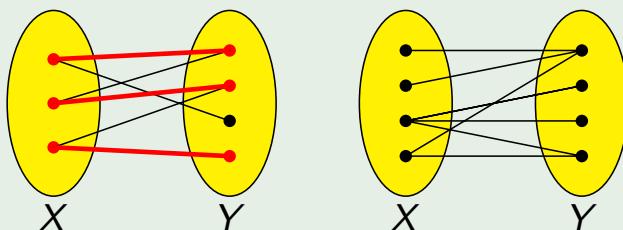
$|M'| > |M| \Rightarrow$ există o cale P în H care începe și se termină cu o muchie din $M' \Rightarrow P$ este M -cale de creștere, contradicție cu ipoteza.



“Cuplaj în”

Dacă G este graf bipartit cu mulțimile partite X și Y , spunem că X poate fi **cuplat în** Y dacă există un cuplaj al lui G care saturează nodurile din X .

Exemplu



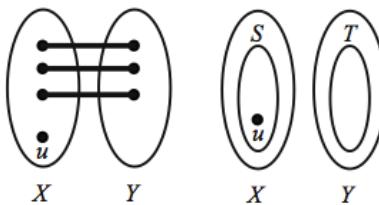
- Primul este un graf bipartit unde X poate fi cuplat în Y .
- Al doilea este un graf bipartit unde X nu poate fi cuplat în Y .
De ce se întâmplă acest lucru?

Teorema lui Hall

Fie G un graf bipartit cu mulțimile partite X și Y . X poate fi cuplat în Y dacă și numai dacă $|N(S)| \geq |S|$ pentru toate submulțimile S ale lui X .

DEMONSTRAȚIE. “ \Rightarrow :” fie $S \subseteq X$. X poate fi cuplat în $Y \Rightarrow S$ poate fi cuplat în Y , deci $|N(S)| \geq |S|$.

“ \Leftarrow :” Presupunem că ar exista un cuplaj maxim M care nu acoperă un nod $u \in X$.



Fie A mulțimea nodurilor din G ce pot fi conectate la u cu o cale M -alternantă, $S = A \cap X$, și $T = A \cap Y$.

Conform Teoremei lui Berge, toate nodurile din T sunt sature de M , iar u este singurul nod nesaturat al lui $S \Rightarrow |T| = |S| - 1$. Din Teorema lui Berge și definiția lui T rezultă că $N(S) = T$. Dar în acest caz avem $|N(S)| = |S| - 1 < |S|$, contradicție.



Cursul 11

Sistem de reprezentanți distincți (SRD)

Definiție

Dacă se dă o familie de mulțimi $X = \{S_1, \dots, S_n\}$, un **sistem de reprezentanți distincți** (sau **SRD**) pentru mulțimile din X este o mulțime de elemente distincte $\{x_1, \dots, x_n\}$ cu $x_i \in S_i$ pentru $1 \leq i \leq n$.

Exemplu

Fie $S_1 = \{2, 8\}$, $S_2 = \{8\}$, $S_3 = \{5, 7\}$, $S_4 = \{2, 4, 8\}$, $S_5 = \{2, 4\}$.

- $X_1 = \{S_1, S_2, S_3, S_4\}$ are SRD {2, 8, 7, 4}
 $S_1 = \{2, 8\}$, $S_2 = \{8\}$, $S_3 = \{5, 7\}$, $S_4 = \{2, 4, 8\}$
- $X_2 = \{S_1, S_2, S_4, S_5\}$ nu are un SRD.

Întrebare. În ce condiții are o familie finită de mulțimi un SRD?

Teoremă (Teorema lui Hall)

Fie S_1, S_2, \dots, S_k o colecție de mulțimi nevide finite. Colecția are un SRD dacă și numai dacă pentru orice $t \in \{1, \dots, k\}$, reuniunea a t astfel de mulțimi conține cel puțin t elemente.

DEMONSTRĂȚIE. Fie $Y = S_1 \cup S_2 \cup \dots \cup S_k$. Presupunem că $Y = \{a_1, \dots, a_n\}$ și considerăm graful bipartit cu mulțimile partite $X = \{S_1, \dots, S_k\}$ și Y în care există o muchie de la S_i la a_j dacă și numai dacă $a_j \in S_i$.

Conform teoremei lui Hall, X poate fi cuplat în Y dacă și numai dacă $t = |A| \leq |N(A)|$ pentru toate submulțimile $A = \{S_{i_1}, \dots, S_{i_t}\}$ ale lui X .



Cuplaje bipartite ponderate

O problemă motivantă de optimizare combinatorială

Trei muncitori: Ion, Dan și Doru sunt disponibili pentru a efectua trei lucrări: să spele baia, să măture, și să spele ferestrele. Fiecare cere un anumit preț pentru fiecare lucrat, de exemplu:

	spălat baie (SB)	măturat (M)	spălat ferestre (SF)
Ion	$w(\text{Ion, SB}) = 20$	$w(\text{Ion, M}) = 30$	$w(\text{Ion, SF}) = 30$
Dan	$w(\text{Dan, SB}) = 30$	$w(\text{Dan, M}) = 20$	$w(\text{Dan, SF}) = 20$
Doru	$w(\text{Doru, SB}) = 30$	$w(\text{Doru, M}) = 30$	$w(\text{Doru, SF}) = 20$

Vrem să alocăm câte o lucrată la fiecare muncitor, a.î. să plătim cel mai puțin posibil

↔ dacă G este graful ponderat bipartit complet dintre $S = \{\text{Ion, Dan, Doru}\}$ și $T = \{\text{SB, M, SF}\}$, cu ponderile muchiilor ca în tabelul anterior, vrem să găsim un **cuplaj perfect minim** M în G :

$$\sum_{(s,t) \in M} w(s, t) \text{ este minimă.}$$



~~PRESUPUNEM CĂ $G = K_{n,n}$ este graf bipartit complet între $S = \{x_1, \dots, x_n\}$ și $T = \{y_1, \dots, y_n\}$, a.î. fiecare $(x, y) \in E$ are greutatea $w(x, y) \geq 0$.~~

Observație (König & Egerváry)

Găsirea unui cuplaj minim perfect în G se poate reduce la problema găsirii unei acoperiri maxime în G .

- ▶ O **acoperire** a lui G este o funcție $c : S \cup T \rightarrow \mathbb{R}$, a.î.
 $c(x) + c(y) \leq w(x, y)$ pentru toți $(x, y) \in E$.
- ▶ O **acoperire maximă** a lui G este o acoperire c astfel încât

$$\sum_{x \in S} c(x) + \sum_{y \in T} c(y)$$

are valoarea maximă posibilă.

Această sumă se numește **costul** acoperirii c .



Cuplaje bipartite ponderate

Rezultate teoretice

- Pentru orice cuplaj perfect M dintre S și T și acoperire c , are loc inegalitatea

$$\sum_{x \in S} c(x) + \sum_{y \in T} c(y) \leq \sum_{(x,y) \in M} w(x, y).$$

Mai mult:

$$\sum_{x \in S} c(x) + \sum_{y \in T} c(y) = \sum_{(x,y) \in M} w(x, y)$$

dacă și numai dacă $c(x) + c(y) = w(x, y)$ pentru toate muchiile $(x, y) \in M$. În acest caz, M este un cuplaj perfect minim iar c este o acoperire maximă a lui G .

Calculează o acoperire a unui graf ponderat $K_{n,n}$ în timp polinomial $O(n^4)$:

- ▶ Inițial, $c(x) = 0$ și $c(y) = \max\{w(x,y) \mid x \in S\}$ pentru toți $x \in S$ și $y \in T$.
(OBSERVAȚIE: c este acoperire a nodurilor lui G .)
- ▶ Algoritmul efectuează un număr finit de pași de modificare a lui c , până când c devine acoperire maximă. La fiecare pas, se consideră:
 - ① **graful $G_c := \{(x,y) \mid c(x) + c(y) = w(x,y)\}$ și un cuplaj M al lui G_c .** Inițial, $M = \emptyset$.
 - ▶ **Algoritmul se oprește când M este cuplaj perfect** (are n muchii).
 - ② $R_S \subseteq S$ sunt nodurile nesaturate de M din S , iar $R_T \subseteq T$ sunt nodurile nesaturate de M din T .
 - ③ $Z :=$ mulțimea de noduri unde se poate ajunge din R_S cu o cale M -alternantă:
 - ▶ Se disting 2 cazuri: dacă $R_T \cap Z = \emptyset$ sau $R_T \cap Z \neq \emptyset$

Cursul 11

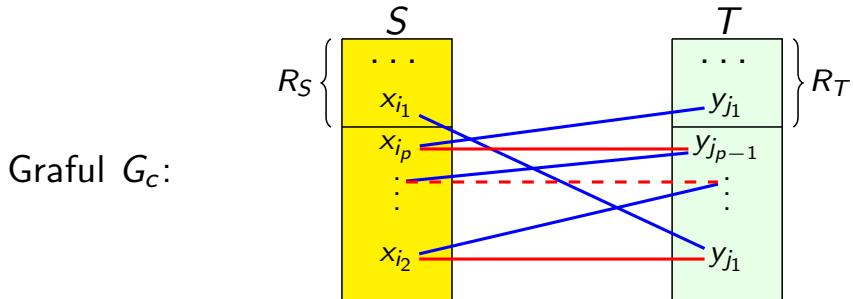
Cuplaje bipartite ponderate

Algoritmul ungar (2)

Dacă $R_T \cap Z \neq \emptyset$, fie $(x_{i_1}, y_{j_1}, x_{i_2}, \dots, x_{i_p}, y_{j_p})$ o cale M -alternantă de la $x_{i_1} \in R_S$ la $y_{j_p} \in R_T$. Înseamnă că

$(x_{i_{k+1}}, y_{j_k}) \in M$ pentru $1 \leq k < p$ și $(x_{i_k}, y_{j_k}) \notin M$ pentru $1 \leq k \leq p$. În acest caz, modificăm M să fie

$$M := (M - \{(x_{i_{k+1}}, y_{j_k}) \mid 1 \leq k < p\}) \cup \{(x_{i_k}, y_{j_k}) \mid 1 \leq k \leq p\}.$$



OBSERVAȚIE: $|M|$ CREȘTE CU 1

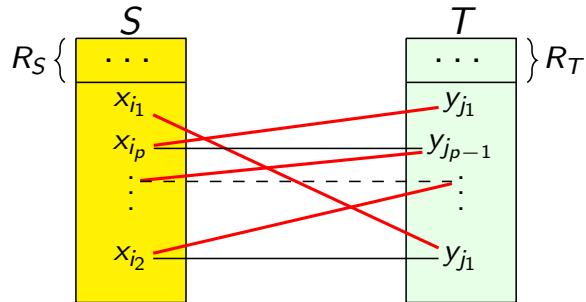
Dacă $R_T \cap Z \neq \emptyset$, fie $(x_{i_1}, y_{j_1}, x_{i_2}, \dots, x_{i_p}, y_{j_p})$ o cale M -alternantă de la $x_{i_1} \in R_S$ la $y_{j_p} \in R_T$. Înseamnă că

$(x_{i_{k+1}}, y_{j_k}) \in M$ pentru $1 \leq k < p$ și $(x_{i_k}, y_{j_k}) \notin M$ pentru

$1 \leq k \leq p$. În acest caz, modificăm M să fie

$$M := (M - \{(x_{i_{k+1}}, y_{j_k}) \mid 1 \leq k < p\}) \cup \{(x_{i_k}, y_{j_k}) \mid 1 \leq k \leq p\}.$$

Graful G_c modificat:



OBSERVAȚIE: $|M|$ CREȘTE CU 1

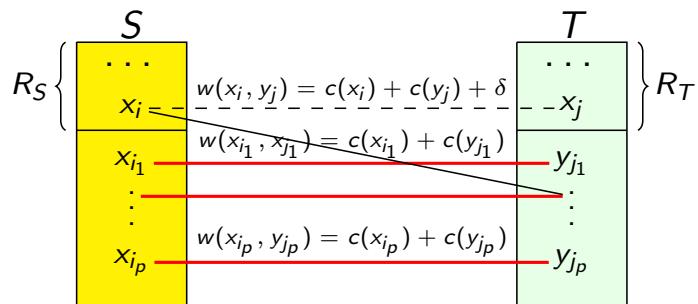
Cuplaje bipartite ponderate

Algoritmul ungar (3)

Dacă $R_T \cap Z = \emptyset$, fie

$$\delta = \min\{w(x, y) - c(x) - c(y) \mid x \in Z \cap S, y \in T \setminus Z\}$$

Graful G_c :

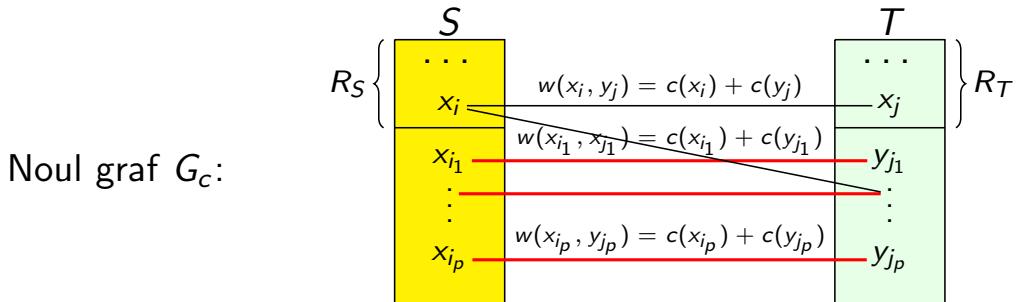


Se observă că $\delta > 0$. Modificăm c astfel:

- ▶ $c(x) := c(x) + \delta$ pentru $x \in Z \cap S$, și
- ▶ $c(y) = c(y) - \delta$ pentru toți $y \in Z \cap T$
- ⇒ G_c se modifică: $|R_T \cap Z|$ crește, iar $M \subseteq G_c$ continuă să aibe loc

Dacă $R_T \cap Z = \emptyset$, fie

$$\delta = \min\{w(x, y) - c(x) - c(y) \mid x \in Z \cap S, y \in T \setminus Z\}$$



Se observă că $\delta > 0$. Modificăm c astfel:

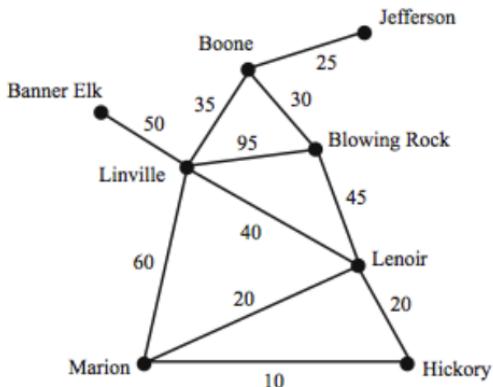
- ▶ $c(x) := c(x) + \delta$ pentru $x \in Z \cap S$, și
- ▶ $c(y) = c(y) - \delta$ pentru toți $y \in Z \cap T$
- ⇒ G_c se modifică: $|R_T \cap Z|$ crește, iar $M \subseteq G_c$ continuă să aibe loc

Remarcă

Problema motivațională poate fi rezolvată cu algoritmul ungar:

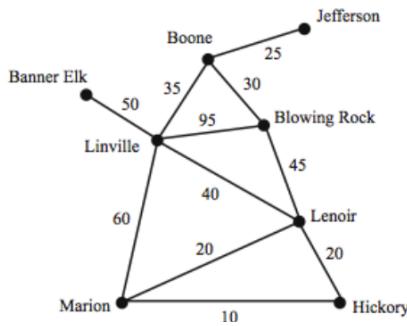
- ▶ Considerăm graful ponderat complet $K_{3,3}$ dintre $S = \{\text{Ion, Dan, Doru}\}$ și $T = \{\text{SB, M, SF}\}$ unde
SB: spălat baie, M: măturat, SF: spălat ferestre.
- ▶ Pentru acest graf, algoritmul ungar calculează un cuplaj perfect M , care indică lucrarea alocată fiecărui muncitor.

Departamentul de Transporturi din Carolina de Nord (NCDOT) a decis să realizeze o rețea feroviar rapidă între 8 orașe din vestul statului. Unele orașe sunt deja conectate cu drumuri, și se dorește plasarea de linii ferate de-a lungul drumurilor existente. Formele diferite de teren impun costuri diferite de amplasare a căii ferate. NCDOT a angajat un consultant să calculeze costurile de construire a unei căi ferate de-a lungul fiecărui drum de legătură între 2 orașe. Consultantul a produs graful ilustrat mai jos, în care sunt marcate costurile de realizare a fiecărei conexiuni. Se dorește ca rețeaua feroviară să fie realizată cu cost minim și să asigure legătură între orice 2 orașe.



Arbore de acoperire

Problemă motivantă



- ▷ Un **arbore de acoperire** al unui graf G este un arbore care conține toate nodurile lui G .
- ▷ Vrem să găsim un arbore de acoperire T al cărui cost total să fie minim, adică un **arbore minim de acoperire**:
 - Suma costurilor muchiilor lui $T \leq$ suma costurilor muchiilor oricărui alt arbore de acoperire al lui G .

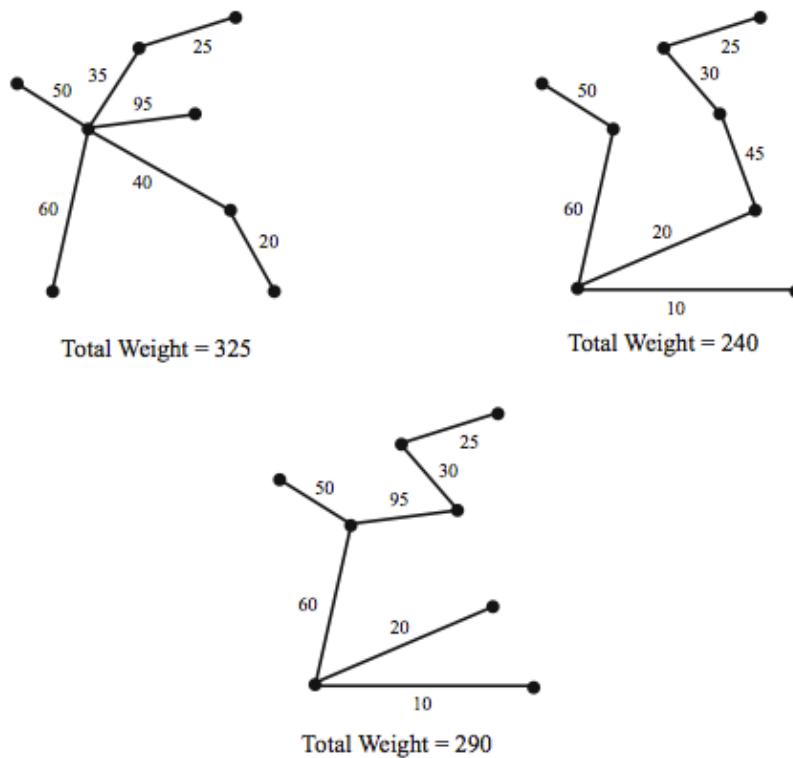


FIGURE 1.42. Several spanning trees.



Cursul 11

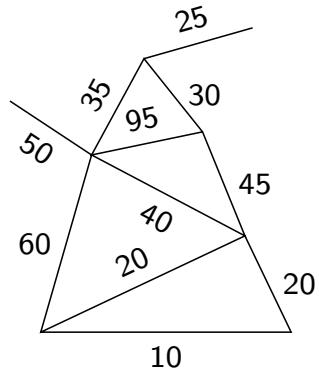
Găsirea unui arbore minim de acoperire

Algoritmul lui Kruskal

Se dă un graf ponderat și conectat G

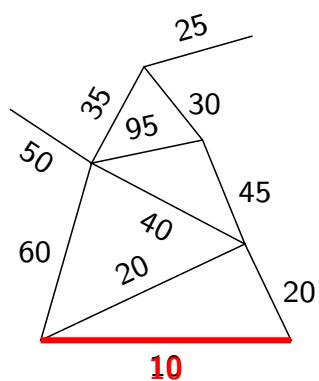
- (1) Se găsește o muchie cu pondere minimă și se marchează.
- (2) Se iau în considerare muchiile nemarcate care nu formează un ciclu cu cele marcate:
 - ▷ se alege o muchie nemarcată care are pondere minimă, și
 - ▷ se marchează.
- (3) Pasul (2) se repetă până când muchiile marcate formează un arbore de acoperire al lui G .

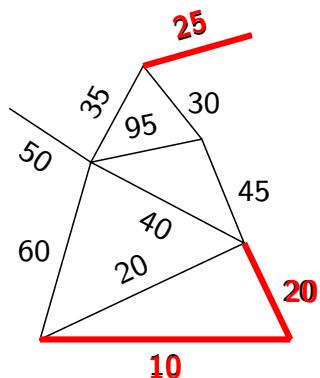
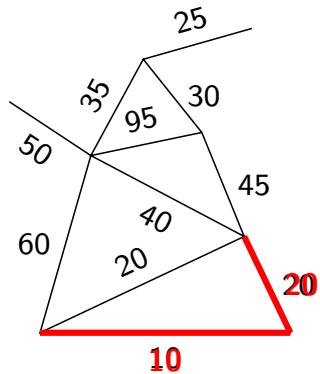
Cursul 11

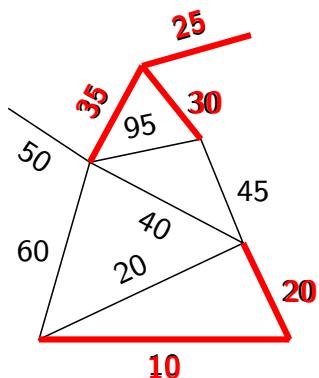
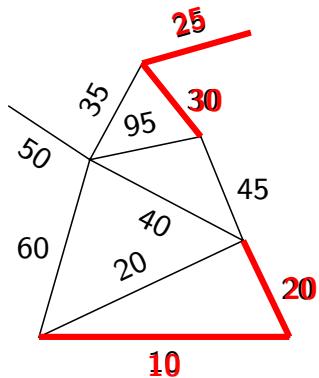


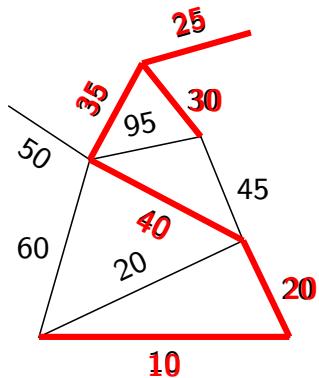
Algoritmul lui Kruskal

Pașii algoritmului pentru graful ilustrat



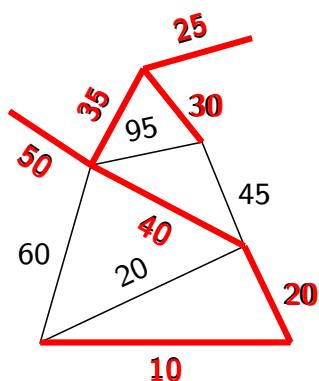


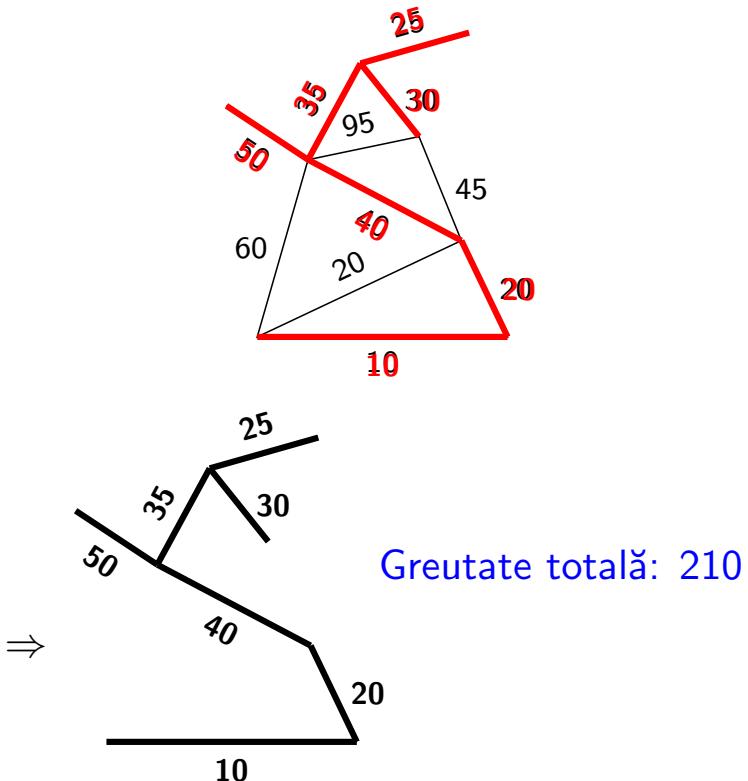




Algoritmul lui Kruskal

Pașii algoritmului pentru graful ilustrat

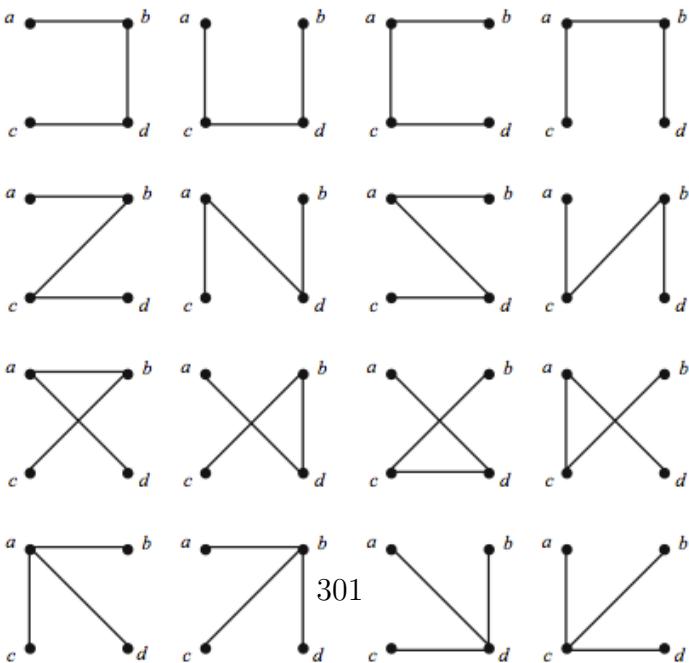




Enumerarea arborilor

Vrem să enumerăm toți arborii cu n noduri.

- Considerăm că pozițiile nodurilor sunt fixate și considerăm toate variantele de trasa un arbore între nodurile respective. De exemplu, pentru $n = 4$ avem 16 arbori etichetați diferenți:



Teoremă (Teorema lui Cayley)

Există n^{n-2} arbori distincți cu n noduri.

- Prüfer a găsit o metodă de enumerare a tuturor arborilor etichetați cu $1, 2, \dots, n$ prin intermediul unei corespondențe bijective între acești arbori și mulțimea secvențelor de numere de $n - 2$ numere între 1 și n :

$$\underbrace{v_1, v_2, \dots, v_{n-2}}_{n-2 \text{ numere}} \quad \text{unde } 1 \leq v_i \leq n$$

OBSERVAȚIE: Există n^{n-2} astfel de secvențe.

Enumerarea arborilor

Calculul secvenței Prüfer pentru un arbore T cu nodurile $1, 2, \dots, n$

Se dă un arbore T cu nodurile $1, \dots, n$

- (1) Inițial, secvența este vidă. Fie $i = 0$ și $T_0 = T$.
- (2) Se caută frunza lui T_i cu cea mai mică etichetă; fie aceasta v .
- (3) Se adaugă la secvență eticheta vecinului lui v .
- (4) Se șterge nodul v din $T_i \Rightarrow$ un arbore mai mic T_{i+1} .
- (5) Dacă T_{i+1} este K_2 , ne oprim. Altfel, incrementăm i cu 1 și revenim la pasul (2).

Arbore curent	secvență curentă
$T = T_0$ 	
T_1 	4
T_2 	4,3
T_3 	4,3,1
T_4 	4,3,1,3
T_5 	4,3,1,3,1



Enumerarea arborilor

Calculul arborelui T corespunzător unei secvențe a_1, \dots, a_k

Se dă o secvență $\sigma = a_1, a_2, \dots, a_k$ de numere din mulțimea $\{1, \dots, k+2\}$

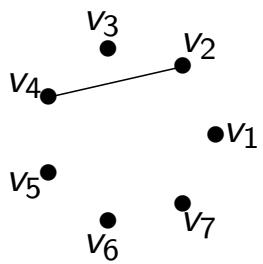
- (1) Se desenează $k+2$ noduri care se etichetează cu numerele $1, 2, \dots, k+2$. Fie $S = \{1, 2, \dots, k+2\}$.
- (2) Fie $i = 0$, $\sigma_0 = \sigma$ și $S_0 = S$.
- (3) Fie j cel mai mic număr din S_i care nu apare în secvența σ_i .
- (4) Se trasează o muchie între nodul j și primul nod din σ_i .
- (5) Se elimină primul nod din secvența $\sigma_i \Rightarrow$ secvența σ_{i+1} . Se sterge j din S_i și rezultă mulțimea S_{i+1} .
- (6) Dacă secvența σ_{i+1} este vidă, se trasează o muchie între nodurile rămase în S_{i+1} iar algoritmul se termină. Altfel, se incrementează i cu 1 și se revine la pasul (3).



Metoda lui Prüfer

Construirea unui arbore etichetat (1)

[Jump to Table of contents](#)

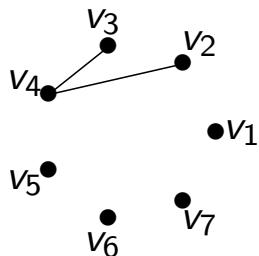


$$\sigma = \sigma_0 = 4, 3, 1, 3, 1$$

$$S = S_0 = \{1, 2, 3, 4, 5, 6, 7\}$$

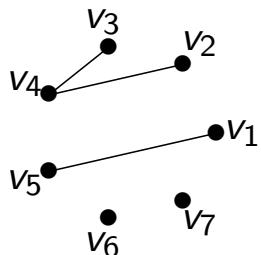
$$\sigma_1 = 3, 1, 3, 1$$

$$S_1 = \{1, 3, 4, 5, 6, 7\}$$



$$\sigma_2 = 1, 3, 1$$

$$S_2 = \{1, 3, 5, 6, 7\}$$



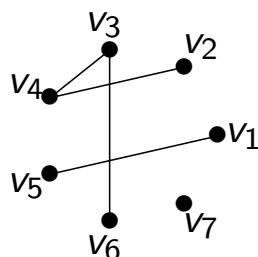
Cursul 11

Metoda lui Prüfer

Construirea unui arbore etichetat (2)

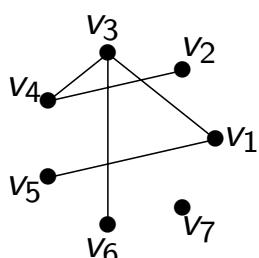
$$\sigma_3 = 3, 1$$

$$S_3 = \{1, 3, 6, 7\}$$



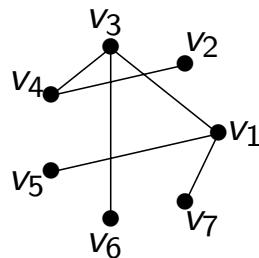
$$\sigma_4 = 1$$

$$S_4 = \{1, 3, 7\}$$



σ_5 este secvența vidă

$$S_5 = \{1, 7\}$$



8 CURS 8: Fluxuri in retele. Secțiuni, drumuri de creștere. Teorema flux maxim - secțiune minima. Algoritmii F&F și E&K.

Algoritmica grafurilor - Cursul 8

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

27 noiembrie 2020

Cuprins

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

1 Fluxuri în rețele

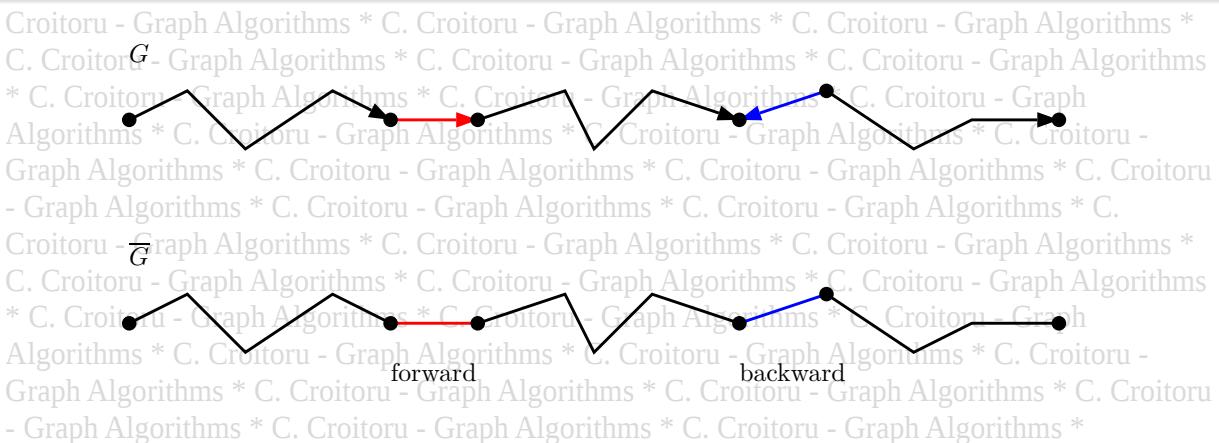
- **Drumuri de creștere**
- **Secțiuni**
- **Teorema drumului de creștere**
- **Teorema fluxului întreg**
- **Teorema flux maxim – secțiune minimă**
- **Algoritmul lui Ford & Fulkerson**
- **Algoritmul lui Edmonds & Karp**

2 Exerciții pentru seminarul din săptămâna viitoare

3 Exerciții rezolvate (partial)

Definiție

Fie P un drum în \tilde{G} – **graful suport** al digrafului G – și $e = v_i v_j$ o muchie a lui P , $e \in E(P)$. Dacă e corespunde arcului $v_i v_j$ atunci e este un **arc înainte (forward sau direct)** al lui P , altfel (e corespunde arcului $v_j v_i$) e este un **arc înapoi (backward sau invers)** al lui P .



Problema fluxului maxim - Drumuri de creștere

Definiție

Fie $R = (G, s, t, c)$ o rețea și x un flux în R . Un **A-drum** (în R relativ la fluxul x) este un drum P în \tilde{G} astfel încât $\forall ij \in E(P)$:

- dacă ij este un arc direct, atunci $x_{ij} < c_{ij}$,
- dacă ij este un arc invers, atunci $x_{ji} > 0$.

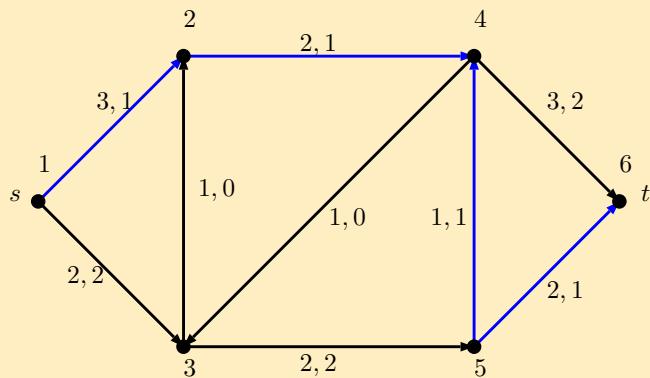
Dacă P este un A-drum în R relativ la fluxul x , atunci **capacitatea reziduală** a arcului $ij \in E(P)$ este

$$r(ij) = \begin{cases} c_{ij} - x_{ij}, & \text{dacă } ij \text{ este un arc direct al lui } P \\ x_{ji}, & \text{dacă } ij \text{ este un arc invers al lui } P \end{cases}$$

Capacitatea reziduală a drumului P este $r(P) = \min_{e \in E(P)} r(e)$.

Exemplu

Considerăm rețeaua de mai jos , unde, pe fiecare arc, prima etichetă este capacitatea iar a doua este fluxul.



$P : 1, 12, 2, 24, 4, 45, 5, 56, 6$ este un A -drum de la s la t cu arcele forward 12 ($x_{12} = 1 < c_{12} = 3$), 24 ($x_{24} = 1 < c_{24} = 2$), 56 ($x_{56} = 1 < c_{56} = 2$), și arcul backward 45 ($x_{45} = 1 > 0$). Capacitatea reziduală a lui P : $r(P) = \min \{2, 1, 1, 1\} = 1$.

Problema fluxului maxim - Drumuri de creștere

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

Definiție

Un drum de creștere relativ la fluxul x în rețeaua $R = (G, s, t, c)$ este un A -drum de la s la t .

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

Lema 1

Dacă P este un drum de creștere relativ la fluxul x în $R = (G, s, t, c)$, atunci $x^1 = x \otimes r(P)$ definit prin

$$x_{ij}^1 = \begin{cases} x_{ij}, & \text{dacă } ij \notin E(P) \\ x_{ij} + r(P), & \text{dacă } ij \in E(P), ij \text{ este arc direct în } P \\ x_{ij} - r(P), & \text{dacă } ij \in E(P), ij \text{ este arc invers în } P \end{cases}$$

este un flux în R cu $v(x^1) = v(x) + r(P)$.

Demonstrație. Din definiția lui $r(P)$, constrângerile (i) sunt satisfăcute de x^1 . Constrângerile (ii) - verificate de x - nu sunt afectate pentru x^1 în vreun nod $i \notin V(P)$.

Pentru $i \in V(P)$ există exact două arce în P incidente cu i , e. g. li și ik . Avem următoarele două cazuri posibile:

a) li și ik sunt arce directe:

$$\begin{aligned} \sum_j x_{ji}^1 - \sum_j x_{ij}^1 &= \sum_{j \neq l} x_{ji} - \sum_{j \neq k} x_{ij} + x_{li}^1 - x_{ik}^1 \\ &= \sum_{j \neq l} x_{ji} - \sum_{j \neq k} x_{ij} + x_{li} + r(P) - x_{ik} - r(P) = \sum_j x_{ji} - \sum_j x_{ij} = 0. \end{aligned}$$

b) li este arc direct și ik arc invers:

$$\begin{aligned} \sum_j x_{ji}^1 - \sum_j x_{ij}^1 &= \sum_{j \neq l, k} x_{ji} - \sum_j x_{ij} + x_{li}^1 + x_{ki}^1 \\ &= \sum_{j \neq l, k} x_{ji} - \sum_j x_{ij} + x_{li} + r(P) + x_{ki} - r(P) = \sum_j x_{ji} - \sum_j x_{ij} = 0. \end{aligned}$$

Problema fluxului maxim - Drumuri de creștere

c) li arc invers și ik arc direct: similar cu b).

d) li și ik arce inverse: similar cu a).

$v(x^1)$ diferă de $v(x)$ datorită fluxului de pe arcul lt din P :

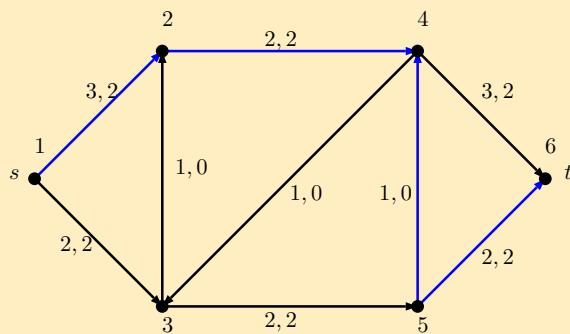
lt arc direct:

$$\begin{aligned} v(x^1) &= \sum_j x_{jt}^1 - \sum_j x_{tj}^1 = \sum_{j \neq l} x_{jt} - \sum_j x_{tj} + x_{lt}^1 = \\ &= \sum_{j \neq l} x_{jt} - \sum_j x_{tj} + x_{lt} + r(P) = v(x) + r(P). \end{aligned}$$

lt arc invers:

$$\begin{aligned} v(x^1) &= \sum_j x_{jt}^1 - \sum_j x_{tj}^1 = \sum_j x_{jt} - \sum_{j \neq l} x_{tj} - x_{tl}^1 = \\ &= \sum_j x_{jt} - \sum_{j \neq l} x_{tj} - (x_{tl} - r(P)) = v(x) + r(P). \quad \square \end{aligned}$$

Pentru exemplul de mai sus, fluxul $x^1 = x \otimes r(P)$ de valoare $v(x^1) = v(x) + r(P) = 3 + 1 = 4$ este:



Remarci

- Lema de mai sus explică și numele drumurilor de creștere și capacitatea reziduală.
- Din definiție, dacă P este un drum de creștere, atunci $r(P) > 0$ și $v(x \otimes r(P)) > v(x)$. Urmează că dacă există un drum de creștere relativ la fluxul x , atunci x nu este un flux de valoare maximă.

Problema fluxului maxim - Secțiuni

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

Definiție

Fie $R = (G, s, t, c)$ o rețea. O secțiune în R este o partiție (S, T) a lui $V(G)$ cu $s \in S$ și $t \in T$. Capacitatea secțiunii (S, T) este

$$c(S, T) = \sum_{i \in S} \sum_{j \in T} c_{ij}.$$

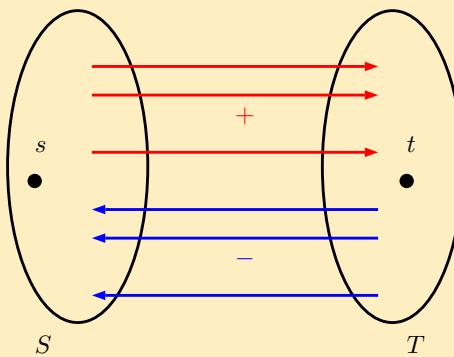
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Lema 2

Dacă x este un flux în $R = (G, s, t, c)$ și (S, T) este o secțiune în această rețea, atunci

$$v(x) = \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji})$$

(valoarea fluxului este fluxul net care trece prin secțiune).



Demonstrație.

$$\begin{aligned}
 v(x) &= \left(\sum_j x_{sj} - \sum_j x_{js} \right) + 0 = \\
 &= \left(\sum_j x_{sj} - \sum_j x_{js} \right) + \sum_{i \in S, i \neq s} \left(\sum_j x_{ij} - \sum_j x_{ji} \right) =
 \end{aligned}$$

Problema fluxului maxim - Secțiuni

Demonstrație (continuare).

$$\begin{aligned}
 &= \sum_{i \in S} \left(\sum_j x_{ij} - \sum_j x_{ji} \right) = \sum_{i \in S} \sum_j (x_{ij} - x_{ji}) = \\
 &= \sum_{i \in S} \sum_{j \in S} (x_{ij} - x_{ji}) + \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}) = \\
 &= 0 + \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}). \quad \square
 \end{aligned}$$

Lema 3

Dacă x este un flux în $R = (G, s, t, c)$ și (S, T) este o secțiune în această rețea, atunci

$$v(x) \leq c(S, T).$$

Demonstrație. Din Lema 2

$$\begin{aligned} v(x) &= \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}) \stackrel{x_{ij} \leq c_{ij}}{\leq} \sum_{i \in S} \sum_{j \in T} (c_{ij} - x_{ji}) \stackrel{x_{ji} \geq 0}{\leq} \\ &\stackrel{x_{ji} \geq 0}{\leq} \sum_{i \in S} \sum_{j \in T} c_{ij} = c(S, T). \quad \square \end{aligned}$$

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

Remarcă

Dacă \bar{x} este un flux în R și (\bar{S}, \bar{T}) este o secțiune astfel încât $v(\bar{x}) = c(\bar{S}, \bar{T})$, atunci, $\forall x$ flux în R , avem $v(x) \leq c(\bar{S}, \bar{T}) = v(\bar{x})$, i. e., \bar{x} este un flux de valoare maximă în R .

Similar, $\forall (S, T)$ secțiune în R , avem $c(S, T) \geq v(\bar{x}) = c(\bar{S}, \bar{T})$, adică, (\bar{S}, \bar{T}) este o secțiune de capacitate minimă în R .

Problema fluxului maxim - Teorema drumului de creștere

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

Teorema 1

Un flux x este un flux de valoare maximă dacă și numai dacă nu există un drum de creștere relativ la x în R .

Demonstrație. " \Rightarrow " Dacă P este un drum de creștere relativ la x , atunci $x \otimes r(P)$ este un flux în R de valoare strict mai mare.

" \Leftarrow " Fie x un flux în R cu proprietatea că nu există drum de creștere relativ la x în R . Fie

$$S = \{i : i \in V \text{ și } \exists P \text{ un } A\text{-drum în } R \text{ de la } s \text{ la } i\}.$$

Evident $s \in S$ (există un A -drum de lungime 0 de la s la s) și $t \notin S$ (nu există vreun A -drum de la s la t). Astfel, luând $T = V \setminus S$, (S, T) este o secțiune în R .

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

Demonstrație (continuare). $\forall i \in S$ și $\forall j \in T$ avem:

- dacă $ij \in E$, atunci $x_{ij} = c_{ij}$ și
- dacă $ji \in E$, atunci $x_{ji} = 0$

(altfel A -drumul de la s la i poate fi extins la un A -drum de la s la j , astfel $j \in S$ - contradicție). Urmărează că $v(x) = \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}) = \sum_{i \in S} \sum_{j \in T} c_{ij} = c(S, T)$, i. e., x este un flux de valoare maximă (vezi remarcă de mai sus). \square

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

Problema fluxului maxim - Teorema fluxului întreg

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

Teorema 2

(Teorema fluxului întreg) Dacă toate capacitatele din R sunt întregi atunci există un flux întreg, x , de valoare maximă (toate $x_{ij} \in \mathbb{Z}_+$).

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

Demonstrație. Considerăm următorul algoritm:

```
 $x^0 \leftarrow 0; i \leftarrow 0;$ 
while ( $\exists P_i$  un drum de creștere relativ la  $x^i$ ) do
     $x^{i+1} \leftarrow x^i \otimes r(P_i); i++;$ 
end while
```

Să observăm că " x^i are doar componente întregi" este un invariant al algoritmului (din definiția lui $r(P_i)$, dacă toate capacitatele sunt întregi și x^i este întreg, atunci $r(P_i)$ este întreg și astfel x^{i+1} este întreg).

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

Demonstrație (continuare). Mai mult, în fiecare iterare **while**, valoarea fluxului curent crește (cu cel puțin 1), astfel algoritmul se oprește în cel mult $c(\{s\}, V \setminus \{s\}) \in \mathbb{Z}_+$ iterății **while**.

Nu mai există drumuri de creștere relativ la fluxul final, astfel, din Teorema 1, fluxul este de valoare maximă. \square

Remarcă

Algoritmul de mai sus se oprește și când toate capacitatele sunt numere rationale.

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

Problema fluxului maxim - Teorema flux maxim – secțiune minimă

Teorema 3

(Teorema flux maxim – secțiune minimă) Valoarea maximă a unui flux în rețeaua $R = (G, s, t, c)$ este egală cu capacitatea minimă a unei secțiuni în R .

Linia demonstrației. Dacă descriem un algoritm care, plecând cu un flux inițial x^0 (e.g., $x^0 = 0$), construiește într-un număr finit de pași un flux x fără drumuri de creștere, atunci secțiunea considerată în demonstrația Teoremei 1 satisface, împreună cu x , cerința teoremei.

Pentru capacitați rationale, algoritmul considerat în demonstrația Teoremei 2 satisface această condiție. Pentru capacitați arbitrar reale vom prezenta mai târziu un astfel de algoritm datorat lui **Edmonds și Karp (1972)**.

Remarci

- O demonstrație scurtă a teoremei de mai sus constă în a arăta că există un flux de valoare maximă și de a aplica construcția din demonstrația Teoremei 1. Un flux de valoare maximă există întotdeauna observând că acesta este soluția optimă a unei probleme de programare liniară (peste un politop nevid).
- Importanța algoritmică a Teoremei 3 este dată de faptul că mulțimea tuturor secțiunilor dintr-o rețea este finită, pe când mulțimea tuturor fluxurilor este infinită.

Problema fluxului maxim - Algoritmul lui Ford & Fulkerson

Algoritmul întreține o etichetare a nodurilor din rețea pentru a determina drumuri de creștere relativ la fluxul curent x . Când nu mai există drumuri de creștere, fluxul x este de valoare maximă.

Fie $R = (G = (V, E), s, t, c)$ o rețea și x un flux în R .

Etiqueta unui nod j , care are trei componente (e_1, e_2, e_3) , semnifică: există un A -drum de la s la j , P , unde $e_1 = i$ este nodul dinaintea lui j pe acest drum, $e_2 \in \{direct, invers\}$ reprezintă direcția arcului ij , iar $e_3 = r(P)$.

Initial s este etichetat (\cdot, \cdot, ∞) . Celelalte noduri primesc eventual etichete prin vizitarea (scanarea) nodurilor deja etichetate:

```
procedure scan(i)
for (j ∈ V, neetichetat) do
    if (ij ∈ E și  $x_{ij} < c_{ij}$ ) then
        etichetează j cu e = (i, direct,  $\min\{e_3[i], c_{ij} - x_{ij}\}$ );
    end if
    if (ji ∈ E și  $x_{ji} > 0$ ) then
        etichetează j cu e = (i, invers,  $\min\{e_3[i], x_{ji}\}$ );
    end if
end for
```

Semnificația componentelor etichetelor este întreținută de procedura **scan**.

Când, în procedura **scan**, nodul *t* este etichetat, se poate detecta un drum de creștere *P*, relativ la fluxul curent *x*, astfel:

Problema fluxului maxim - Algoritmul lui Ford & Fulkerson

$r(P)$ este componenta e_3 a etichetei lui *t*, nodurile lui *P* pot fi găsite în $\mathcal{O}(n)$ prin explorarea primei componente a etichetelor, și modificarea $x \otimes r(P)$ se poate face în timpul acestei explorări folosind a doua componentă a etichetelor.

Pentru noul flux, se pornește cu o nouă etichetare (de la *s*).

Dacă toate noduri etichetate au fost scanate și nodul *t* nu a primit etichetă, urmează că fluxul curent nu admite drumuri de creștere, deci este de valoare maximă. Dacă *S* este mulțimea nodurilor etichetate și $T = V \setminus S$, atunci (S, T) este o secțiune de capacitate minimă în *R*.

```

pornește cu un flux inițial  $x = (x_{ij})$  (e. g.,  $x = 0$ )
 $e(s) \leftarrow (\cdot, \cdot, \infty)$ ;
while ( $\exists$  noduri etichetate și nescanate ) do
    alege  $i$  un nod etichetat și nescanat;
    scan( $i$ );
    if ( $t$  a fost etichetat) then
        modifică fluxul pe drum dat de etichete;
        șterge toate etichetele;  $e(s) \leftarrow (\cdot, \cdot, \infty)$ ;
    end if
end while
 $S \leftarrow \{i : i \in V, i$  este etichetat};
 $T \leftarrow V \setminus S$ ;

```

x este un flux de valoare maximă, (S, T) este o secțiune de capacitate minimă.

Problema fluxului maxim - Algoritmul lui Ford & Fulkerson

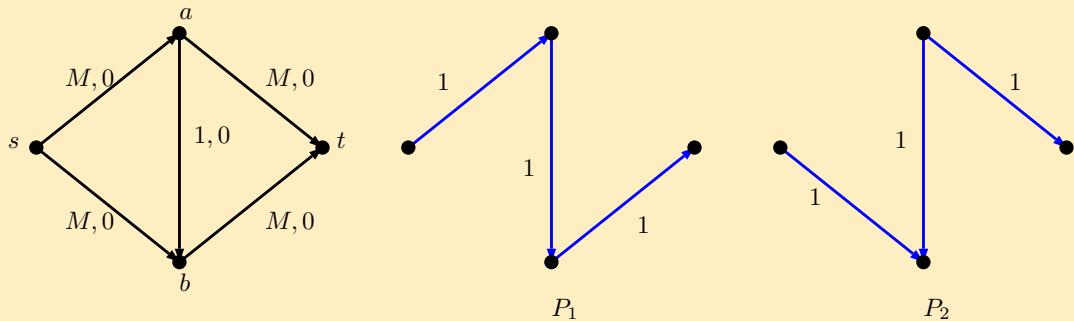
Complexitatea timp: Fiecare creștere a fluxului curent necesită cel mult $2m$ ($m = |E|$) inspecții ale arcelor pentru etichetarea altor noduri. Dacă toate capacitatele sunt intregi, sunt necesare cel mult v creșteri (v fiind valoarea fluxului maxim). Astfel algoritmul are complexitatea timp $\mathcal{O}(mv)$.

Dacă U este un majorant al tuturor capacitateilor pe arce, atunci $v \leq (n - 1)U$ (acesta este un majorant al secțiunii $(\{s\}, V \setminus \{s\})$), deci algoritmul are complexitatea timp $\mathcal{O}(nmU)$.

Remarcă

E posibil ca algoritmul să nu se termine pentru capacitați iraționale. Această situație nu apare în implementările practice, dar neajunsul acestei descrieri a algoritmului este dat de faptul că numărul de creșteri ale fluxului curent depinde de capacitați (și nu de dimensiunile rețelei).

Exemplu



Dacă operația **alege** din algoritmul de mai sus determină drumurile de creștere $P_1, P_2, P_1, P_2, \dots$, unde $P_1 = (s, sa, a, ab, b, bt, t)$ și $P_2 = (s, sb, b, ba, a, at, t)$, atunci fiecare creștere a fluxului se face cu 1, și algoritmul necesită $2M$ creșteri, ceea ce este prea mult.

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

Aceste deficiențe ale algoritmului pot fi evitate dacă alegerea nodurilor etichetate în vederea scanării nu se face aleatoriu (**Dinic (1970), Edmonds & Karp (1972)**).

Definiție

Un cel mai scurt drum de creștere relativ la fluxul x în R este un drum de creștere de lungime minimă printre toate drumurile de creștere relativ la x .

Fie x un flux în rețeaua R . Fie x^k ($k \geq 1$) secvența de fluxuri:

$$x^1 \leftarrow x;$$

$x^{k+1} \leftarrow x^k \otimes r(P_k)$, P_k un cel mai scurt drum de creștere relativ la x^k ;

Pentru a arăta că această secvență este finită, fie $\forall i \in V$ și $\forall k \in \mathbb{N}^*$:

$\sigma_i^k =$ lungimea minimă a unui A -drum de la s la i relativ la x^k .

$\tau_i^k =$ lungimea minimă a unui A -drum de la i la t relativ la x^k .

Lema 4

$\forall i \in V$ și $\forall k \in \mathbb{N}^*$ avem

$$\sigma_i^{k+1} \geq \sigma_i^k \text{ și } \tau_i^{k+1} \geq \tau_i^k.$$

Demonstrație. Omisă.

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

Teorema 4

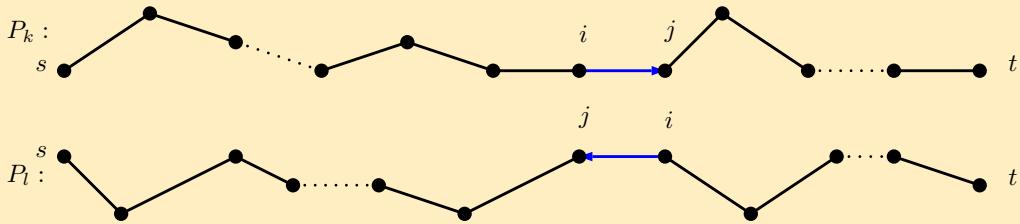
(Edmonds & Karp, 1972) Dacă $x = x^1$ este un flux arbitrar în rețeaua R , atunci secvența de fluxuri x^1, x^2, \dots , obținută din x^1 prin creșteri succesive cu drumuri de creștere de lungime minimă, are cel mult $mn/2$ termeni (în cel mult $mn/2$ creșteri succesive se obține un flux x cu proprietatea că nu există drum de creștere relativ la x).

Demonstrație. Dacă P este un drum de creștere relativ la un flux x în R , cu capacitatea reziduală $r(P)$, un **arc critic** în P este un arc $e \in E(P)$ cu $r(e) = r(P)$. În $x \otimes r(P)$, fluxul de pe arce critice devine fie egal cu capacitatea (pe arcele directe), sau nul (pe arcele inverse).

Fie ij un arc critic de pe un cel mai scurt drum de creștere P_k relativ la x^k . Lungimea lui P_k este $\sigma_i^k + \tau_j^k = \sigma_j^k + \tau_j^k$

Deoarece ij este a critic în P_k , în x^{k+1} nu mai poate fi utilizat în aceeași direcție ca în P_k . Fie P_l (cu $l > k$) primul cel mai scurt drum de creștere relativ la x^l în care fluxul de pe arcul ij va fi modificat, (când arcul va fi folosit în direcție inversă decât în P_k). Avem două cazuri:

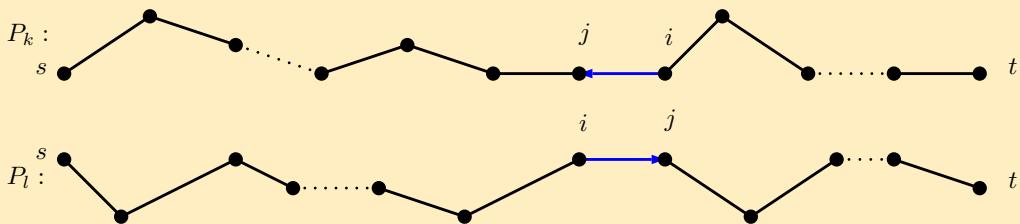
ij este un arc direct în P_k . Atunci $\sigma_j^k = \sigma_i^k + 1$; în P_l ij va fi arc invers, deci $\sigma_i^l = \sigma_j^l + 1$.



Urmează că $\sigma_i^l + \tau_i^l = \sigma_j^l + 1 + \tau_i^l \geq \sigma_j^k + 1 + \tau_i^k = \sigma_i^k + \tau_i^k + 2$ (din Lema 4). Am obținut că $\text{length}(P_l) \geq \text{length}(P_k) + 2$.

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

ij este un arc invers în P_k . Atunci $\sigma_i^k = \sigma_j^k + 1$; în P_l ij va fi arc direct, deci $\sigma_j^l = \sigma_i^l + 1$.



Urmează că $\sigma_j^l + \tau_j^l = \sigma_i^l + 1 + \tau_j^l \geq \sigma_i^k + 1 + \tau_j^k = \sigma_j^k + \tau_j^k + 2$. Obținem că $\text{length}(P_l) \geq \text{length}(P_k) + 2$.

Astfel orice cel mai scurt drum de creștere pe care arcul ij este critic are lungimea cu cel puțin 2 mai mare decât lungimea drumului precedent pe care ij a fost critic. Deoarece lungimea unui drum în G este cel mult $n - 1$, urmează că un arc fixat nu poate fi critic de mai mult de $n/2$ ori (de-a lungul întregului proces de creștere).

Orice drum de creștere are cel puțin un arc critic. Astfel în secvența (P_k) avem cel mult $mn/2$ drumuri de creștere cele mai scurte.

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

Corolar

În orice rețea există un flux x cu proprietatea că nu există drumuri de creștere relativ la x .

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

Remarci

- Demonstrația Teoremei 4 este acum încheiată.
- Singura modificare din **Algoritmul lui Ford & Fulkerson** este în alegerea nodului etichetat care va fi scanat: se folosește regula "primul etichetat-primul scanat" adică, se întreține o coadă a nodurilor etichetate (initializată cu s , la fiecare început de creștere).

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

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

În concluzie, avem următoarea teoremă.

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

Teorema 5

(Edmonds-Karp, 1972) Dacă în **Algoritmul lui Ford & Fulkerson** scanarea noduri etichetate se face într-o manieră bfs, atunci un flux de valoare maximă se obține în $\mathcal{O}(m^2n)$ time.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
Demonstrație. Există $\mathcal{O}(mn)$ creșteri de flux (din Teorema 4), fiecare de complexitate $\mathcal{O}(m)$. □

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo321ns * C. Croitoru - Graph Algorithms *

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

Exercițiu 2. Considerăm o rețea $R = (G, s, t, ; c)$, unde $G = (V, E)$ are n noduri și m arce, și funcția de capacitate are doar valori întregi ($c : E \rightarrow \mathbb{Z}_+$). Fie $C = \max_{e \in E} c(e)$.

- (a) Arătați că valoarea maximă a unui flux în R este cel mult $m \cdot C$.
- (b) Arătați că, pentru fiecare flux x din R și pentru fiecare $K \in \mathbb{Z}_+$, putem găsi un drum de creștere P cu capacitatea reziduală, $r(P)$, cel puțin K - dacă un asemenea drum există - în $\mathcal{O}(m)$.

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

Exercițiu 2 (continuare).

- (c) Considerăm următorul algoritm:

```
SC-MAX-FLOW( $R$ ) {
     $C \leftarrow \max_{e \in E} c(e);$ 
     $x \leftarrow 0; // x fluxul curent;$ 
     $K \leftarrow 2^{1+\lfloor \log C \rfloor};$ 
    while( $K \geq 1$ ) {
        while( $x$  are un drum de creștere  $P$  cu  $r(P) \geq K$ )
             $x \leftarrow x \otimes r(P);$ 
         $K \leftarrow K/2;$ 
    }
    return  $x;$ 
}
```

Exercițiu 2 (continuare).

1. Arătați că procedura SC-MAX-FLOW(R) returnează un flux de valoare maximă x în R .
2. Arătați că, după fiecare iterație **while** exterioară, valoarea maximă a unui flux în R este cel mult $v(x) + m \cdot K$.
3. Arătați că, $\forall K \in \mathbb{Z}_+$, există cel mult $2m$ iterării **while** interioare. În consecință procedura are complexitatea timp $\mathcal{O}(m^2 \log C)$.

Exercițiu 3. Digraful $G = (V, E)$ reprezintă topologia unei rețele de procesoare. Fiecare procesor, $v \in V$, îi cunoaștem încărcarea, $load(v) \in \mathbb{R}_+$. Folosind un flux maxim într-o anumită rețea determinați o strategie statică de echilibrare a încărcării (static load balancing strategy) în G : indicați pentru fiecare procesor încărcarea ce trebuie trimisă și cărui procesor aşa încât toate procesoarele să aibă aceeași încărcare.

Exerciții pentru seminarul din 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

Exercițiu 4. Fie $R = (G, s, t, c)$ o rețea și $(S_i, T_i)(i = \overline{1, 2})$ două secțiuni de capacitate minimă în R . Arătați că $(S_1 \cup S_2, T_1 \cap T_2)$ și $(S_1 \cap S_2, T_1 \cup T_2)$ sunt de asemenea secțiuni de capacitate minimă.

Exercițiu 5. Fie $R = (G = (V, E), s, t, c)$ o rețea și $c : E \rightarrow \mathbb{Z}_+$, $n = |V|$, $m = |E|$.

- (a) Descrieți un algoritm de complexitate timp $\mathcal{O}(n + m)$ pentru a determina o secțiune de capacitate minimă în R , având la îndemână un flux de valoare maximă x^* .
- (b) Folosind un algoritm de flux maxim pentru o anumită funcție de capacitate, arătați că se poate găsi o secțiune de capacitate minimă în R , (S_0, T_0) , cu număr minim de arce.

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

Exercițiu 6. Fie $G = (S, T; E)$ un graf bipartit. Demonstrați teorema lui Hall (*există un cuplaj în G care saturează toate nodurile din S dacă și numai dacă (H) $\forall A \subseteq S, |N_G(A)| \geq |A|$*) folosind **teorema flux maxim - secțiune minimă** pe o rețea particulară.

Exercițiu 7. Fie $R = (G, s, t, c)$ o rețea care are toate capacitatele întregi pozitive și pare. Arătați că există un flux maxim cu toate valorile întregi pozitive și pare.

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

Exercițiu 8 (continuare).

Arătați că există un flux legal în R dacă și numai dacă există un flux în \overline{R} de valoare $M = \sum_{e \in E(G)} m(e)$.

- (c) Folosind un flux legal inițial *legal flux* (vezi b)) arătați Ford&Fulkcum se poate modifica algoritmul Ford-Fulkerson pentru a obține un flux legal de valoare maximă.

Exercițiu 9. Fie $R = (G, s, t, c)$ o rețea de transport.

- (a) Descrieți un algoritm eficient care să verifice dacă o secțiune dată, (S, T) , este de capacitate minimă în R .
- (b) Descrieți un algoritm de complexitate timp $\mathcal{O}(n + m)$ care să verifice dacă un flux dat, x , este de valoare maximă în R .
- (c) Fie $c : E \rightarrow \mathbb{Z}_+$ și x^* este un flux întreg de valoare maximă în R . Să presupunem că pe un arc, $e_0 \in E(G)$, capacitatea crește cu 1.
³²⁴ Descrieți un algoritm eficient care să determine un flux maxim în noua rețea.

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

Exercițiu 10. Fie x un flux întreg într-o rețea dată $R = (G, s, t, c)$, cu $v(x) = v_1 + v_2 + \dots + v_p$, unde $v_i \in \mathbb{N}^*$, $\forall i = \overline{1, p}$, $p \geq 1$. Arătați că există p fluxuri întregi x^1, x^2, \dots, x^p astfel ca $v(x^i) = v_i$, $\forall i = \overline{1, p}$ și

$$x = \sum_{i=1}^p x^i.$$

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

Exercițiu 2. Soluție.

(a) Valoarea unui flux x în R poate fi majorată astfel:

$$v(X) = \sum_{su \in E} x(su) - \sum_{vs \in E} x(vs) \leq \sum_{su \in E} x(su) \leq \sum_{su \in E} c(su) \leq m \cdot C$$

(b) Dacă un astfel de drum există, poate fi găsit folosind algoritmul Ford-Fulkerson pentru determinarea drumurilor de creștere, dar cu următoarele restricții (de ce?):

- pentru un arc $uv \in E$, dacă u este etichetat, v poate fi și el etichetat (dacă nu este deja) numai dacă $c(uv) > x(uv) + K$;
- pentru un arc $vu \in E$, dacă u este etichetat, v poate fi și el etichetat (dacă nu este deja) numai dacă $x(vu) > K$;

- În acest fel capacitatea reziduală de pe fiecare arc eligibil este cel puțin K (de ce?). Inspectarea tuturor acestor arce necesită timpul $\mathcal{O}(m)$.

- (c1) An drum de creștere în R nu poate avea o capacitate reziduală strict mai mare decât C ; cum $2^{1+\lfloor \log C \rfloor} \geq C$, procedura de mai sus detectează toate drumurile de creștere relativ la x , deci la final x este un flux maxim.
- (c2) Să presupunem că, fluxul curent x , nu admite drumuri de creștere P de capacitate reziduală $\delta(P) \geq K$ (aceasta se întâmplă după o iterare în *while*-ul exterior).
- Dacă începem o nouă procedură de etichetare ca în (b) vom termina cu o tăietură (S, T) în care toate nodurile etichetate sunt în S , $s \in S, t \in T$ și:
 - $\forall uv \in E$, if $u \in S$ și $v \in T$, atunci $c(uv) \leq x(uv) + K$ (**de ce?**);
 - $\forall vu \in E$, if $u \in S$ și $v \in T$, atunci $x(vu) \leq K$ (**de ce?**);

Exerciții rezolvate (partial)

- Pe de altă parte, v_0 , valoarea maximă a unui flux în R este:

$$\begin{aligned}
v_0 &\leq c(S, T) = \sum_{\substack{u \in S, v \in T, \\ uv \in E}} c(uv) \leq \sum_{\substack{u \in S, v \in T, \\ uv \in E}} [x(uv) + K] = \\
&= \text{why? } v(x) + \sum_{\substack{u \in S, v \in T, \\ uv \in E}} K + \sum_{\substack{u \in S, v \in T, \\ vu \in E}} x(vu) \leq \\
&\leq v(x) + \sum_{\substack{u \in S, v \in T, \\ uv \in E}} K + \sum_{\substack{u \in S, v \in T, \\ vu \in E}} K \leq v(x) + m \cdot K
\end{aligned}$$

- (c3) Fie x fluxul curent, înaintea intrării într-o nouă iterare a *while*-ului exterior pentru valoarea curentă a lui K .

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

- Din (c2) știm că următoarea iterație în bucla **while** exterioară va crește fluxul cu cel mult $2m \cdot K$.
- Fiecare iterare din bucla **while** interioară va crește valoarea acestui flux cu cel puțin K ; astfel, vor fi cel mult $2m$ iterării în bucla **while** interioară.
- Există $(2 + \lfloor \log C \rfloor)$ iterării în bucla **while** exterioară.
- Deci, complexitatea timp totală este $\mathcal{O}(m^2 \log C)$ (de ce?).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
 C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

Exercițiul 3. Soluție. Fie m încărcarea medie totală,

$$m = \frac{1}{n} \sum_{v \in V} \text{load}(v) \text{ și } V_< = \{v \in V : \text{load}(v) < m\},$$

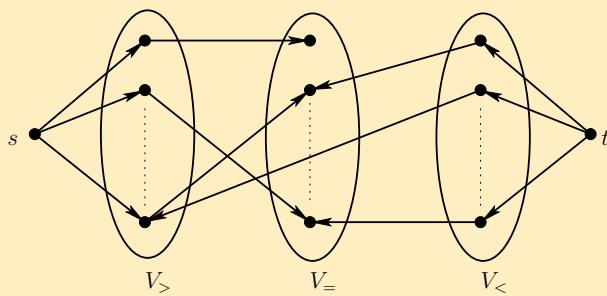
$$V_< = \{v \in V : \text{load}(v) = m\}, \quad V_> = \{v \in V : \text{load}(v) > m\}$$

- $(V_<, V_<, V_>)$ este o partiție (slabă) a lui V .
- Adăugăm două noduri noi s, t (sursa și destinația fictive) și definim un digraf $G' = (V', E')$, unde $V' = V \cup \{s, t\}$, $E' = E \cup \{sv : v \in V_>\} \cup \{ut : u \in V_<\}$.
- Definim deasemeni capacitatea arcelor:

$$c(sv) = \text{load}(v) - m, \quad \forall v \in V_>,$$

$$c(ut) = m - \text{load}(v), \quad \forall u \in V_<,$$

$$c(xy) = +\infty, \quad \forall xy \in E.$$



- Putem să arătăm că există o strategie de echilibrare statică dacă și numai dacă rețeaua $R = (G', s, t, c)$ are un flux φ care saturează toate arcele care pleacă din s (sau echivalent, toate arcele care intră în t - de ce?).

Exerciții rezolvate (partial)

- " \implies " Să presupunem că știm pentru orice $uv \in E$ încărcarea $\varphi(uv)$ care trebuie transferată de la u către v .
- Definim $\varphi(sv) = c(sv), \forall v \in V_>$ și $\varphi(ut) = c(ut), \forall u \in V_<$.
- Evident $0 \leq \varphi(xy) \leq c(xy), \forall xy \in E'$; pentru un procesor $v \in V_>$ supra-încărcarea $load(v) - m$ va părăsi v , deci

$$\varphi(sv) = load(v) - m = \sum_{vx \in E} \varphi(vx) - \sum_{yv \in E} \varphi(yv)$$

Un argument similar pentru orice $u \in V_<$:

$$\varphi(ut) = m - load(u) = \sum_{xu \in E} \varphi(xu) - \sum_{uy \in E} \varphi(uy)$$

Acstea două relații arată că legea de conservare a fluxului este respectată.

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

- " \Leftarrow " Dacă φ este un flux în R , atunci din legea de conservare orice procesor supra-încărcat își va descărca această supra-încărcare către celealte procesoare iar orice procesor sub-încărcat va primi încărcarea necesară până la m de la vecinii săi.
 - **Care poate fi ordinea de transfer a încărcării între procesoare?** (Cu alte cuvinte cum se poate evita situația în care $load(uv)$ este mai mare decât încărcarea curentă din u ?)

Exercitii rezolvate (partial)

Exercițiu 4. Soluție.

- Arătăm că $(S_1 \cup S_2, T_1 \cap T_2)$ este o tăietură de capacitate minimă; evident că este o bipartiție a lui $V(G)$ (**de ce?**), și $s \in S_1 \cup S_2$, $t \in T_1 \cap T_2$.
 - Fie x un flux maxim în R ($c(S_1, T_1) = c(S_2, T_2) = v(x)$), avem

$c(uv) = x(uv), c(vu) = 0, \forall u \in S_i, v \in T_{i+1}$, (de ce?)

- Astfel,

$$c(uv) = x(uv), x(vu) = 0, \forall u \in S_1 \cup S_2, \forall v \in T_1 \cap T_2$$

$$\Rightarrow c(S_1 \cup S_2, T_1 \cap T_2) = v(x)$$

- Deci $(S_1 \cup S_2, T_1 \cap T_2)$ este deasemeni o tăietură de capacitate minimă (de ce?).

Exercițiu 5. Soluție.

- (b) Vom folosi notația $|(S, T)| := |\{uv \in E(G) : u \in S, v \in T\}|$, pentru orice tăietură (S, T) .
- Fie $R' = (G, s, t, c')$ o rețea cu $c'(uv) = (|E(G)| + 1)c(uv) + 1$. În acest fel, pentru orice tăietură (S, T) we have $c'(S, T) = (|E(G)| + 1) \cdot c(S, T) + |(S, T)|$.
 - Suppose now that (S_0, T_0) și (S, T) are two cuts of G , (S_0, T_0) of capacitate minimă tăietură in R' :

$$\begin{aligned} c'(S_0, T_0) &= (|E(G)| + 1) \cdot c(S_0, T_0) + |(S_0, T_0)| \stackrel{\text{why?}}{\leq} \\ &\leq c'(S, T) = (|E(G)| + 1) \cdot c(S, T) + |(S, T)| \Rightarrow \\ &\Rightarrow (|E(G)| + 1) \cdot [c(S_0, T_0) - c(S, T)] + |(S_0, T_0)| \stackrel{\text{why?}}{\leq} |(S, T)| \end{aligned}$$

Exerciții rezolvate (partial)

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

- Sacă presupunem că $c(S_0, T_0) \geq c(S, T) + 1$, atunci

$$|E(G)| + 1 + |(S_0, T_0)| \leq |(S, T)| \leq |E(G)| - \text{contradictie.}$$

- Dacă (S_0, T_0) este o tăietură de capacitate minimă în R' și (S, T) de capacitate minimă în R avem $c(S_0, T_0) = c(S, T)$ și

$$\begin{aligned} c'(S_0, T_0) &= (|E(G)| + 1) \cdot c(S_0, T_0) + |(S_0, T_0)| \stackrel{\text{dece?}}{\leq} \\ &\leq (|E(G)| + 1) \cdot c(S_0, T_0) + |(S, T)|, \end{aligned}$$

astfel, $|(S_0, T_0)| \leq |(S, T)|$.

Algorithms * C. Croitoru - Graph Algo330ns * C. Croitoru - Graph Algorithms *

Exercițiu 6. Soluție. Fie $R = (H, s, t, c)$ următoarea rețea:

$$V(H) = V(G) \cup \{s, t\}, E(H) = \{su : u \in S\} \cup \{uv : uv \in E(G)\} \cup \{vt : v \in T\},$$

$$c(su) = 1, \forall u \in S; c(uv) = 1, \forall uv \in E(G); c(vt) = 1, \forall v \in T.$$

- Evident, oricărui cuplaj M din G îi corespunde un flux întreg x_M în R :

$$x_M(uv) = 1, \text{ dacă } uv \in M; x_M(su) = 1, \text{ dacă } u \in S(M),$$

$$x_M(vt) = 1, \text{ dacă } v \in S(M) \text{ și } x_M(xy) = 0 \text{ pe orice arc } xy.$$

Exerciții rezolvate (partial)

- Reciproc, oricărui flux întreg x din R îi putem asocia un cuplaj M_x of G :

$$M_x = \{uv \in E(G) : x(uv) = 1, u \in S \text{ și } v \in T\}.$$

- Mai mult, $|M| = v(x_M)$.
- Dacă $M \in \mathcal{M}_G$ saturează toate nodurile din S ($S \subseteq S(M)$), atunci x_M va fi un flux maxim în R (de ce?).
- Fie $A \subseteq S$ și $(\Sigma, \Theta) = (A \cup N_G(A) \cup \{s\}, V(G) \setminus S)$,

$$c(\Sigma, \Theta) = \sum_{v \in N_G(A)} c(vt) + \sum_{u \in S \setminus A} c(su) \geq v(x_M), \forall A \subseteq S$$

ceea ce este echivalent cu (de ce?):

$$|N_G(A)| + |S \setminus A| \geq |S|, \forall A \subseteq S \Leftrightarrow (H) : |N_G(A)| \geq |A|, \forall A \subseteq S.$$

- Dacă (H) are loc, atunci fie (Σ, Θ) o tăietură de capacitate minimă din R .
- Fie $A = \Sigma \cap S$ și $B = \Sigma \cap T$.
- $B \subseteq N_G(A)$ (de ce?)
- Pentru un nod $v \in N_G(A)B$, avem $c(\Sigma \cup \{v\}, \Theta \setminus \{v\}) = c(\Sigma, \Theta) - |N_G(v) \cap A| + 1 \leq c(\Sigma, \Theta)$.
- Astfel putem presupune că $N_G(A) = B$.
- Pentru un flux maxim întreg x :

$$\begin{aligned} |M_x| &= v(x) = c(\Sigma, \Theta) = |S \setminus A| + |N_G(A)| \\ &\geq |S \setminus A| + |A| = |S| \Rightarrow |M_x| = |S|, \text{ deci } S \subseteq S(M_x). \end{aligned}$$

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

Exercițiu 7. Soluție.

- Se folosește algoritmul Edmonds-Karp plecând cu un flux inițial

Exerciții rezolvate (partial)

Exercițiu 9. Soluție.

- (a) Fie (S, T) o tăietură și x un flux în R ; avem

$$c(S, T) = \sum_{i \in S} \sum_{j \in T} c_{ij} \geq \sum_{i \in S} \sum_{j \in T} x_{ij} \geq \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}) = v(x).$$

- Dacă x este flux de valoare maximă, atunci (S, T) îs de capacitate minimă dacă și numai dacă $c(S, T) = v(x)$ ceea ce este echivalent cu $c_{ij} = x_{ij}, \forall ij \in E(G), i \in S, j \in T$ (de ce?).
 - Mai întâi se determină un flux de valoare maximă în rețea și apoi se verifică relațiile de mai sus.
- (b) Se încearcă determinarea unui nou drum de creștere relativ la x - fluxul dat.
- (c) Se încearcă determinarea unui nou drum de creștere relativ la x^* - fluxul maxim dat.

Exercițiu 10. Soluție. Demonstrăm prin inducție după $v(x)$.

- Pentru $v(x) = 0$ proprietatea e evidentă.
- Fie $v(x) > 0$ și $x' : E(G) \rightarrow \mathbb{N}$ un flux întreg astfel încât $x'(e) \leq x(e)$, pentru orice $e \in E(G)$, $v(x') \geq v_1$ iar suma $\sum_{e \in E(G)} x'(e)$ este minimă (de ce există un astfel de flux?).
- Arătăm că $v(x') = v_1$.
- Presupunem prin reducere la absurd că $v(x') \neq v_1$ și fie $S = \{u \in V(G) : \exists P \in \mathcal{P}_{su} \text{ a. i. } x'(e) > 0, \forall e \in E(P)\}$.
- t trebuie să fie în S (de ce?).

Exerciții rezolvate (partial)

- Există un drum $P \in \mathcal{P}_{st}$ cu $x'(e) > 0, \forall e \in E(P)$.
- Scăzând cu 1 fluxul de pe arcele acestui drum obținem un flux x'' cu $x''(e) \leq x(e)$, pentru orice $e \in E(G)$, $v(x'') = v(x') - 1 \geq v_1$ (de ce?), și $\sum_{e \in E(G)} x''(e) < \sum_{e \in E(G)} x'(e)$, contradicție.
- Astfel, $x - x'$ este un flux de valoare $\sum_{i=2}^p v_i$ și putem aplica ipoteza inductivă.

9 CURS 9: Prefluxuri. Algoritmul Ahuja&Orlin. Aplicatii combinatoriale ale fluxurilor in retele.

Preflux :

- $0 \leq \text{fluxul} \leq \text{capacitatea}$
- suma fluxului muchiilor care intra in nod \geq cea careiese

Capacitatea reziduala : $R_{ij} = C_{ij} - x_{ij} + x_{ji}$ (x este fluxul)

Noduri active :

- $0 \leq \text{fluxul} \leq \text{capacitatea}$
- suma fluxului care intra in nod $<$ cea careiese

Excesul in noduri : (suma fluxului care intra)-(suma fluxului careiese)

Algoritmica Grafurilor - Cursul 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 -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
 Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
 C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

4 decembrie 2020

Cuprins

- 1 **Fluxuri în rețele**
 - **Prefluxuri**
 - Schema generală a unui algoritm de tip preflux
 - Algoritmul lui Ahuja & Orlin
 - **Aplicații combinatoriale**
 - Cuplaje în grafuri bipartite
 - Recunoașterea secvențelor digrafice
 - Conexiunea pe muchii
 - Conexiunea pe noduri
- 2 **Exercitii pentru seminarul din săptămâna viitoare**
- 3 **Exercitii rezolvate (partial)**

Definiție

Un preflux în $R = (G, s, t, c)$ este o funcție $x : E(G) \rightarrow \mathbb{R}$ astfel încât

- (i) $0 \leq x_{ij} \leq c_{ij}, \forall ij \in E;$
- (ii) $\forall i \neq s, e_i = \sum_{ji \in E} x_{ji} - \sum_{ij \in E} x_{ij} \geq 0.$

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

e_i (pentru $i \in V \setminus \{s, t\}$) este numit **excesul** din nodul i . Dacă $i \in V \setminus \{s, t\}$ și $e_i > 0$, atunci i este un **nod activ**. Dacă $ij \in E$, x_{ij} va fi numit **fluxul de pe arcul ij** .

Dacă în R nu există noduri active, atunci prefluxul x este un **flux** cu $v(x) = e_t$.

Ideea algoritmilor de tip preflux: un preflux inițial în R este transformat prin modificarea fluxurilor de pe arce **într-un flux cu proprietatea că nu există drumuri de creștere** în R relativ la el.

Prefluxuri

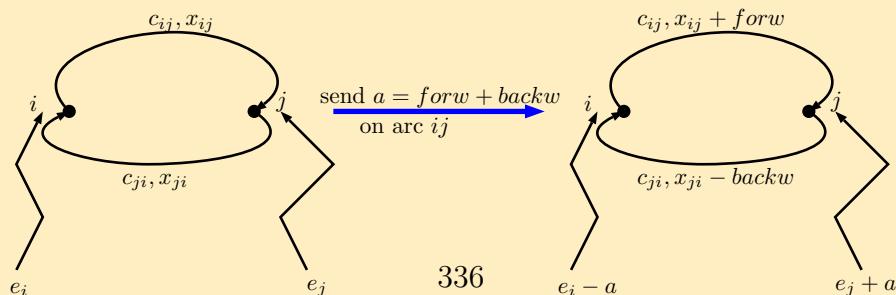
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

G este reprezentat folosind liste de adiacență. Vom presupune că dacă $ij \in E$, atunci $ji \in E$ de asemenei (altfel, adăugăm arcul ji de capacitate 0). Astfel, G devine digraf simetric.

Dacă x este un preflux în R și $ij \in E$, atunci **capacitatea reziduală** a lui ij este

$$r_{ij} = c_{ij} - x_{ij} + x_{ji},$$

(reprezentând fluxul suplimentar care poate fi trimis de la nodul i la nodul j folosind arcele ij și ji).



În cele de urmă, să trimitem flux de la i la j înseamnă creșterea fluxului pe arcul ij sau scăderea fluxului pe arcul ji .

Un A -drum în R relativ la prefluxul x , este un drum în G având toate arcele de capacitate reziduală strict pozitivă.

O funcție distanță în R relativ la prefluxul x este o funcție $d : V \rightarrow \mathbb{Z}_+$ a. î.

$$(D_1) \quad d(t) = 0,$$

$$(D_2) \quad \forall ij \in E, r_{ij} > 0 \Rightarrow d(i) \leq d(j) + 1.$$

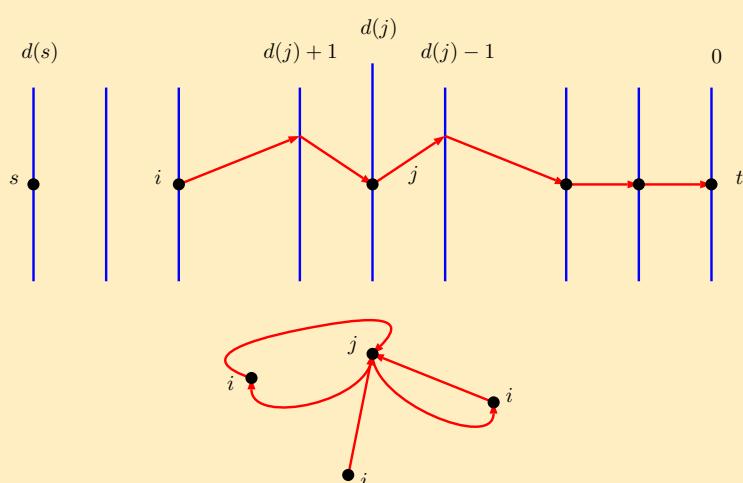
Remarci

- Dacă P este un A -drum relativ la prefluxul x în R de la i la t , atunci $d(i) \leq \text{length}(P)$ (arcele lui P au capacitate reziduale pozitive și apoi se aplică (D_2) în mod repetat). Urmează că (τ_i este lungimea minimă a unui A -drum de la i la t): $d(i) \leq \tau_i$.

Prefluxuri

Remarci

- Ca de obicei, notăm cu $A(i)$ lista de adiacență a nodului i .



Definiție

Fie x un preflux în R și d o funcție distanță relativ la x . Un arc $ij \in E$ este numit **admisibil** dacă $r_{ij} > 0$ și $d(i) = d(j) + 1$.

Dacă R este o rețea, considerăm următoarea procedură de inițializare, care construiește în $\mathcal{O}(m)$ un preflux x și o funcție distanță d relativ la x .

```

procedure initialization();
for ( $ij \in E$ ) do
  if ( $i = s$ ) then
     $x_{sj} \leftarrow c_{sj}$ 
  else
     $x_{ij} \leftarrow 0$ ;
   $d[s] \leftarrow n$ ;  $d[t] \leftarrow 0$ ;
for ( $i \in V - \{s, t\}$ ) do
   $d[i] \leftarrow 1$ ;
  
```

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

Prefluxuri

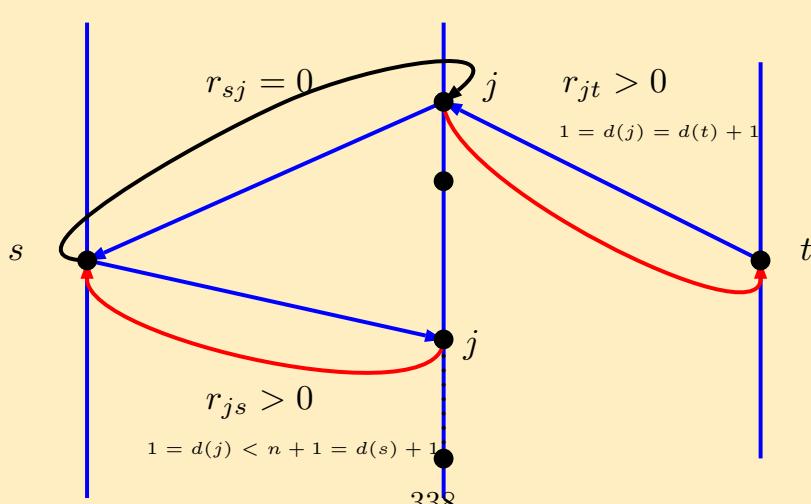
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Se observă că după execuția acestei proceduri, avem $r_{sj} = 0$, $\forall sj \in A(s)$. Astfel condiția (D_2) nu este not afectată de $d(s) = n$.
Pentru toate arcele ij , (D_2) este îndeplinită:

$$d(s) = n$$

$$d(j) = 1$$

$$d(t) = 0$$



Alegerea $d(s) = n$ are semnificația: nu există A -drumuri de la s la t relativ la x (altfel, dacă P este un astfel de drum, lungimea lui trebuie să fie cel puțin $d(s) = n$, ceea ce este imposibil).

Dacă acesta va fi un invariant al algoritmilor de tip preflux, atunci când x va deveni un flux, va fi un flux maxim.

Considerăm următoarele două proceduri:

```
procedure push(i); // i este un nod diferit de s, t  
alege un arc admisibil  $ij \in A(i)$ ;  
"trimite"  $\delta = \min \{e_i, r_{ij}\}$  (unități de flux) de la i la j;
```

Dacă $\delta = r_{ij}$ atunci avem o **push/pompare saturată**, altfel avem o **pompare nesaturată**.

```
procedure relabel(i); // i este un nod diferit de s, t  
 $d[i] \leftarrow \min \{d[j] + 1 : ij \in A(i) \text{ și } r_{ij} > 0\}$ ;
```

Prefluxuri - Schema generală a unui algoritm de tip preflux

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

```
initialization;  
while ( $\exists$  noduri active în  $R$ ) do  
    alege un nod activ i;  
    if ( $\exists$  arce admisibile în  $A(i)$ ) then  
        push(i);  
    else  
        relabel(i);
```

Lema 1

" d este funcție distanță relativ la prefluxul x " este un invariant al algoritmului de mai sus. La fiecare apel $relabel(i)$, $d(i)$ crește.

Demonstrație. Am demonstrat deja că procedura de inițializare construiește un preflux x și o funcție distanță d relativ la x . Arătăm că dacă perechea (d, x) satisface (D_1) și (D_2) înaintea unei iterări **while**, atunci după această iterare cele două condiții sunt de asemenei satisfăcute. Avem două cazuri, în funcție de care dintre procedurile **push** sau **relabel** este executată în iterarea **while** curentă:

Este apelată *push*(i): singura pereche care poate viola (D_2) este $d(i) \neq d(j)$. Deoarece ij este admisibil, la apelul *push*(i) avem $d(i) = d(j) + 1$. După execuția *push*(i), arcul ji poate avea $r_{ji} > 0$ (fără a fi astfel înaintea apelului), dar condiția $d(j) \leq d(i) + 1$ este evident satisfăcută.

Este apelată *relabel*(i): actualizarea lui $d(i)$ este astfel încât (D_2) este satisfăcută pentru fiecare arc ij cu $r_{ij} > 0$. Deoarece *relabel*(i) este apelată când $d(i) < d(j) + 1$, $\forall ij$ cu $r_{ij} > 0$, urmează că, după apel, $d(i)$ crește (cu cel puțin 1). \square

Prefluxuri - Schema generală a unui algoritm de tip preflux

Pentru a arăta că algoritmul se oprește, este necesar să arătăm că (în timpul execuției) dacă un nod i este activ atunci în lista lui de adiacență, $A(i)$, există cel puțin un arc ij cu $r_{ij} > 0$. Aceasta demonstrează lema următoare.

Lema 2

Dacă i_0 este un x -nod activ în R , atunci există un i_0s A -drum relativ la x .

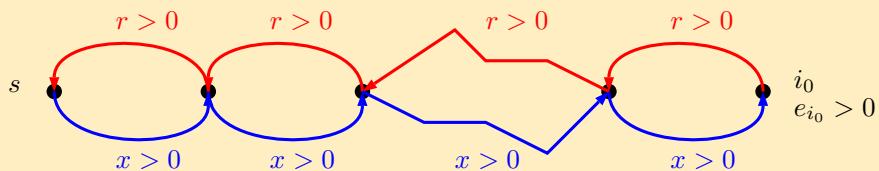
Demonstrație. Dacă x este un preflux în R , atunci x poate fi descompus în $x = x^1 + x^2 + \dots + x^p$, unde fiecare x^k are proprietatea că mulțimea $A^k = \{ij : ij \in E, x_{ij}^k \neq 0\}$ este

- (a) mulțimea arcelor unui drum de la s la t , sau
- (b) mulțimea de arce ale unui drum de la s la un nod activ, sau
- (c) mulțimea arcelor unui circuit.³⁴⁰

Mai mult, în cazurile (a) și (c), x^k este un flux. (Demonstrația este algoritmică: construim mai întâi mulțimile de tipul (a), apoi pe cele de tipul (c) și (b); la fiecare pas, căutăm inversul unui drum de tip (a) sau (c) (sau (b)); prefluxul obținut este scăzut din cel curent; deoarece excesele nodurilor sunt nenegative, construcția poate fi realizată, ori de câte ori prefluxul curent este nul; construcția este finită, deoarece numărul de arce pe care fluxul curent este 0, crește la fiecare pas.)

Deoarece i_0 este un nod activ în R relativ la x , cazul (b) va apărea pentru nodul i_0 (cazurile (a) și (c) nu afectează excesul din nodul i_0). Arcele inverse de pe acest drum au capacitatea reziduală pozitivă (vezi imaginea de mai jos), astfel ele formează A -drumul cerut. \square

Prefluxuri - Schema generală a unui algoritm de tip preflux



Corolarul 1

$$\forall i \in V, d(i) < 2n.$$

Demonstrație. Într-adevăr, dacă i nu a fost reetichetat, atunci $d(i) = 1 < 2n$. Altfel, înaintea apelului $relabel(i)$, i este un nod activ, astfel din Lema 2, există un is -drum P cu $length(P) \leq n - 1$. Din (D_2) , urmează că, după reetichetare, $d(i) \leq d(s) + n - 1 = 2n - 1$ ($d(s) = n$ nu este modificat vreodată). \square

Corolarul 2

Numărul total de apeluri ale procedurii relabel nu este mai mare decât $2n^2$.

Demonstrație. Într-adevăr, există $n - 2$ noduri care pot fi reetichetate. Fiecare dintre ele poate fi reetichetat de cel mult $2n - 1$ ori (din Lema 1, Corolarul de mai sus și distanța inițială d). \square

Corolarul 3

Numărul total de pompări saturate nu este mai mare decât nm .

Demonstrație. Într-adevăr, când un arc ij devine saturat, avem $d(i) = d(j) + 1$. După aceea, algoritmul nu va mai trimite flux pe acest arc până când nu va trimite flux pe arcul ji , când vom avea $d'(j) = d'(i) + 1 \geq d(i) + 1 = d(j) + 2$.

Prefluxuri - Schema generală a unui algoritm de tip preflux

Demonstrație (continuare). Astfel o modificare a acestui flux pe arcul ij nu va apărea până când $d(j)$ nu va crește cu 2. Urmează că un arc nu poate deveni saturat de mai mult de n ori și (deoarece numărul total de arce este m), nu există mai mult de nm pompări saturate. \square

Lema 3

(Goldberg și Tarjan, 1986). Numărul total de pompări nesaturate nu este mai mare decât $2n^2m$.

Demonstrație. Omisă.

Lema 4

Algoritmul de tip preflux returnează un flux de valoare maximă în R .

Demonstrație. Din Lemele 1 și 3 și din Corolarul 3 al Lemei 2, algoritmul se oprește în cel mult $\mathcal{O}(n^2m)$ iterații **while**. Deoarece $d(s) = n$ nu se modifică niciodată, urmează că nu există vreun drum de creștere relativ la fluxul x obținut, astfel x este de valoare maximă: dacă P ar fi un drum de creștere (în digraful suport al lui G), atunci înlocuind de-a lungul lui P fiecare arc invers cu simetricul său se obține un A -drum de la s la t , deci $n = d(s) \leq d(t) + n - 1 = n - 1$. \square

În locul demonstrației Lemei 3, vom prezenta **Algoritmul lui Ahuja & Orlin (1988)** care utilizează o **metodă de scalare - scaling method** pentru a reduce numărul de pompări nesaturate de la $\mathcal{O}(n^2m)$ (**Goldberg & Tarjan algoritm**) la $\mathcal{O}(n^2 \log U)$.

Prefluxuri - Algoritmul lui Ahuja & Orlin

Să presupunem că $c_{ij} \in \mathbb{N}$ și $U = \max_{ij \in E} (1 + c_{ij})$. Fie $K = \lceil \log_2 U \rceil$.

Ideea algoritmului: Avem $K + 1$ etape. În fiecare etapă p , cu p luând succesiv valorile $K, K - 1, \dots, 1, 0$, următoarele două condiții sunt îndeplinite:

- (a) la începutul etapei p , $e_i \leq 2^p$, $\forall i \in V \setminus \{s, t\}$.
- (b) în timpul etapei p , procedurile push-relabel sunt folosite pentru a elimina nodurile active, i , cu $e_i \in \{2^{p-1} + 1, \dots, 2^p\}$.

Din definiția lui K , în prima etapă ($p = K$), condiția (a) este îndeplinită, și dacă condiția (b) va fi menținută de-a lungul execuției algoritmului, după $K + 1$ etape, dacă se menține integralitatea exceselor de-a lungul algoritmului, atunci urmează că, după etapa $K + 1$, **excesul fiecărui nod $i \in V \setminus \{s, t\}$ este 0**, deci avem un **flux de valoare maximă**.

Pentru a menține (b) de-a lungul execuției algoritmului, schema generală a unui algoritm de tip preflux este adaptată după cum urmează:

- fiecare etapă p începe prin construirea listei $L(p)$ a tuturor nodurilor $i_1, i_2, \dots, i_{l(p)}$ cu excese $e_i > 2^{p-1}$, ordonate crescător după d (aceasta poate fi făcută cu hash-sorting în timpul $O(n)$, deoarece $d(i) \in \{1, 2, \dots, 2n - 1\}$).
- nodul activ selectat pentru **push-relabel** în timpul etapei p va fi **primul nod din $L(p)$** . Urmează că, dacă se face o pompare (push) pe un arc admisibil ij , atunci $e_i > 2^{p-1}$ și $e_j \leq 2^{p-1}$ (deoarece $d(j) = d(i) - 1$ și i este primul nod din $L(p)$). Dacă δ , fluxul trimis de la i la j de către apelul $push(i)$, este limitat la $\delta = \min(e_i, r_{ij}, 2^p - e_j)$, atunci (deoarece $2^p - e_j \geq 2^{p-1}$) urmează că o **pompare nesaturată** trimite cel puțin 2^{p-1} unități de flux.

Prefluxuri - Algoritmul lui Ahuja & Orlin

După apelul **$push(i)$** excesul din nodul j (singurul pentru care excesul poate crește) va fi $e_j + \min(e_i, r_{ij}, 2^p - e_j) \leq e_j + 2^p - e_j \leq 2^p$, deci condiția (b) rămâne îndeplinită.

- etapa p este încheiată când lista $L(p)$ devine vidă.

Pentru a determina în mod eficient un arc admisibil pentru o pompare (**push**), sau pentru a inspecta toate arcele care părăsesc un nod i în vederea reetichetării (**relabel**), vom organiza liste de adiacență $A(i)$ după cum urmează:

- fiecare element din listă conține: nodul j , x_{ij} , r_{ij} , un pointer către elementul corespunzător lui i din lista de adiacență $A(j)$ și un pointer către următorul element din lista $A(i)$.
- lista are asociat un iterator pentru a-i îngesa parcurgerea.

Toate aceste liste sunt construite în $O(m)$, înaintea apelului procedurii **initialization**.

```

initialization;  $K \leftarrow \lceil \log_2 U \rceil$ ;  $\Delta \leftarrow 2^{K+1}$ ;
for ( $p = \overline{K, 0}$ ) do
    construiește  $L(p)$ ;  $\Delta \leftarrow \Delta/2$ ;
    while ( $L(p) \neq \emptyset$ ) do
        fie  $i$  primul nod din  $L(p)$ ;
        caută în  $A(i)$  primul arc admisibil sau până se ajunge la sfârșit;
        if ( $ij$  este arcul admisibil găsit) then
             $\delta \leftarrow \min(e_i, r_{ij}, \Delta - e_j)$ ;
             $e_i \leftarrow e_i - \delta$ ;  $e_j \leftarrow e_j + \delta$ ;
            "trimite"  $\delta$  unități de flux de la  $i$  la  $j$ ;
            if ( $e_i \leq \Delta/2$ ) then
                șterge  $i$  din  $L(p)$ ;
            if ( $e_j > \Delta/2$ ) then
                adaugă  $j$  la începutul listei  $L(p)$ ;
        else
            calculează  $d[i] = \min\{d[j] + 1 : ij \in A(i) \text{ și } r_{ij} > 0\}$ 
            repoziționează  $i$  în  $L(p)$ ;
            setează poziția curentă (a pointerului) la începutul listei  $A(i)$ ;

```

Prefluxuri - Algoritmul lui Ahuja & Orlin

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 este dominată de **pompările nesaturate** (ceea ce rămâne este de complexitate $\mathcal{O}(nm)$).

Lema 5

Numărul pompărilor nesaturate este cel mult $8n^2$ în fiecare etapă a scalării, astfel numărul total este $\mathcal{O}(n^2 \log U)$.

Demonstrație. Fie

$$F(p) = \sum_{i \in V, i \neq s, t} \frac{e_i \cdot d(i)}{2^p}.$$

La începutul etapei p , $F(p) < \sum_{i \in V} \frac{2^p \cdot (2n)}{2^p} = 2n^2$.

Dacă, în etapa p , se execută $\text{relabel}(i)$, atunci nu există arce admisibile ij , și $d(i)$ crește cu $h \geq 1$ unități. $F(p)$ va crește cu cel mult h . Deoarece, $\forall i, d(i) < 2n$ urmează că $F(p)$ va crește (până la finalul etapei p) cel mult până la $4n^2$.

Dacă, în etapa p , se execută $\text{push}(i)$, atunci aceasta trimite $\delta \geq 2^{p-1}$ pe arcul admisibil ij cu $r_{ij} > 0$ și $d(i) = d(j) + 1$. Astfel, după push , $F(p)$ va avea valoarea $F'(p) = F(p) - \frac{\delta \cdot d(i)}{2^p} + \frac{\delta \cdot d(j)}{2^p} = F(p) - \frac{\delta}{2^p} \leq F(p) - \frac{2^{p-1}}{2^p} = F(p) - 1/2$.

Această descreștere nu poate apărea de mai mult de $8n^2$ ori (deoarece $F(p)$ poate crește cel mult până la $4n^2$ și $F(p)$ este nenegativ). Evident, numărul pompărilor nesaturate este dominat de acest număr de descreșteri ale lui $F(p)$.

Prefluxuri - Algoritmul lui Ahuja & Orlin

În concluzie:

Teorema

(Ahuja-Orlin, 1988) Algoritm de tip preflux cu scalarea exceselor are complexitatea timp $\mathcal{O}(nm + n^2 \log U)$.

A. Determinarea unui cuplaj de cardinal maxim și a unei mulțimi stable de cardinal maxim într-un graf bipartit.

Fie $G = (V_1, V_2; E)$ un graf bipartit cu n noduri și m muchii.

Considerăm rețeaua $R = (G_1, s, t, c)$, unde

- $V(G_1) = \{s, t\} \cup V_1 \cup V_2$;
- $E(G_1) = E_1 \cup E_2 \cup E_3$, cu

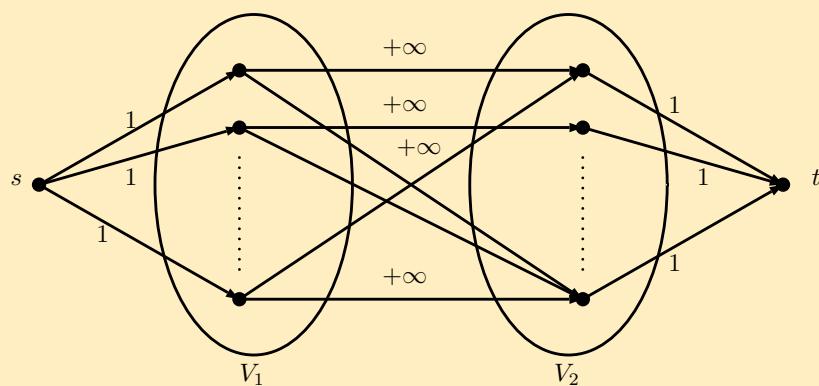
$$E_1 = \{sv_1 : v_1 \in V_1\}, E_2 = \{v_2t : v_2 \in V_2\},$$

$$E_3 = \{v_1v_2 : v_1 \in V_1, v_2 \in V_2, v_1v_2 \in E(G)\},$$

- $c : E(G_1) \rightarrow \mathbb{N}$ definită prin

$$c(e) = \begin{cases} 1, & \text{dacă } e \in E_1 \cup E_2 \\ \infty, & \text{dacă } e \in E_3 \end{cases}$$

Aplicații combinatoriale - Cuplaje în grafuri bipartite



Dacă $x = (x_{ij})$ este un flux întreg în R , atunci multimea $\{ij : i \in V_1, j \in V_2 \text{ și } x_{ij} = 1\}$ corespunde unui cuplaj M^x în graful bipartit G , cu $|M^x| = v(x)$.

Reciproc, orice cuplaj $M \in \mathcal{M}_G$ corespunde unei mulțimi de arce neadiacente din G_1 ; dacă pe fiecare astfel de arc ij ($i \in V_1, j \in V_2$) considerăm $x_{ij}^M = 1$ și $x_{si}^M = x_{jt}^M = 1$, și adăugăm $x^M(e) = 0$ pe restul arcelor, atunci fluxul întreg x^M satisfacă $v(x^M) = |M|$.³⁴⁷

Astfel, dacă rezolvăm problema fluxului maxim în R (începând cu fluxul nul), atunci obținem în $\mathcal{O}(nm + n^2 \log n)$ un cuplaj de cardinal maxim în G . (Constanta care înlocuiește în practică $+\infty$ trebuie să fie mai mare decât cardinalul oricărei tăieturi în graful suport, de exemplu poate fi $n^2 + 1$.)

Fie (S, T) secțiunea de capacitate minimă (obținută în $\mathcal{O}(m)$ dintr-un flux maxim determinat). Din Teorema de flux maxim-secțiune minimă, $c(S, T) = \nu(G)$.

Deoarece $\nu(G) < \infty$, luând $S_i = S \cap V_i$ și $T_i = T \cap V_i$ ($i = \overline{1, 2}$), avem $|T_1| + |S_2| = \nu(G)$ și $X = S_1 \cup T_2$ este o mulțime stabilă în G (pentru a avea $c(S, T) < \infty$). Mai mult, $|X| = |V_1 \setminus T_1| + |V_2 \setminus S_2| = n - \nu(G)$. Urmează că X este o mulțime stabilă de cardinal maxim, deoarece $n - \nu(G) = \alpha(G)$ (din teorema lui König).

Aplicații combinatoriale - Recunoașterea secvențelor digrafice

B. Recunoașterea secvențelor digrafice

Considerăm următoarea problemă:

Date $(d_i^+)_i=\overline{1,n}$ și $(d_i^-)_i=\overline{1,n}$, există un digraf $G = (\{1, \dots, n\}, E)$ astfel încât $d_G^+(i) = d_i^+$ și $d_G^-(i) = d_i^-$, $\forall i = \overline{1, n}$?

Condiții evident necesare pentru un răspuns afirmativ sunt:

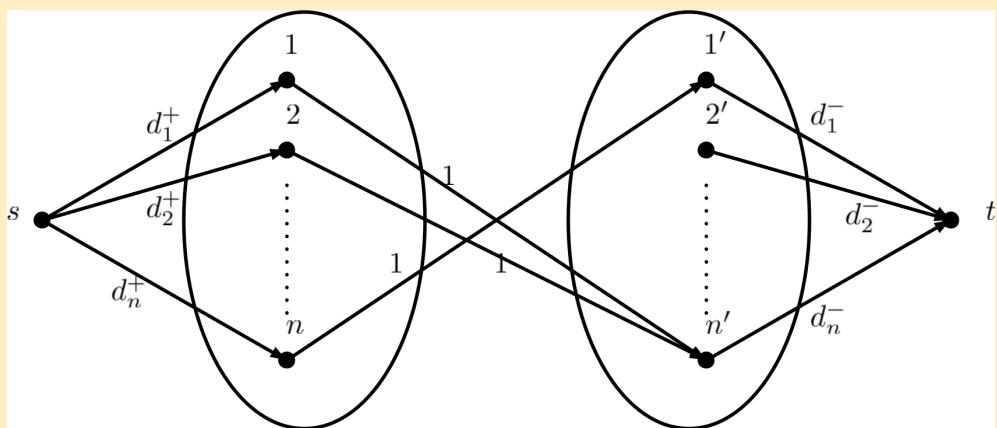
$$d_i^+ \in \mathbb{N}, 0 \leq d_i^+ \leq n - 1 \text{ și } d_i^- \in \mathbb{N}, 0 \leq d_i^- \leq n - 1, \forall i = \overline{1, n};$$

$$\sum_{i=1}^n d_i^+ = \sum_{i=1}^n d_i^- = m \text{ (unde } m = |E|).$$

În aceste ipoteze considerăm rețeaua bipartită $R = (G_1, s, t, c)$.

G_1 este obținut din graful complet bipartit $K_{n,n}$ cu bipartiția $(\{1, 2, \dots, n\}, \{1', 2', \dots, n'\})$, prin stergerea mulțimii de muchii $\{11', 22', \dots, nn'\}$ și prin orientarea fiecărei muchii ij' ($\forall i \neq j \in \{1, 2, \dots, n\}$) de la i la j' , și prin adăugarea a două noi noduri s, t , și a tuturor arcelor si , $i \in \{1, 2, \dots, n\}$ și $j't$, $j \in \{1, 2, \dots, n\}$.

Funcția de capacitate: $c(si) = d_i^+$, $c(j't) = d_j^-$, $c(ij') = 1$, $\forall i, j = \overline{1, n}$.



Aplicații combinatoriale - Recunoașterea secvențelor digrafice

Dacă în R există un flux întreg, x , de valoare maximă m , atunci din fiecare nod i vor pleca exact d_{ij}^+ arce, ij' , pe care $x_{ij'} = 1$, și în fiecare nod j' vor intra exact d_i^- arce, ij' , pe care $x_{ij'} = 1$.

Digraful dorit, G , este construit luând $V(G) = \{1, 2, \dots, n\}$ și punând $ij \in E(G)$ dacă și numai dacă $x_{ij'} = 1$.

Reciproc, dacă G există, atunci inversând construcția anterioară vom obține un flux întreg în R de valoare m (deci de valoare maximă).

Urmează că, recunoașterea secvențelor digrafice (și construirea digrafului, în cazul unui răspuns afirmativ) poate fi făcută în $\mathcal{O}(nm + n^2 \log n) = \mathcal{O}(n^3)$.

C. Determinarea numărului de conexiune pe muchii a unui graf

Fie $G = (V, E)$ un graf. Pentru $s, t \in V$, $s \neq t$, notăm

- $p_e(s, t) =$ numărul maxim de drumuri disjuncte pe muchii de la s la t în G ,
- $c_e(s, t) =$ cardinalul minim al unei multimi de muchii astfel încât nu există drum de la s la t în graful obținut prin stergerea ei din G .

Teoremă

$$p_e(s, t) = c_e(s, t).$$

Demonstrație. Fie G_1 digraful obținut din G prin înlocuirea fiecărei muchii printr-o pereche de arce simetrice. Fie $c : E(G_1) \rightarrow \mathbb{N}$ o funcție de capacitate definită prin $c(e) = 1$, $\forall e \in E(G_1)$.

Aplicații combinatoriale - Conexiunea pe muchii

Demonstrație (continuare). Fie x^0 un flux întreg de valoare maximă în $R = (G_1, s, t, c)$. Dacă există un circuit C în G_1 cu $x_{ij}^0 = 1$ pe toate arcele lui C , atunci putem pune fluxul 0 pe toate arcele lui C fără a modifica valoarea fluxului x_0 . Astfel, putem presupune că fluxul x^0 este aciclic și atunci x^0 poate fi scris ca o sumă de $v(x^0)$ fluxuri întregi x^k fiecare cu $v(x^k) = 1$.

Fiecare flux x^k corespunde unui drum de la s la t în G_1 (considerând arcele pe care fluxul nu este 0), care este un drum de la s la t și în graful G .

Urmează că $v(x^0) = p_e(s, t)$, deoarece orice multime de drumuri disjuncte pe muchii de la s la t în G generează un flux 0 – 1 în R de valoare egală cu numărul acestor drumuri.

Demonstrație (continuare). Fie (S, T) o secțiune de capacitate minimă în R ; avem $c(S, T) = v(x^0)$, din Teorema flux maxim-secțiune minimă. Pe de altă parte, $c(S, T)$ este numărul de arce cu o extremitate în S și cealaltă în T (deoarece toate capacitatele arcelor sunt 1). Această mulțime de arce generează în G o mulțime de muchii de același cardinal astfel încât nu există vreun drum de la s la t în graful obținut prin ștergerea ei din G .

Astfel am obținut o mulțime de $c(S, T) = v(x^0) = p_e(s, t)$ muchii în G care deconectează pe s de t prin ștergerea lor din G . Urmează că $c_e(s, t) \leq p_e(s, t)$. Deoarece inegalitatea $c_e(s, t) \geq p_e(s, t)$ este evidentă, teorema este demonstrată. \square

Dacă G este un graf conex, $\lambda(G)$, valoarea maximă alui $p \in \mathbb{N}$ pentru care G este p -muchie-conex, este

$$\min_{s, t \in V(G), s \neq t, st \notin E(G)} c_e(s, t) (*)$$

Urmează că, pentru a calcula $\lambda(G)$, este necesar să rezolvăm $n(n-1)/2$ probleme de flux maxim descrise mai sus. Acest număr poate fi redus dacă observăm că, pentru o pereche fixată (s, t) , dacă (S, T) este o secțiune de capacitate minimă, atunci

$$\forall v \in S \text{ și } \forall w \in T \text{ } c_e(v, w) \leq c(S, T) (**)$$

În particular, dacă (s, t) este perechea pentru care minimul din $(*)$ se atinge, avem egalitate în $(**)$.

Dacă fixăm un nod $s_0 \in V$ și rezolvăm cele $n-1$ probleme de flux maxim luând $t_0 \in V \setminus \{s_0\}$ vom obține o pereche (s_0, t_0) cu $c(s_0, t_0) = \lambda(G)$ (t_0 nu va fi în aceeași clasă a bipartiției cu s_0 în (S, T)).

Concluzie: $\lambda(G)$ poate fi găsit în $\mathcal{O}(n \cdot (nm + n^2c)) = \mathcal{O}(n^2m)$.

D. Determinarea numărului de conexiune pe noduri a unui graf.

Fie $G = (V, E)$ un graf. C $s, t \in V$, $s \neq t$, dacă notăm

- $p(s, t) =$ numărul maxim de drumuri intern disjuncte (pe noduri) de la s la t în G ,
 - $c(s, t) =$ cardinalul minim al unei mulțimi st -separatoare de noduri din G ,

atunci, din teorema lui Menger, avem

$$p(s, t) = c(s, t) (***)$$

În plus, numărul de conexiune pe noduri, $k(G)$, al grafului G (valoarea maximă a lui $p \in \mathbb{N}$ pentru care G este p -conex) este

$$k(G) = \begin{cases} n - 1, & \text{dacă } G = K_n \\ \min_{s,t \in V(G), s \neq t} c(s,t), & \text{dacă } G \neq K_n \end{cases} \quad (***)$$

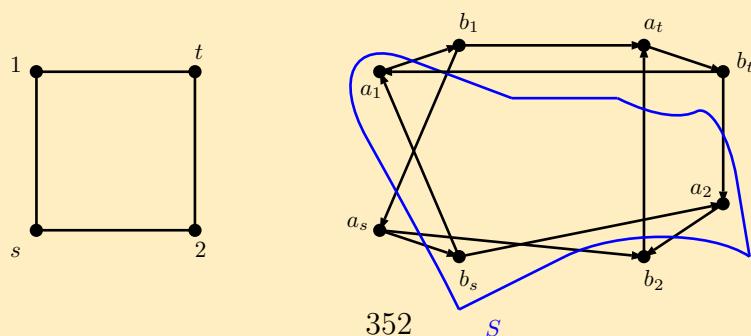
Aplicații combinatoriale - Conexiunea pe noduri

Arătăm că egalitatea (***) provine din Teorema flux maxim-secțiune minimă, aplicată unei rețele corespunzătoare.

Fie $G_1 = (V(G_1), E(G_1))$ digraful construit pornind de la G astfel:

- $\forall v \in V$, adăugăm $a_v, b_v \in V(G_1)$ și $a_v b_v \in E(G_1)$;
 - $\forall vw \in E$, adăugăm $b_v a_w, b_w a_v \in E(G_1)$.

Exemplu



Mai definim $c : E(G_1) \rightarrow \mathbb{N}$ prin

$$c(e) = \begin{cases} 1, & \text{dacă } e = a_v b_v \\ \infty, & \text{altfel} \end{cases}$$

Considerăm rețeaua $R = (G_1, b_s, a_t, c)$.

Fie x^0 un flux întreg în R de valoare maximă. În nodurile $b_v (v \in V)$ intră exact câte un arc de capacitate 1 și din nodurile $a_v (v \in V)$ pleacă exact câte un arc de capacitate 1.

Urmează că (din legea de conservare a fluxului) că $x_{ij}^0 \in \{0, 1\}$, $\forall ij \in E(G_1)$. Astfel x^0 poate fi descompus în $v(x^0)$ fluxuri x^k , fiecare de valoare 1, cu proprietatea că arcele pe care x_k este nenul corespund la $v(x^0)$ drumuri intern disjuncte din G .

Pe de altă parte, din orice mulțime de p st -drumuri intern disjuncte în G , putem construi p $b_s a_t$ -drumuri intern disjuncte în G_1 , pe care se poate transporta o singură unitate de flux. Urmează că $v(x^0) = p(s, t)$.

Aplicații combinatoriale - Conexiunea pe noduri

Fie (S, T) o secțiune de capacitate minimă în R astfel încât $v(x^0) = c(S, T)$. Deoarece $v(x^0) < \infty$, urmează că $\forall i \in S, \forall j \in T$ cu $ij \in E(G_1)$; avem $c(ij) < \infty$, deci $c(ij) = 1$, adică $\exists u \in V$ astfel încât $i = a_u$ și $j = b_u$.

Astfel, secțiunea (S, T) corespunde unei mulțimi de noduri $A_0 \subseteq V$ astfel încât $c(S, T) = |A_0|$ și A_0 este o mulțime st -separatoare.

Pe de altă parte, pentru orice mulțime st -separatoare, A , avem $|A| \geq p(s, t) = v(x^0)$. Deci

$$c(s, t) = |A_0| = c(S, T) = v(x^0) = p(s, t).$$

Demonstrația de mai sus arată că, pentru a determina $k(G)$ este suficient să determinăm minimul în (***) rezolvând $|E(\overline{G})|$ probleme de flux maxim, unde \overline{G} este complementul lui G .

Avem astfel un algoritm cu complexitatea timp

$$\mathcal{O} \left(\left(\frac{n(n-1)}{2} - m \right) (nm + n^2 \log n) \right).$$

O observație simplă ne oferă un algoritm mai eficient. Evident,

$$k(G) \leq \min_{v \in V} d_G(v) = \frac{1}{n} \left(n \cdot \min_{v \in V} d_G(v) \right) \leq \frac{1}{n} \sum_{v \in V} d_G(v) = \frac{2m}{n}.$$

Dacă A_0 este a mulțime separatoare în G cu $|A_0| = k(G)$, atunci $G \setminus A_0$ nu este conex și există o partiție a $V \setminus A_0$ (V' , V'') astfel încât nu există muchii de la V' la V'' și $\forall v' \in V'$, $\forall v'' \in V''$ avem $p(v', v'') = k(G)$.

Aplicații combinatoriale - Conexiunea pe noduri

Urmează că rezolvând o problemă de flux maxim cu $s_0 \in V'$ și $t_0 \in V''$ obținem că $p(s_0, t_0) =$ valoarea fluxului maxim = $k(G)$.

Putem determina o astfel de pereche după cum urmează: fie $l = \left\lceil \frac{2m}{n} \right\rceil + 1$, alegem l noduri arbitrar din $V(G)$, și pentru fiecare astfel de nod, v , rezolvăm toate problemele de flux maxim $p(v, w)$, cu $vw \notin E$. Numărul acestor probleme este $\mathcal{O}(nl) = \mathcal{O}(n((\frac{2m}{n} + 1))) = \mathcal{O}(m)$.

Astfel complexitatea timp a determinării lui $k(G)$ este $\mathcal{O}(m(nm + n^2 \log n))$.

Exerciții pentru seminarul din săptămâna viitoare

[Jump to Table of contents](#)

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

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

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

Exercițiu 1. Arătați că, utilizând un algoritm de flux maxim (într-o anumită rețea), se poate găsi, într-o matrice $0 - 1$, o mulțime de cardinal maxim de elemente egale cu 0, în care oricare două elemente nu se află pe aceeași linie sau pe aceeași coloană.

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

Exercițiu 2. Fie S și T mulțimi nevide, disjuncte și finite. Se dă o funcție $a : S \cup T \rightarrow \mathbb{N}$. Să se decidă dacă există un graf bipartit $G = (S, T; E)$ astfel încât $d_G(v) = a(v)$, pentru orice $v \in S \cup T$; dacă răspunsul este afirmativ trebuie returnate muchiile lui G (S și T sunt clasele bipartiției lui G). Arătați că această problemă poate fi rezolvată în timp polinomial ca o problemă de flux maxim într-o anumită rețea.

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 din săptămâna viitoare

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

Exercițiu 3. Fiecare student dintr-o mulțime S de cardinal $n > 0$ subscrive la o submulțime de 4 cursuri opționale dintr-o mulțime C de cardinal $k > 4$. Descrieți un algoritm (de complexitate timp polinomială) care să determine (dacă există) o alocare a studenților către cursurile opțional din C astfel încât fiecare student să fie asignat la exact 3 cursuri (din cele 4 la care a subscris) și fiecare curs să aibă asignați cel mult $\lceil n/k \rceil$ studenți.

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

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

Exercițiu 4.

- Adevărat sau fals? Într-o rețea $R = (G, s, t, c)$ cu capacitați distincte există un singur flux maxim. De ce?
- Descrieți și demonstrați corectitudinea unui algoritm de complexitate (timp) polinomială care să decidă dacă într-o rețea dată există un singur flux maxim.

Exercițiu 7 (continuare). O tăietură minimă în G este o tăietură (S_0, T_0) astfel încât

$$c(S_0, T_0) = \min_{(S, T) \text{ tăietură în } G} c(S, T)$$

- (a) Arătați că se poate determina o tăietură minimă în timp polinomial rezolvând un număr polinomial de probleme de flux maxim în anumite rețele.
- (b) Pentru $G = C_n$ (circuit inducător de ordin $n \geq 3$) cu toate capacitatele 1, arătați că există $\frac{n(n-1)}{2}$ tăieturi minime.

Exerciții pentru seminarul din săptămâna viitoare

Exercițiu 8. Fie $G = (V; E)$ un digraf și $w : V \rightarrow \mathbb{R}$ astfel că $w(V) \cap \mathbb{R}_+$ și $w(V) \cap \mathbb{R}_- \neq \emptyset$. O submulțime $A \subseteq V$ se numește izolată în G dacă nu există nici un arc care iese din A . Ponderea unei submulțimi $A \subseteq V$ este $w(A) = \sum_{v \in A} w(v)$. Descrieți un algoritm de complexitate polinomială bazat pe un algoritm de flux maxim într-o anumită rețea care să determine o submulțime izolată de pondere maximă în G .

Exercițiu 9. La Departamentul de Informatică există p studenți ($S = \{S_1, S_2, \dots, S_p\}$) care doresc să absolve cu o diplomă de licență și k profesori ($\mathbb{P} = \{P_1, P_2, \dots, P_k\}$). Lucrările de licență ale studenților sunt evaluate de echipe formate din câte r profesori.

Pentru subiectul unei lucrări de licență un profesor poate specializat sau nu; se cunoaște mulțimea, $\mathbb{P}_i \subsetneq \mathbb{P}$ ($\mathbb{P}_i \neq \emptyset$), a profesorilor competenți în a judeca lucrarea de licență a studentului S_i , pentru orice i . Fiecare profesor P_l poate participa în cel mult n_l astfel de echipe de evaluare.

Exercițiu 9 (cont.) Fiecare student trebuie să-și prezinte lucrarea unei echipe de $r (\leq k)$ profesori, $a (\leq r)$ fiind specializați în proiectul respectiv, iar $(r - a)$ nu.

- Descrieți un model cu o rețea de transport pentru a organiza ca mai sus echipele de evaluare formate din profesori (fiecare profesor trebuie asignat unei mulțimi de lucrări de licență).
- Caracterizați existența unei soluții pentru această problemă în termenii existenței unui anumit flux maxim în rețeaua de mai sus. (Caracterizarea trebuie demonstrată!)
- Care este complexitatea timp necesară pentru a decide dacă există soluții?

Exerciții rezolvate (partial)

Exercițiu 1. Soluție.

- Mai întâi se reduce problema la determinarea unui cuplaj maxim într-un anume graf bipartit.
- Fie $A = (a_{ij})_{\substack{1 \leq i \leq n \\ 1 \leq j \leq m}}$, definim $G = (S, T; E)$ astfel

$$S = \{v_1, v_2, \dots, v_n\}, T = \{u_1, u_2, \dots, u_m\}, E = \{v_i u_j : a_{ij} = 1\}.$$

- Fie $M \subseteq E(G)$ și $P = \{a_{ij} : v_i u_j \in M\}$; $M \in \mathcal{M}_G$ dacă și numai dacă orice două elemente din P aparțin la linii și coloane diferite (de ce?).
- Cum $|P| = |M|$, un cuplaj maxim corespunde unei mulțimi de cardinal maxim cu proprietățile de mai sus,
- Putem folosi un algoritm de flux maxim pentru a determina un cuplaj maxim într-un graf bipartit (cum?).

Exercițiu 2. Soluție.

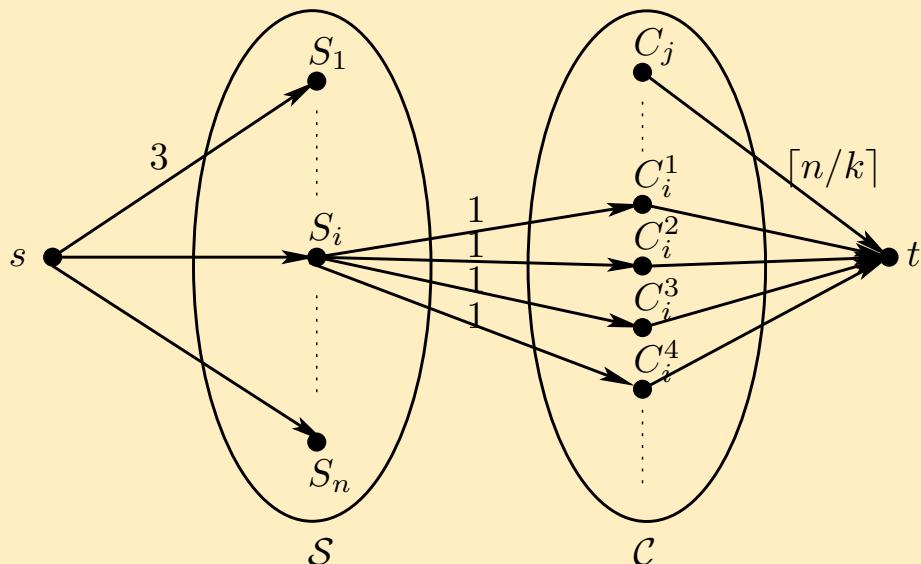
- Putem reformula problema ca una de recunoaștere a unei secvențe digrafice.
- Există un digraf G' cu $V(G') = S \cup T$, și $d_{G'}^-(v) = d^-(v)$ și $d_{G'}^+(v) = d^+(v)$, pentru orice $v \in V$, unde

$$d_{G'}^-(v) = \begin{cases} a(v), & v \in T \\ 0, & v \in S \end{cases}, \quad d_{G'}^+(v) = \begin{cases} a(v), & v \in S \\ 0, & v \in T \end{cases}$$

- Apoi se poate utiliza un algoritm de flux maxim într-o anumită rețea (cum?).

Exerciții rezolvate (partial)

Exercițiu 3. Soluție.



Exerciții rezolvate (partial)

[Jump to Table of contents](#)

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

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

- Presupunem că mulțimea studenților este $S = \{S_1, S_2, \dots, S_n\}$, cea a cursurilor $C = \{C_1, C_2, \dots, C_k\}$ iar fiecare student S_i a ales $\{C_i^1, C_i^2, C_i^3, C_i^4\} \subset C$, for $i = \overline{1, n}$.
- Definim o rețea de transport: $R = (G, s, t, c)$, unde $c : E(G) \rightarrow \mathbb{R}_+$ și

$$V = V(G) = \{s, t\} \cup \Sigma \cup C$$

$$\begin{aligned} E = E(G) = & \{sS_i : i = \overline{1, n}\} \cup \{S_i C_i^l : i = \overline{1, n}, l = \overline{1, 4}\} \\ & \cup \{C_j t : j = \overline{1, k}\} \end{aligned}$$

$$c(sS_i) = 3, \forall i = \overline{1, n}; \quad c(S_i C_i^l) = 1, \forall i = \overline{1, n}, l = \overline{1, 4};$$

$$c(C_j t) = \lceil n/k \rceil, \forall j = \overline{1, k}.$$

Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții rezolvate (partial)

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

- Există o alocare a cursurilor/studenților cu proprietățile cerute dacă și numai dacă valoarea maximă a unui flux în R este $3n$ (altfel spus toate arcele care părăsesc nodul s sunt saturate - **de ce?**).
- “ \implies ” Dacă avem o alocare astfel ca orice student S_i participă la 3 cursuri C_{i1}, C_{i2}, C_{i3} , iar fiecare curs C_j adună $m_j \leq \lceil n/k \rceil$ studenți, atunci fie $x : E \rightarrow \mathbb{R}_+$:

$$x(sS_i) = 3, \forall i = \overline{1, n}; \quad x(S_i C_{il}) = 1, \forall i = \overline{1, n}, l = \overline{1, 3};$$

$$x(C_j t) = m_j, \forall j = \overline{1, k}.$$

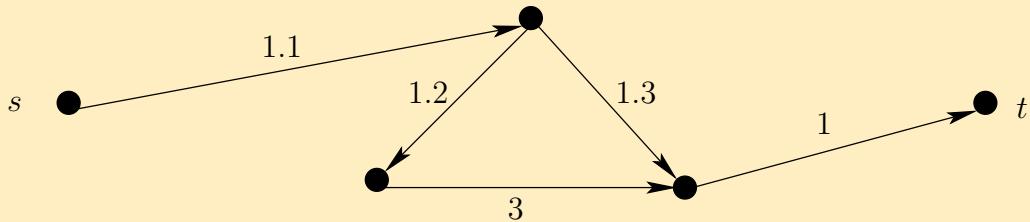
- Evident x este un flux maxim în R de valoare $3n$ (**de ce?**).
- “ \iff ” ... cum?

Graph Algorithms * C. Croitoru - Graph Algo360ns * C. Croitoru - Graph Algorithms *

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

Exercițiu 4. Soluție.

(a)



Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
 C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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

(b) Fie x un flux de valoare maximă în rețeaua R ; arătăm că x este singurul flux de valoare maximă dacă și numai dacă nu există A -drumuri închise relativ la x .

- " \Rightarrow " Să presupunem că există un A -drum P închis relativ la x ($\delta(P) > 0$). Definim următoarea funcție $y : E(G) \rightarrow \mathbb{R}_+$ prin

$$y(e) = \begin{cases} x(e), & \text{if } e \notin E(P), \\ x(e) - \delta(P), & \text{if } e \in E(P), \text{ este arc invers,} \\ x(e) + \delta(P), & \text{if } e \in E(P), \text{ este arc direct,} \end{cases}$$

y ($\neq x$) este deasemeni flux maxim în R (de ce?) - contradicție.

- " \Leftarrow " Să presupunem prin absurd că există două fluxuri maxime distincte în R , x și y . Următorul algoritm oferă un A -drum închis în R :

```
for( $v \in V(G)$ ) {
    mark[ $v$ ], before[ $v$ ]  $\leftarrow 0$ ;
let  $e = uv \in E(G)$  s. t.  $x(uv) \neq y(uv)$ ;
mark[ $u$ ], mark[ $v$ ]  $\leftarrow 1$ ;
 $z \leftarrow v$ ; before[ $v$ ]  $\leftarrow u$ ; orientation  $\leftarrow direct$ ;
while(true) {
    if(orientation = direct) {
        if( $\exists e' = wz \in E(G)$  s. t. sign( $e$ )  $\neq$  sign( $e'$ ))
            orientation  $\leftarrow inverse$ ;
        else let  $e' = zw \in E(G)$  s. t. sign( $e$ ) = sign( $e'$ );
        if(mark[ $w$ ] = 1)
            return;
        mark[ $w$ ]  $\leftarrow 1$ ; before[ $w$ ]  $\leftarrow z$ ;  $z \leftarrow w$ ,  $e \leftarrow e'$ ;
    }
}
```

Exerciții rezolvate (partial)

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

```
if(orientation = inverse) {
    if( $\exists e' = zw \in E(G)$  s. t. sign( $e$ )  $\neq$  sign( $e'$ ))
        orientation  $\leftarrow direct$ ;
    else let  $\exists e' = wz \in E(G)$  s. t. sign( $e$ ) = sign( $e'$ );
    if(mark[ $w$ ] = 1)
        return;
    mark[ $w$ ]  $\leftarrow 1$ ; before[ $w$ ]  $\leftarrow z$ ;  $z \leftarrow w$ ,  $e \leftarrow e'$ ;
}
}

} // for while
```

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algo362ns * C. Croitoru - Graph Algorithms *

- Existența arcului e' este garantată de relațiile (de ce?):

$$\sum_{ij \in E(G)} x(ij) - \sum_{ji \in E(G)} x(ji) = \sum_{ij \in E(G)} y(ij) - \sum_{ji \in E(G)} y(ji), \forall i \in V(G),$$

- Bucla *while* de mai sus se încheie prin detectarea un nod deja marcat, i.e. când găsim un drum închis, P , care are proprietatea:

$$x(e) \begin{cases} > 0, & \text{dacă } e \text{ este un arc direct,} \\ < c(e), & \text{dacă } e \text{ este un arc invers} \end{cases}.$$

- Dacă P nu este un A -drum închis relativ la x , atunci parcurgându-l invers, obținem un A -drum închis relativ la x - ceea ce e o contradicție (de ce?).
- Putem verifica în $\mathcal{O}(n + m)$ dacă există un A -drum închis relativ la un flux maxim x folosind o strategie *bfs* asemănătoare celei din algoritmul Edmonds-Karp.

Exerciții rezolvate (partial)

Exercițiul 5. Soluție.

- Fie $R_k = (G, s, t, c_k)$ o rețea (pentru k săptămâni) unde

$$V(G) = \{s, t\} \cup \{P_1, P_2, \dots, P_n, L_1, L_2, \dots, L_m\},$$

$$E(G) = \{sP_i : 1 \leq i \leq n\} \cup \{L_j t : 1 \leq j \leq m\} \cup$$

$$\{P_i L_j : L_j \in \mathcal{L}_i, 1 \leq i \leq n\},$$

$$c_k(sP_i) = ks_i, i = \overline{1, n}, c_k(P_i L_j) = \infty, \text{ for } L_j \in \mathcal{L}_i,$$

$$c_k(L_j t) = 1, j = \overline{1, m}$$

- Valoarea unui flux maxim este cel mult $\sum_{j=1}^m c_k(L_j t) = m$ (de ce?), adică valoarea necesară pentru încheia toate proiectele. Astfel, numărul minim de săptămâni este $\min\{k : v(\varphi) = m, \varphi \text{ flux maxim in } R_k\}$ (de ce?)³⁶³

- Acest minim poate fi găsit folosind o procedură de căutare binară (notăm cu $v(R_k)$ valoarea maximă a unui flux în R_k):

```

min ← 0;
max ← ? ;
while (min < max - 1){
    current ← ?;
    if (v(Rcurrent) < m)
        min ← current;
    else
        max ← current;
}
if (v(Rmin) == m)
    return min;
else
    return max;

```

Exerciții rezolvate (partial)

Exercițiul 7. Soluție.

- (a) Fără a restrânge generalitatea putem presupune că $G \equiv K_n$ (adăugăm muchiile lipsă și punem capacitatea nule).
- Fie s și t două noduri noi ($s, t \notin V$) iar H următorul digraf

$$V(H) = V(G) \cup \{s, t\},$$

$$E(H) = \{uv, vu : uv \in E(G)\} \cup \{su, ut : u \in V(G)\}.$$

- Fie $\alpha > |E| \cdot \max_{e \in E} c(e) + 1$ mai mare decât capacitatea oricărei căi din G .
- Definim rețeaua $R_{ij} = (H, s, t, c_{ij})$, unde

$$c_{ij}(uv) = c(uv), \forall uv \in E(G),$$

$$c_{ij}(sv) = c_{ij}(vt) = \alpha, \forall v \in V(H) \setminus \{v_i, v_j\} \text{ și } c(sv_i) = c(v_jt) = 0$$

- Fie (S', T') o secțiune de capacitate minimă în rețeaua R_{ij} .
 - Notăm $S = S' \setminus \{s\}$ și $T = T' \setminus \{t\}$, atunci S și T sunt amândouă vide:
 - Presupunem prin reducere la absurd că $S = \emptyset$, atunci $c_{ij}(S', T') = (n-1)\alpha$, dar $c_{ij}(S' \cup \{v_j\}, T' \setminus \{v_j\}) = (n-2)\alpha + c(\{v_j\}, V \setminus \{v_j\}) < c(S', T')$ - contradicție (de ce?).
 - Similar se poate arăta că $T \neq \emptyset$.
 - $c_{ij}(S', T') = c(S, T) + (n-2)\alpha + a_{ij}(S, T)$ unde

$$a_{ij}(S, T) = \begin{cases} 2\alpha, & \text{if } |S \cap \{v_j\}| + |T \cap \{v_i\}| = 0, \\ \alpha, & \text{if } |S \cap \{v_j\}| + |T \cap \{v_i\}| = 1, \\ 0, & \text{if } |S \cap \{v_j\}| + |T \cap \{v_i\}| = 2. \end{cases}$$

Exercitii rezolvate (partial)

- Deoarece $\alpha > 0$ vom avea $c_{ij}(S', T') = c(S, T) + (n - 2)\alpha$, astfel $v_j \in S$, și $v_i \in T$.
 - Fie (S_0, T_0) o tăietură de capacitate minimă în G ; se poate arăta că

$$(n-2)\alpha + c(S_0, T_0) = \min_{1 \leq i \neq j \leq n} \min_{(S', T')} \text{sectiune in } R_{ij} c_{ij}(S', T') \text{ (de ce?)}$$

- Astfel, pentru a determina o tăietură de capacitate minimă în G trebuie rezolvate $\frac{n(n - 1)}{2}$ probleme de flux maxim (de ce?).

Exercițiu 9. Soluție.

(a) Considerăm următoarea rețea, $R = (G, s, t, c)$, unde

$$V(G) = \mathcal{S} \cup \mathcal{P} \cup \{s, t\},$$

$$E(G) = \{sS_i : i = \overline{1, p}\} \cup \{S_iC_i : i = \overline{1, p}\} \cup \{S_iN_i : i = \overline{1, p}\} \cup$$

$$\cup \{C_iP_l : P_l \in \mathcal{P}_i\} \cup$$

$$\cup \{N_iP_l : P_l \in \mathcal{P} \setminus \mathcal{P}_i\} \cup \{P_lt : l = \overline{1, k}\},$$

$$c(sS_i) = r, c(S_iC_i) = a, c(S_iN_i) = r - a,$$

$$c(C_iP_j) = (N_iP_j) = 1, c(P_lt) = n_l$$

Exerciții rezolvate (partial)

(b) Există o soluție a problemei dacă și numai dacă fluxul maxim din această rețea are valoarea $p \cdot r$ (de ce?).

(c) $|G| = 3p+k+2$, $|E(G)| = 3p + \sum_{i=1}^p |\mathcal{P}_i| + \sum_{i=1}^p |\mathcal{P} \setminus \mathcal{P}_i| + k = pk + 3p + k$.

Capacitatea maximă a unei rețele în R este

$$U = \max \{r, \max_{1 \leq l \leq k} n_l\} \leq \max \{r, p\} \leq \max \{k, p\}.$$

- Complexitatea algoritmului Edmonds-Karp este: $\mathcal{O}((p+k) \cdot p^2 \cdot k^2)$.
- Complexitatea algoritmului Ahuja-Orlin: ?
- Complexitatea algoritmului Ford-Fulkerson: ?

10 CURS 10: Fluxuri de cost minim. Reduceri poliomiale pentru probleme de decizie pe grafuri.

graf hamiltonian - se verifica daca exista ciclu elementar care sa treaca prin toate nodurile o singura data

drum hamiltonian - se cauta drum, nu ciclu, care trece prin toate nodurile o singura data

Daca arcele sunt directe \implies aduni (capacitatea)

Daca arcele sunt inverse \implies scazi (capacitatea)

(di)graf trasabil = (di)graf care contine un drum hamiltonian

A – drum inchis :

- arc direct: flux < capacitate
- arc invers: flux > 0
- se calculeaza: + costul de pe arc direct - cost de pe arc invers

Algoritmica Grafurilor - Cursul 10

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

11 decembrie 2020

Cuprins

- 1 **Fluxuri în rețele**
 - Fluxuri de cost minim
- 2 **Reduceri în timp polinomial pentru probleme pe grafuri**
 - Multimii stabile de cardinal maxim
 - Colorări ale nodurilor
 - Probleme Hamiltoniene
- 3 **Exerciții pentru seminarul din săptămâna viitoare**
- 4 **Exerciții rezolvate (parțial)**

Să presupunem că în rețeaua $R = (G, s, t, c)$, se dă încă plus o funcție de cost: $a : E \rightarrow \mathbb{R}$; $\forall ij \in E$, $a(ij) = a_{ij}$ este costul arcului ij (interpretat ca fiind costul trimiterii unei "unități" de flux pe arcul ij).

Dacă x este un flux în R , atunci costul lui x este

$$a(x) = \sum_{i,j} a_{ij}x_{ij}.$$

Problema fluxului de cost minim

Date: R o rețea, $a : E \rightarrow \mathbb{R}$ o funcție de cost, și $v \in \mathbb{R}_+$,

Determină: un flux x^0 în R astfel încât

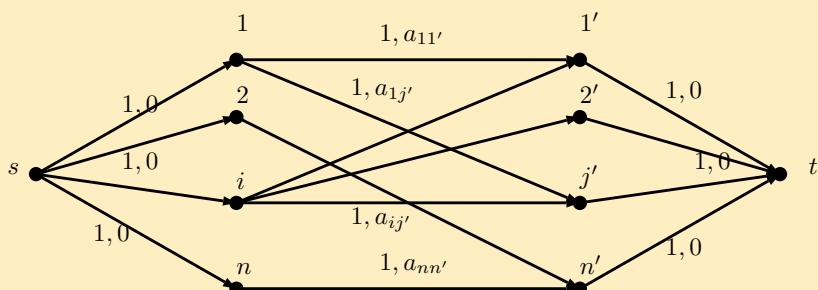
$$a(x^0) = \min \{a(x) : x \text{ flux în } R, v(x) = v\}.$$

Să observăm că, dacă v nu este mai mare decât valoarea fluxului maxim în R , atunci problema are întotdeauna soluții ($a(x)$ este o funcție liniară definită pe mulțimea nevidă și compactă din \mathbb{R}^{m+1} a tuturor fluxurilor de valoare v).

Fluxuri de cost minim - Exemple

1. Problema asignării. Se dau n muncitori și n slujbe. Costul repartizării muncitorului i slujbei j este a_{ij} . Să se asigneze fiecare muncitor către unei slujbe astfel încât costul total să fie minim.

Considerăm rețeaua bipartită de mai jos, unde fiecare arc este etichetat cu capacitatea sa urmată de cost. Astfel, $c_{ij'} = 1$, $c_{si} = 1$, $a_{si} = 0$, $c_{j't} = 1$, și $a_{j't} = 0$, $\forall i, j \in \{1, 2, \dots, n\}$.



Un flux de cost minim de valoare n constituie soluția problemei.

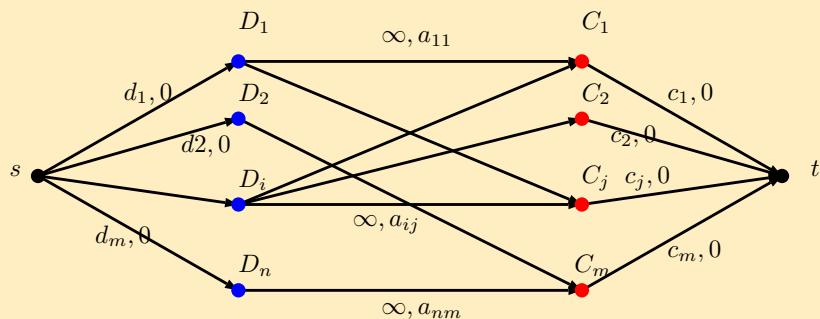
Similar putem determina un cuplaj perfect de pondere minimă într-un graf bipartit.

2. Problema de transport Hitchcock-Koopmans. Un produs, disponibil în depozitele D_1, \dots, D_n în cantitățile d_1, \dots, d_n , respectiv, este cerut decătre clienții C_1, \dots, C_m în cantitățile c_1, \dots, c_m , respectiv. Se cunosc costurile de transport unitar, a_{ij} - de la depozitul D_i la clientul C_j , $\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}$.

Să se determine o schemă de transport care să satisfacă cererea clientilor cu un cost total al transportului minim.

Fluxuri de cost minim - Exemplu

Problema are soluție numai dacă $\sum_{i=1}^n d_i \geq \sum_{j=1}^m c_j$. În acest caz, un flux de cost minim de valoare $v = \sum_{i=1}^m c_i$, în rețeaua de mai jos, rezolvă problema.



Definiție

Fie x un flux în $R = (G, s, t, c)$ și $a : E \rightarrow \mathbb{R}$ o funcție de cost.

- Dacă P este un A -drum în R relativ la x , atunci **costul drumului P** este definit ca

$$a(P) = \sum_{ij \in E(P), ij \text{ forward}} a_{ij} - \sum_{ij \in E(P), ji \text{ backward}} a_{ji}$$

- Dacă C este un A -drum închis în R relativ la x , atunci $a(C)$ este calculat ca mai sus, după stabilirea unui sens de parcursere alui C (este posibil ca ambele sensuri să ofere A -drumuri relativ la x).

Remarci

- Dacă P este un drum de creștere relativ la x , atunci $x^1 = x \otimes r(P)$ este un flux de valoare $v(x^1) = v(x) + r(P)$ și cost $a(x^1) = a(x) + r(P) \cdot a(P)$.

Fluxuri de cost minim

Remarci

- Dacă C este un A -drum închis în R relativ la x , atunci $x^1 = x \otimes r(C)$ este un flux de valoare $v(x^1) = v(x)$ și cost $a(x^1) = a(x) + r(C) \cdot a(C)$. Urmează că dacă $a(C) < 0$ atunci x^1 este un flux de aceeași valoare cu x dar de cost $a(x^1) < a(x)$.

Teorema 1

Un flux x de valoare v este un flux de cost minim dacă și numai dacă nu există un A -drum închis de cost negativ relativ la x în R .

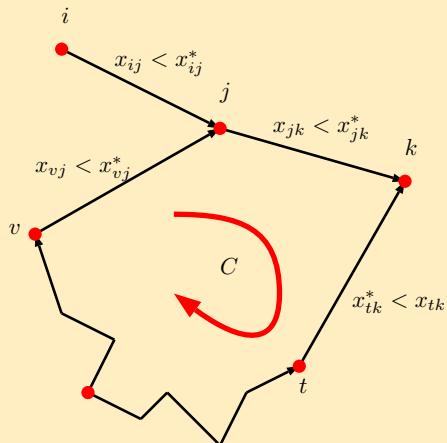
Demonstrație. " \Rightarrow " Din remarca de mai sus.

" \Leftarrow " Fie x un flux de valoare v astfel încât nu există un A -drum închis de cost negativ relativ la x în R . Fie x^* un flux de cost minim de valoare v astfel încât

$\Delta(x, x^*) = \min \{\Delta(x, x') : x'$ flux de cost minim de valoare $v\}$,
 unde $\Delta(x, x') = |\{ij \in E : x_{ij} \neq x'_{ij}\}|$.

Dacă $\Delta(x, x^*) = 0$, atunci $x = x^*$, astfel x este un flux de cost minim. Altfel, $\Delta(x, x^*) > 0$ și există ij astfel încât $x_{ij} \neq x_{ij}^*$. Să presupunem că $0 \leq x_{ij} < x_{ij}^* \leq c_{ij}$ (dacă $x_{ij} > x_{ij}^*$, raționamentul este similar). Din legea conservării fluxului, există $jk \in E$ astfel încât $0 \leq x_{jk} < x_{jk}^* \leq c_{jk}$, sau există $kj \in E$ astfel încât $0 \leq x_{kj}^* < x_{kj} \leq c_{kj}$.

Deoarece numărul de noduri este finit, repetând acest raționament, obținem, C , un A -drum închis relativ la x în R :



Fluxuri 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

Dacă parcurgem C în sens contrar, obținem un A -drum închis, C' , relativ la x^* . Deoarece $a(C) \geq 0$ (din ipoteză), și $a(C') = -a(C)$ urmează că $a(C) = 0$. (x^* este de cost minim; astfel, din implicația directă, $a(C') \geq 0$).

Dacă vom considera $x' = x^* \otimes \delta(C')$, unde

$$\delta(C') = \min \left\{ \min_{kj \text{ forward in } C'} (x_{kj} - x_{kj}^*), \min_{kj \text{ backward in } C'} (x_{kj}^* - x_{kj}) \right\},$$

atunci x' satisfacă $v(x') = v(x^*) = v$, $a(x') = a(x^*) + \delta(C') \cdot a(C') = a(x^*)$.

Astfel x' este un flux de cost minim de valoare v , dar $\Delta(x, x') < \Delta(x, x^*)$, în contradicție cu alegerea lui x^* . Astfel $\Delta(x, x^*) = 0$, și teorema este demonstrată. \square

Teorema 2

Dacă x este un flux de cost minim de valoare v și P_0 este un drum de creștere relativ la x astfel încât

$$a(P_0) = \min \{ a(P) : P \text{ drum de creștere relativ la } x \},$$

atunci $x^1 = x \otimes r(P_0)$ este un flux de cost minim de valoare $v(x^1) = v + r(P_0)$.

Demonstrație. Omisă.

Un drum de creștere de cost minim poate fi găsit folosind un algoritm pentru drumuri de cost minim. Dacă x este un flux în R și $a : E \rightarrow \mathbb{R}$ este o funcție de cost, atunci luând $a_{ij} = \infty$ dacă $ij \notin E$ (când $x_{ij} = 0$), construim

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

$$\bar{a}_{ij} = \begin{cases} a_{ij}, & \text{dacă } x_{ij} < c_{ij} \text{ și } x_{ji} = 0, \\ \min \{ a_{ij}, -a_{ji} \}, & \text{dacă } x_{ij} < c_{ij} \text{ și } x_{ji} > 0, \\ -a_{ji}, & \text{dacă } x_{ij} = c_{ij} \text{ și } x_{ji} > 0, \\ +\infty, & \text{dacă } x_{ij} = c_{ij} \text{ și } x_{ji} = 0. \end{cases}$$

Un st -drum de cost \bar{a} minim corespunde unui drum de creștere de cost minim relativ la x în R , și un circuit de cost negativ corespunde unui A -drum închis relativ la x în R de cost negativ.

Atunci, avem următorul algoritm pentru determinarea unui flux 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 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algo373ns * C. Croitoru - Graph Algorithms *

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

fie x un flux de valoare $v' \leq v$; // x poate fi nul sau $x = (v/v(y) \cdot y$, unde y este un flux valoare maximă.

```

while ( $\exists C$  un circuit de cost  $\bar{a}$  negativ) do
     $x \leftarrow x \otimes r(C)$ ;
end while
while ( $v(x) < v$ ) do
    determină un  $st$ -drum  $P$  de cost  $\bar{a}$  minim;
     $x \leftarrow x \otimes \min \{r(P), v - v(x)\}$ ;
end while

```

Complexitatea timp al celui de-al doilea **while** este $\mathcal{O}(n^3v)$ (dacă pornim cu fluxul nul și capacitatele sunt întregi). Primul **while** poate fi implementat astfel încât să aibă $\mathcal{O}(nm^2 \log n)$ iteratii.

Reduceri în timp polinomial pentru probleme pe grafuri - Memento

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

Memento

- Fie $P_i : I_i \rightarrow \{da, nu\}$ ($i \in \{1, 2\}$) două **probleme de decizie**. P_1 se reduce polinomial la P_2 , și notăm aceasta prin $P_1 \leqslant_P P_2$, dacă există o **funcție calculabilă în timp polinomial** $\Phi : I_1 \rightarrow I_2$, astfel încât $P_1(i) = P_2(\Phi(i))$, $\forall i \in I_1$.
- Funcțiile Φ vor fi definite folosind un algoritm care construiește, pentru fiecare instanță $i_1 \in I_1$, o instanță $i_2 \in I_2$ în timp polinomial (în dimensiunea lui i_1), astfel încât $P_1(i_1) = da$ dacă și numai dacă $P_2(i_2) = da$.
- Construcția din spatele reducerii în timp polinomial arată cum prima problemă poate fi rezolvată eficient folosind un oracol pentru cea de-a doua.

- Relația \leqslant_P este o relație tranzitivă pe mulțimea problemelor de decizie (deoarece, clasa funcțiilor polinomial calculabile este închisă la compunere).

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

Exemplu

$$\text{SAT} \leqslant_P \text{3SAT}$$

SAT

Instantă: $U = \{u_1, u_2, \dots, u_n\}$ o mulțime finită de variabile booleene;

$C = C_1 \wedge C_2 \dots \wedge C_m$ o formulă CNF peste U :

$C_i = v_{i1} \vee v_{i2} \vee \dots \vee v_{ik_i}$, $i = \overline{1, m}$, unde

$\forall i_j, \exists \alpha \in \{1, 2, \dots, n\}$ a. i. $v_{i_j} = u_\alpha$ sau $v_{i_j} = \bar{u}_\alpha$.

Întrebare: Există o asignare $t : U \rightarrow \{\text{true}, \text{false}\}$ a. î. $t(C) = \text{true}$?

Reduceri în timp polinomial pentru probleme pe grafuri - Memento

3SAT este restricția problemei SAT la mulțimea instanțelor în care fiecare clauză C_i are exact 3 literali ($k_i = 3$), unde un literal, v_{ij} , este o variabilă sau o variabilă negată.

Problema SAT este faimoasă deoarece este prima problemă despre care s-a arătat ca este NP-completă (Cook, 1971).

$\text{NP} \neq \text{NP} \cap \text{coNP} = \text{P}$

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo³⁷⁵ns * C. Croitoru - Graph Algorithms *

SM

Instanță: $G = (V, E)$ un graf și $k \in \mathbb{N}$.

Întrebare: Există o mulțime stabilă S în G astfel încât $|S| \geq k$?

Teorema 3

(Karp, 1972). $3\text{SAT} \leq_P \text{SM}$.

Demonstrație. Fie $U = \{u_1, u_2, \dots, u_n\}$, ($n \in \mathbb{N}^*$), $C = C_1 \wedge C_2 \dots \wedge C_m$ ($m \in \mathbb{N}^*$) cu $C_i = v_{i_1} \vee v_{i_2} \vee v_{i_3}$, $i = \overline{1, m}$, (unde $\forall i_j, \exists \alpha \in \{1, 2, \dots, n\}$ a. î. $v_{i_j} = u_\alpha$ sau $v_{i_j} = \bar{u}_\alpha$) reprezentând datele unei instanțe a problemei 3SAT .

Vom construi în timp polinomial (în $n + m$) un graf G și $k \in \mathbb{N}$, astfel încât există o asignare t care satisface C dacă și numai dacă există o mulțime stabilă S în G astfel încât $|S| \geq k$.

Demonstrație (continuare).

Construcția grafului G :

- $\forall i \in \{1, 2, \dots, n\}$, fie grafurile disjuncte $T_i = (\{u_i, \bar{u}_i\}, \{u_i \bar{u}_i\})$.
- $\forall j \in \{1, 2, \dots, m\}$, fie grafurile disjuncte $Z_j = (\{a_{j1}, a_{j2}, a_{j3}\}, \{a_{j1}a_{j2}, a_{j2}a_{j3}, a_{j3}a_{j1}\})$.
- $\forall j \in \{1, 2, \dots, m\}$, fie mulțimea de muchii $E_j = \{a_{j1}v_{j1}, a_{j2}v_{j2}, a_{j3}v_{j3}\}$, unde $v_{j1} \vee v_{j2} \vee v_{j3}$ este clauza C_j .

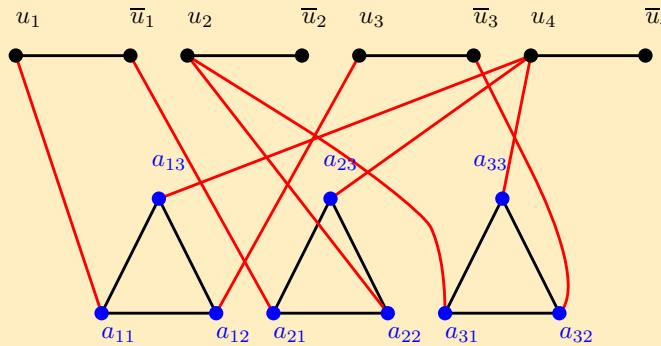
$$V(G) = \left(\bigcup_{i=1}^n V(T_i) \right) \cup \left(\bigcup_{j=1}^m V(Z_j) \right)$$

$$E(G) = \left(\bigcup_{i=1}^n E(T_i) \right) \cup \left(\bigcup_{j=1}^m E(Z_j) \cup E_j \right).$$

Evident, construcția lui G se poate face în timp polinomial relativ la dimensiunea $n + m$ a instanței 3SAT. Fie $k = n + m$.

Exemplu

$U = \{u_1, u_2, u_3, u_4\}; C = (u_1 \vee u_3 \vee u_4) \wedge (\bar{u}_1 \vee u_2 \vee u_4) \wedge (u_2 \vee \bar{u}_3 \vee u_4); k = 4 + 3 = 7.$



Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

Să presupunem că răspunsul la întrebarea problemei SM pentru (G, k) instanță este da.

$\exists S \in \mathcal{S}_G$ (familia tuturor mulțimilor stabile din G) astfel încât $|S| \geq k$. Deoarece fiecare mulțime stabilă poate avea cel mult un nod în fiecare $V(T_i)$ și fiecare $V(Z_j)$, urmează că $|S| = k$ și $|S \cap V(T_i)| = 1$, $|S \cap V(Z_j)| = 1$, $\forall i = \overline{1, n}$, $\forall j = \overline{1, m}$.

Fie $t : U \rightarrow \{\text{true}, \text{false}\}$ dată prin

$$t(u_i) = \begin{cases} \text{true}, & \text{dacă } S \cap V(T_i) = \{\bar{u}_i\} \\ \text{false}, & \text{dacă } S \cap V(T_i) = \{u_i\} \end{cases}$$

Atunci, $t(C_j) = \text{true}$, $\forall j = \overline{1, m}$, astfel $t(C) = \text{true}$, și răspunsul la 3SAT este da.

Într-adevăr, $\forall j = \overline{1, m}$, dacă $C_j = v_{j1} \vee v_{j1} \vee v_{j3}$ și $S \cap V(Z_j) = \{a_{jk}\}$ ($k \in \{1, 2, 3\}$), atunci (deoarece $a_{jk} v_{jk} \in E$) urmează că $v_{jk} \notin S$.

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

- Dacă $v_{jk} = u_\alpha$, atunci $u_\alpha \notin S$, deci $\bar{u}_\alpha \in S$, și - din definiția lui t - avem $t(u_\alpha) = \text{true}$, adică, $t(v_{jk}) = \text{true}$, ceea ce implică $t(C_j) = \text{true}$.
- Dacă $v_{jk} = \bar{u}_\alpha$, atunci $\bar{u}_\alpha \notin S$, deci $u_\alpha \in S$, și - din definiția lui t - avem $t(\bar{u}_\alpha) = \text{true}$, adică, $t(v_{jk}) = \text{true}$, ceea ce implică $t(C_j) = \text{true}$.

Reciproc, dacă răspunsul la întrebarea problemei 3SAT este da, atunci $\exists t : U \rightarrow \{\text{true}, \text{false}\}$ astfel încât $t(C_j) = \text{true}$, $\forall j = \overline{1, m}$.

Fie \bar{S}_1 mulțimea stabilă $\bar{S}_1 = \bigcup_{i=1}^n V'_i$ cu n noduri, unde

$$V'_i = \begin{cases} \{\bar{u}_i\}, & \text{dacă } t(u_i) = \text{true} \\ \{u_i\}, & \text{dacă } t(u_i) = \text{false} \end{cases}$$

Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

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

Atunci, deoarece $t(C_j) = \text{true}$, $\forall j = \overline{1, m}$, urmează că există $k_j \in \{1, 2, 3\}$ astfel încât $t(v_{jk_j}) = \text{true}$. Fie $\bar{S}_2 = \bigcup_{j=1}^m \{a_{jk_j}\}$. Evident, \bar{S}_2 este o mulțime stabilă în G având m noduri.

Fie $\bar{S} = \bar{S}_1 \cup \bar{S}_2$. Evident, $|\bar{S}| = n + m = k$ (astfel $|\bar{S}| \geq k$). Dacă arătăm că \bar{S} este o mulțime stabilă, atunci **răspunsul la SM pentru instanță (G, k) este da**.

Să presupunem că $\exists v, w \in \bar{S}$ astfel încât $e = vw \in E(G)$. Atunci o extremitate a lui e este în \bar{S}_1 , iar cealaltă în \bar{S}_2 . Dacă $v \in \bar{S}_1$, atunci avem două cazuri:

- $v = u_\alpha$, $w = a_{jk_j}$, $\alpha \in \{1, 2, \dots, n\}$, $j \in \{1, 2, \dots, m\}$, $k_j \in \{1, 2, 3\}$ și $v_{jk_j} = u_\alpha$.

Deoarece $t(v_{jk_j}) = \text{true}$, urmează că $t(u_\alpha) = \text{true}$, astfel $v = u_\alpha \notin \bar{S}_1$, contradicție.

- $v = \bar{u}_\alpha$, $w = a_{jk_j}$, $\alpha \in \{1, 2, \dots, n\}$, $j \in \{1, 2, \dots, m\}$, $k_j \in \{1, 2, 3\}$
 si $v_{jk_j} = \bar{u}_\alpha$.

Deoarece $t(v_{jk_j}) = \text{true}$, urmează că $t(\bar{u}_\alpha) = \text{true}$, astfel $t(u_\alpha) = \text{false}$. Astfel, $v = \bar{u}_\alpha \notin \overline{S}_1$, contradicție. \square

O demonstrație similară poate fi făcută pentru a arăta că $SAT \leqslant_P SM$, singura diferență este că Z_i sunt grafuri complete cu k_i noduri.

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

Reduceri în timp polinomial - Colorări ale nodurilor

COL

Instantă: $G = (V, E)$ graf și $p \in \mathbb{N}^*$.

Întrebare: Există o p -colorare (a nodurilor) lui G ?

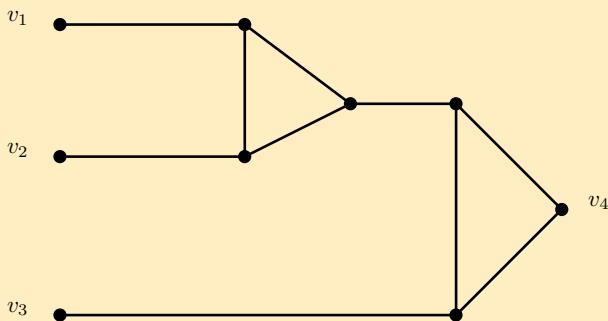
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Teorema 4

$\text{3SAT} \leqslant_P \text{COL.}$

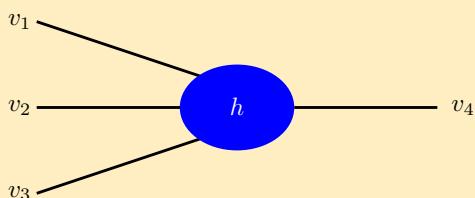
Această teoremă arată că problema colorării nodurilor este NP-hard. Demonstrația dată mai jos arată că problema, care se obține din COL prin restricția la instanțele cu $p = 3$, care poate fi numită 3-COL, este NP-hard, de asemenei.

Lema 1

Fie H graful

- a) Dacă c este o 3-colorare a lui H a. î. $c(v_1) = c(v_2) = c(v_3) = a \in \{1, 2, 3\}$, atunci în mod necesar $c(v_4) = a$ (forcing).
- b) Dacă $c : \{v_1, v_2, v_3\} \rightarrow \{1, 2, 3\}$ satisfacă $c(\{v_1, v_2, v_3\}) \neq \{a\}$, ($a \in \{1, 2, 3\}$), atunci c poate fi extinsă la o 3-colorare c a lui H cu $c(v_4) \neq a$.

Reduceri în timp polinomial - Colorări ale nodurilor

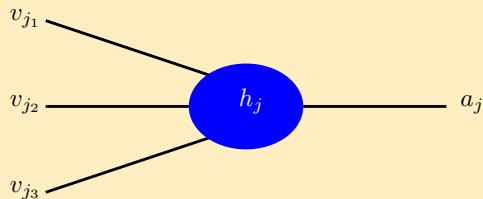
Demonstrație. Se examinează lista 3-colorărilor lui H .Vom utiliza reprezentarea simplificată a grafului H :

Demonstrația Teoremei 4. Considerăm datele unei instanțe a 3SAT: $U = \{u_1, \dots, u_n\}$, ($n \in \mathbb{N}^*$), o mulțime de variabile booleene, și $C = C_1 \wedge \dots \wedge C_m$, ($m \in \mathbb{N}^*$) formulă CNF cu $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$, $\forall j = \overline{1, m}$, unde $\forall i = \overline{1, 3}$, $\exists \alpha$ a. î. $v_{j_i} = u_\alpha$ sau $v_{j_i} = \bar{u}_\alpha$.

Vom construi un graf G cu proprietatea că G este 3-colorabil dacă și numai dacă răspunsul la 3SAT pentru această instanță este da, adică, există o asignare $t : U \rightarrow \{\text{true}, \text{false}\}$ astfel încât $t(C) = \text{true}$.

Construcția necesită un timp polinomial, și constă din următorii pași:

- Considerăm grafurile disjuncte (V_i, E_i) , $\forall i = \overline{1, n}$, unde $V_i = \{u_i, \bar{u}_i\}$ și $E_i = \{u_i \bar{u}_i\}$.
- Pentru $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$, $\forall j = \overline{1, m}$, considerăm grafurile:



unde, v_{jk} ($k = \overline{1, 3}$) sunt nodurile corespunzătoare literalilor v_{jk} , grafurile h_j sunt disjuncte, și a_j sunt noduri distințte.

- Considerăm un nod nou a , și toate muchiile aa_j , $\forall j = \overline{1, m}$.
- Considerăm un nod nou b , toate muchiile bu_i , $b\bar{u}_i$, $\forall i = \overline{1, n}$ și ba .

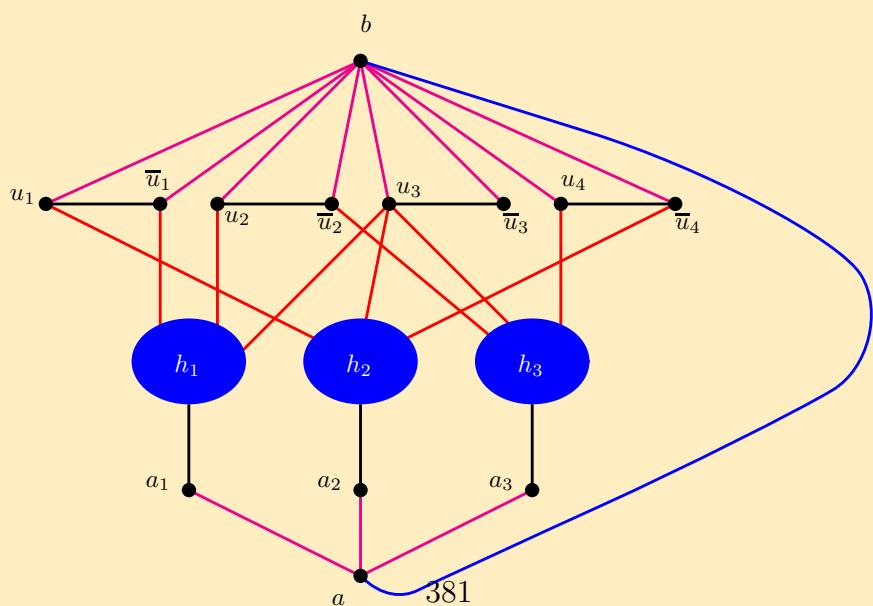
Graful G are un număr de liniar de noduri relativ la $n + m$.

Reduceri în timp polinomial - Colorări ale nodurilor

Exemplu

$$U = \{u_1, u_2, u_3, u_4\}, C = (\bar{u}_1 \vee u_2 \vee u_3) \wedge (u_1 \vee u_3 \vee \bar{u}_4) \wedge (\bar{u}_2 \vee u_3 \vee u_4).$$

Graful G este



Să presupunem că răspunsul la 3SAT pentru instanță considerată este da.

Astfel $\exists t : U \rightarrow \{\text{true}, \text{false}\}$ a. î. $t(C) = \text{true}$, adică, $t(C_j) = \text{true}$, $\forall j = \overline{1, m}$. Vom arăta că G este 3-colorabil.

Colorăm mai întâi nodurile u_i și \bar{u}_i , $\forall i \in \overline{1, n}$.

$$\begin{cases} c(u_i) = 1 \text{ și } c(\bar{u}_i) = 2, \text{ dacă } t(u_i) = \text{true} \\ c(u_i) = 2 \text{ și } c(\bar{u}_i) = 1, \text{ dacă } t(u_i) = \text{false} \end{cases}$$

Observăm că, dacă v este un literal, atunci $c(v) = 2$ dacă și numai dacă $t(v) = \text{false}$.

Deoarece t este o asignare pentru care satisfacă C , $t(C_j) = \text{true}$, $\forall j = \overline{1, m}$. Urmează că $c(\{v_{j_1}, v_{j_2}, v_{j_3}\}) \neq \{2\}$, $\forall j = \overline{1, m}$.

Din Lema 1 b), putem extinde c la o 3-colorare, în fiecare graf h_j , astfel încât $c(a_j) \neq 2$, adică $c(a_j) \in \{1, 3\}$.

Luând $c(a) = 2$ și $c(b) = 3$, obținem o 3-colorare a lui G .

Reduceri în timp polinomial - Colorări ale nodurilor

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

Reciproc, să presupunem că G este 3-colorabil.

Putem presupune că $c(b) = 3$ și $c(a) = 2$ (altfel, redenumim culorile).

Urmează că $\{c(u_i), c(\bar{u}_i)\} = \{1, 2\}$, $\forall i = \overline{1, n}$ și $c(a_j) \in \{1, 3\}$, $\forall j = \overline{1, m}$.

Din Lema 1 a), urmează că $c(\{v_{j_1}, v_{j_2}, v_{j_3}\}) \neq 2$, $\forall j = \overline{1, m}$. Aceasta înseamnă că, $\forall j = \overline{1, m}$, există a $v_{j_k} \in C_j$ astfel încât $c(v_{j_k}) = 1$.

Astfel, definind $t : U \rightarrow \{\text{true}, \text{false}\}$ prin

$$t(u_i) = \begin{cases} \text{true}, & \text{dacă } c(u_i) = 1 \\ \text{false}, & \text{dacă } c(u_i) = 2 \end{cases},$$

Obținem o asignare cu proprietatea că $t(C_j) = \text{true}$, $\forall j = \overline{1, m}$.

Astfel răspunsul la 3SAT pentru instanță dată este da.

Memento: Fie G un (di)graf. Un circuit C al lui G este un circuit Hamiltonian dacă $V(C) = V(G)$.

Un drum deschis P al lui G este un **drum Hamiltonian** dacă $V(P) = V(G)$. Un **(di)graf Hamiltonian** este un (di)graf care are un circuit Hamiltonian. Un **(di)graf trasabil** este a (di)graf care conține un drum Hamiltonian.

Teorema 5

(Nash-Williams, 1969) Următoarele cinci probleme sunt echivalente polinomial:

CH: Dat un graf G . Este G Hamiltonian?

TR: Dat un graf G . Este G trasabil?

Reduceri în timp polinomial - Probleme Hamiltoniene

DCH: Dat un digraf G . Este G Hamiltonian?

DTR: Dat un digraf G . Este G trasabil?

BCH: Dat un graf bipartit G . Este G Hamiltonian?

Remarcă

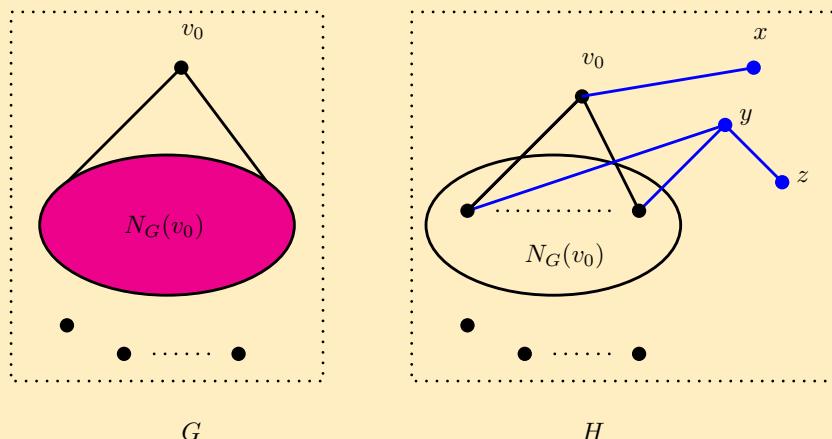
P_1 și P_2 sunt **echivalente polinomial** dacă $P_1 \leqslant_P P_2$ și $P_2 \leqslant_P P_1$.

Demonstrația Teoremei 5.

CH \leqslant_P TR Fie G un graf și $v_0 \in V(G)$. Construim în timp polinomial un graf H astfel încât G este Hamiltonian dacă și numai dacă H este trasabil.

Fie $V(H) = V(G) \cup \{x, y, z\}$ și $E(H) = E(G) \cup \{xv_0, yz\} \cup \{wy : w \in V(G) \cap N_G(v_0)\}$.

Demonstrația Teoremei 5 (continuare).

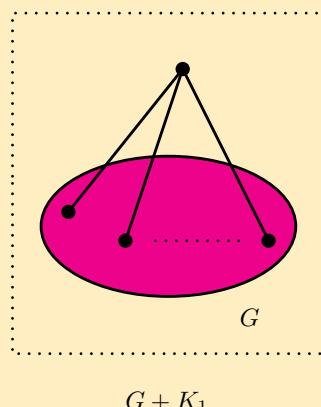


Atunci, H este trasabil dacă și numai dacă are un drum Hamiltonian, P , cu extremitățile x și z (care au gradul 1 în H). P există în H dacă și numai dacă în G există un drum Hamiltonian cu o extremitate în v_0 și cealaltă un vecin al lui v_0 , adică, dacă și numai dacă G este Hamiltonian.

Reduceri în timp polinomial - Probleme Hamiltoniene

Demonstrația Teoremei 5 (continuare).

$\text{TR} \leqslant_P \text{CH}$ Fie G un graf. Considerăm $H = G + K_1$. Atunci, H este Hamiltonian dacă și numai dacă G are un drum Hamiltonian.



Echivalența problemelor **DCH** și **DTR** se dovedește similar.

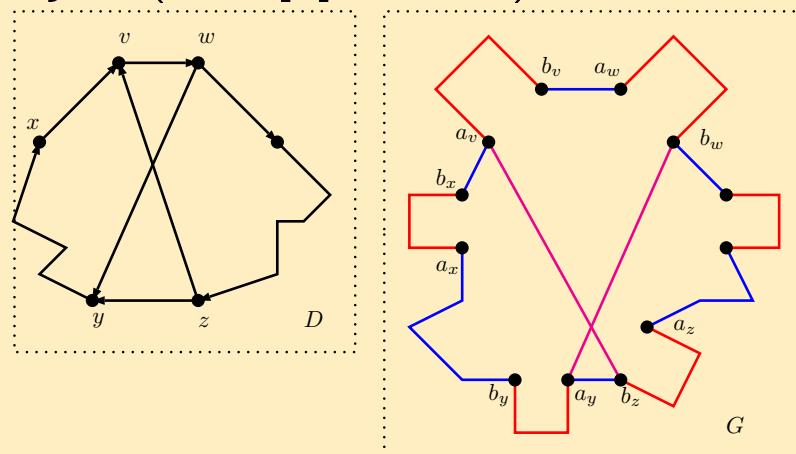
$\text{CH} \leqslant_P \text{DCH}$ Fie G un graf. Fie D digraful obținut din G prin înlocuirea fiecărei muchii cu o pereche simetrică de arce.

Evident fiecare circuit în G dă un circuit în D și reciproc, fiecare circuit în D provine dintr-un circuit în G .

DCH \leqslant_P **CH** fie D un digraf. Fiecare nod $v \in V(D)$ este înlocuit printr-un graf neorientat $P_3(v)$ cu extremități a_v și b_v :

$$P_3(v) = (\{a_v, b_v, c_v, d_v\}, \{a_v c_v, c_v d_v, d_v b_v\}).$$

Fiecăreia arc $vw \in E(D)$ este înlocuit prin muchia (neorientată) $b_v a_w$. Fie G graful obținut (în timp polinomial) astfel:



Reduceri în timp polinomial - Probleme Hamiltoniene

Fiecare circuit C al lui D corespunde unui circuit în G și, reciproc, fiecare circuit în G corespunde unui circuit al lui D . Urmează că D este Hamiltonian dacă și numai dacă G este Hamiltonian.

Să observăm că dacă C este un circuit al lui G , atunci acesta este generat de un circuit C' al lui D , și $\text{length}(C) = 3 \cdot \text{length}(C') + \text{length}(C') = 4 \cdot \text{length}(C')$. Urmează că orice circuit al lui G este par, deci G este un graf bipartit.

Astfel demonstrația de mai sus ($DCH \leqslant_P CH$) este de fapt $DCH \leqslant_P BCH$.

Deoarece $BCH \leqslant_P CH$ este evidentă, Teorema 5 este complet demonstrată. \square

Exercițiu 1. Arătați că următoarea problemă este NP-completă INT

Instanță: $n, m \in \mathbb{N}^*$, $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$.

Întrebare: Există o asignare $x \in \mathbb{Z}^n$ a. i. $Ax \leq b$?

(Inegalitatea \leq dintre doi vectori este pe componente.) Hint: You can try SM \leq_P INT.

Exercițiu 2. Considerăm următoarea problemă de decizie ACYCLIC

Instanță: $G = (V, E)$ un digraf și $p \in \mathbb{N}$.

Întrebare: Există $A \subseteq V$ astfel încât $|A| \leq p$ și $G - A$ nu conține circuite?

Arătați că SM \leq_P ACYCLIC.

Exerciții pentru seminarul din săptămâna viitoare

Exercițiu 3. Un hipergraf k -uniform este o pereche $H = (V, E)$, unde $V \neq \emptyset$ este o mulțime finită, $k \in \mathbb{N}^* \setminus \{1\}$, și $E \subseteq \mathcal{P}_k(V) = \{A \subseteq E : |A| = k\}$. Este ușor de văzut că un hipergraf 2-uniform este un graf simplu.

Spunem că un hipergraf k -uniform $H = (V, E)$ este *simple* dacă există o funcție $c : V \rightarrow \{1, 2, \dots, k\}$ astfel încât $\forall u, v \in V$, $u \neq v$, dacă $u, v \in e$ pentru o anumită muchie $e \in E$, atunci $c(u) \neq c(v)$. Considerăm următoarea problemă de decizie

k -SIMPLE

Instanță: H un hipergraf k -uniform.

Întrebare: H este simplu?

- Arătați că problema 3-SIMPLE este NP-completă.
- Arătați că problema 2-SIMPLE este în P.

Exercițiu 4. Considerăm următoarea problemă de decizie:

AGM

Instanță: G un graf, $k \in \mathbb{N}$.

Întrebare: G are un arbore parțial T astfel încât $\Delta(T) \geq k$?

Arătați că AGM $\in P$.

Exercițiu 5. Fie $D = (V, E)$ un digraf fără bucle. O mulțime stabilă a lui D , $S \subseteq V$, este numită **quasi-kernel** dacă fiecare nod $v \in V \setminus S$ este accesibil din S pe un drum de lungime cel mult 2.

- (a) Arătați că un quasi-kernel poate fi construit în $\mathcal{O}(n + m)$, unde $n = |V|$ și $m = |E|$.
- (b) Arătați că 3-SAT se poate reduce în timp polinomial la problema determinării dacă într-un digraf dat există un quasi-kernel care conține un nod dat.

Exercițiu 6. Considerăm următoarea problemă de decizie: **LPL**

Instanță: G un graf, $k \in \mathbb{N}$.

Întrebare: Are G un drum P astfel încât $length(P) \geq k$?

Arătați că LPL este NP-completă.

Exercițiu 7. Un **kernel** într-un digraf $G = (V, E)$ este o mulțime stabilă $S \subseteq V$ astfel încât $\forall u \in V \setminus S$ există $v \in S$ cu $vu \in E$. Considerăm următoarea problemă de decizie:

KERNEL

Instanță: G un digraf.

Întrebare: Are G un kernel?

Arătați că următoarea construcție conduce la o reducere polinomială a lui SAT la KERNEL (i.e. $SAT \leq_P KERNEL$):

Exercițiu 7 (continuare). Pentru fiecare conjuncție de clauze, F , instanță a lui SAT, definim un digraf G (o instanță pentru NUCLEU):

- pentru fiecare clauză C a lui F adăugăm un 3-circuit la G

$$v_C^1, v_C^1 v_C^2, v_C^2, v_C^2 v_C^3, v_C^3, v_C^3 v_C^1;$$

- pentru fiecare variabilă x care apare în formula F , adăugăm un 2-circuit la G

$$v_x, v_x v_{\bar{x}}, v_{\bar{x}}, v_{\bar{x}} v_x, v_x;$$

- pentru fiecare clauză C și fiecare literal u care apare în C adăugăm a G trei arce

$$v_u v_C^1, v_u v_C^2, v_u v_C^3.$$

Exerciții pentru seminarul din săptămâna viitoare

Exercice 8. Considerăm următoarea problemă de decizie

2SAT

Instanță: C o familie de clauze fiecare cu doi literali.

Întrebare: Există o asignare a valorilor de adevăr către variable astfel ca toate clauzele din C să fie satisfăcute?

Define G digraful (implicațiilor): $V(G) =$ mulțimea literalilor folosiți în C și $E(G) = \{\overline{v_j} w_j, \overline{w_j} v_j : C_j = v_j \vee w_j, j = \overline{1, m}\}$ (fiecare clauză introduce în G două arce). Arătați că C este satisfiabilă dacă și numai dacă x_i și \overline{x}_i aparțin la componente tari conexe diferite ale lui G , $\forall i = \overline{1, n}$. Arătați că această proprietate poate fi verificată în $\mathcal{O}(n + m)$.

Exercice 9. Arătați că următoarea problemă este NP-completă.

MAX-2SAT

Instanță: C o familie de clauze fiecare cu cel mult doi literali și $k \in \mathbb{N}$.

Întrebare: Există o asignare a valorilor de adevăr către variable astfel ca cel puțin k dintre clauze să fie satisfăcute?

Exercise 10. Considerăm următoarea problemă de decizie**NAE-3SAT**

Instanță: C o familie de clauze fiecare cu trei literali.

Întrebare: Există o asignare a valorilor de adevăr către variabile astfel ca în fiecare clauză să avem și un literal adevărat și unul fals?

Arătați că următoarea construcție conduce la o reducere polinomială a lui 3SAT la NAE 3SAT (i.e. $3SAT \leqslant_P NAE\ 3SAT$):

- păstrăm variabilele booleene ale instanței 3SAT, $U = \{u_1, u_2, \dots, u_n\}$ și adăugăm o variabilă (nouă) x ;
- pentru fiecare clauză $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$ adăugăm o variabilă nouă y_j și înlocuim C_j cu două clauze:

$$C_j^1 = v_{j_1} \vee v_{j_2} \vee y_j, C_j^2 = v_{j_3} \vee x \vee \bar{y}_j.$$

Exercițiul 11. Considerăm următoarea problemă de decizie**MAX-CUT**

Instanță: $G = (V, E)$ un graf, $c : E \rightarrow \mathbb{R}$ o funcție de pondere și $k \in \mathbb{R}$.

Întrebare: există o tăietură în G de pondere $\geq k$?

Arătați că următoarea construcție duce la o reducere polinomială a problemei NAE 3SAT la MAXCUT (i.e. $NAE\ 3SAT \leqslant_P MAXCUT$):

- considerăm o instanță a problemei NAE 3SAT cu clauzele $C = \{C_1, \dots, C_m\}$ peste mulțimea de variabile booleene $U = \{u_1, u_2, \dots, u_n\}$; putem presupune că fiecare clauză conține trei literali diferiți,
- $V(G) = \{u_i, \bar{u}_i : i = \overline{1, n}\}$ și adăugăm la $E(G)$ muchiile $u_i \bar{u}_i$ de pondere $10m$,
- pentru fiecare clauză $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$ adăugăm la $E(G)$ trei muchii $v_{j_1} v_{j_2}$, $v_{j_2} v_{j_3}$ și $v_{j_1} v_{j_3}$, fiecare de pondere 1,
- $k = 10nm + 2m$.

Memento

- Pentru a arăta că o problemă este NP-completă:
- I. Mai întâi trebuie arătat mai întâi că este în clasa NP adică un candidat pentru a fi soluție a problemei poate fi verificat în timp polinomial (pentru 3COL: fie o funcție $c : V(G) \rightarrow \mathbb{N}$, se poate verifica că c este o colorare și folosește cel mult trei culori în timp polinomial, de exemplu $\mathcal{O}(n^2)$).
 - II. Apoi urmează demonstrația reducerii polinomiale de la o altă problemă NP-completă (pentru 3COL: 3SAT se reduce în timp polinomial la 3COL).
- Dacă pasul I. lipsește atunci problema de decizie în cauză este NP-hard.

Exerciții rezolvate (partial)

Exercițiul 1. Soluție.

- Mai întâi aratăm că INT \in NP: pentru orice $x \in \mathbb{Z}^m$ se poate verifica în $\mathcal{O}(nm)$ dacă $Ax \leq b$ (de ce?).
- Fie $G = (V, E)$ be a graf and $k \in \mathbb{N}$ an instanță for SM.
- Existența unei mulțimi stabile S în G cu $|S| \geq k$ este echivalentă cu existența unei soluții a următoarei probleme liniare

$$(LP) \left\{ \begin{array}{l} (1) \quad \sum_{v \in V} x_v \geq k, \\ (2) \quad x_u + x_v \leq 1, \quad \forall uv \in E. \\ (3) \quad x_v \in \{0, 1\}, \quad \forall v \in V(G). \end{array} \right.$$

unde:

- relațiile (3) definesc x drept vectorul caracteristic al unei submulțimi S a lui $V(G)$,
- relațiile (2) arată că S este o mulțime stabilă,
- iar (1) spune că S conține cel puțin k noduri.

- Fie $V(G) = \{v_1, v_2, \dots, v_n\}$ și $E(G) = \{e_1, e_2, \dots, e_m\}$; putem defini

$$b = (1, 1, \dots, 1, -k, 0, 0, \dots, 0, 1, 1, \dots, 1)^T \in \mathbb{Z}^{2n+m+1}$$

$$A = (a_{ij})_{\substack{1 \leq i \leq 2n+m+1 \\ 1 \leq j \leq n}},$$

$$a_{ij} = \begin{cases} 1, & e_i = v_j v_k, 1 \leq i \leq m \text{ or } n+m+2 \leq i = j \leq 2n+m, \\ -1, & i = m+1 \text{ or } m+2 \leq i = j \leq n+m, \\ 0, & \text{otherwise.} \end{cases}$$

- Construcția lui A și b se face în timpul $\mathcal{O}(n+m)$ (de ce?).
- (LP) are o soluție dacă și numai dacă la întrebarea problemei INT cu A și b drept input răspunsul este afirmativ (de ce?).
- Astfel, $\text{SM} \leq_P \text{INT}$.

Exerciții rezolvate (partial)

Exercițiul 2. Soluție.

- Fie H un graf și $k \in \mathbb{N}$ - o instanță pentru SM.
- Definim digraful G astfel:

$$V(G) = V(H), E(G) = \{xy, yx : xy \in E(H)\}$$

și $p = |H| - k$ (G se construiește plecând de la H în $\mathcal{O}(|V(H)| + |E(H)|))$; (G și p formează o instanță pentru problema ACYCLIC).

- Dacă H conține o mulțime stabilă S cu $|S| \geq k$, atunci definim $A = V(H) \setminus S$, avem $|A| \leq p$ iar $G \setminus A$ este aciclic (de ce?).
- Dacă G conține o mulțime de noduri A , cu $|A| \leq p$ a. î. $G \setminus A$ este aciclic, atunci $S = V(H) \setminus A$ este o mulțime stabilă în H (de ce?) și $|S| \geq k$.

Exercițiu 3. Soluție.

- Evident, 3-SIMPLE $\in \text{NP}$ deoarece, pentru o funcție dată $c : V(H) \rightarrow \{1, 2, 3\}$, se poate verifica în $\mathcal{O}(n^3)$ dacă, pentru orice $e \in E(H)$, $c(u) \neq c(v)$, $\forall u \neq v \in e$. (de ce?)
- Arătăm că 3-COL \leqslant_P 3-SIMPLE. Fie $G = (V, E(G))$ un graf (input pentru 3-COL).
- Definim o instanță pentru 3-SIMPLE astfel

```

 $V(H) \leftarrow V(G);$ 
for ( $[\{x, y, z\}]_G \cong K_3$ )
     $E(H) \leftarrow E \cup \{\{x, y, z\}\};$ 
    for ( $e = xy \in E(G)$  not contained in any  $K_3$ ) {
         $E(H) \leftarrow E \cup \{\{x, y, w_{xy}\}\};$ 
         $V(H) \leftarrow V(H) \cup \{w_{xy}\};$ 
    }
}

```

Exerciții rezolvate (partial)

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

- Acesta este un algoritm de complexitate polinomială și $\chi(G) \leqslant 3$ dacă și numai dacă H este simplu (de ce?).

Exercițiu 4. Soluție.

- Ce se întâmplă dacă G nu este conex?
- Dacă $T \in \mathcal{T}_G$, $\Delta(G) \geqslant \Delta(T)$; ce se întâmplă când $k > \Delta(G)$?
- Presupunem că G este un graf conex și fie $x_0 \in V(G)$ cu $d_G(x_0) = \Delta(G)$ și T rezultatul unei parcurgeri bfs a lui G plecând din x_0 , evident $\Delta(T) = ?$; deci răspunsul la întrebarea AGM este ...?.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algo392ns * C. Croitoru - Graph Algorithms *

- (b) Considerăm o instanță a problemei 3-SAT, $X = \{x_1, x_2, \dots, x_n\}$ - mulțimea variabilelor booleene, $\mathcal{C} = \{C_1, C_2, \dots, C_m\}$ o familie de clauze, toate având cel mult trei literali peste X .
- Fie L mulțimea tuturor literalilor care apar în cel puțin o clauză din \mathcal{C} . Plecând de la această instanță construim un digraf D :

$$V(D) = \{x_C, y_{C_1}, z_{C_1}, y_{C_2}, z_{C_2}, \dots, y_{C_m}, z_{C_m}\} \cup \left(\bigcup_{j=1}^m \{x_j, \bar{x}_j\} \right).$$

$$\begin{aligned} E(D) = & \left(\bigcup_{i=1}^m \{y_{C_i} x_C, z_{C_i} x_C, z_{C_i} y_{C_i}\} \right) \cup \left(\bigcup_{i=1}^m \{v z_{C_i} : v \in C_i\} \right) \cup \\ & \cup \left(\bigcup_{j=1}^n \{x_i \bar{x}_i, \bar{x}_i x_i\} \right). \end{aligned}$$

Exerciții rezolvate (partial)

- Vom arăta că familia de clauze \mathcal{C} este satisfiabilă dacă și numai dacă D admite un quasi-kernel care conține x_C .
- " \Rightarrow " Fie $\varphi : X \rightarrow \{0, 1\}$ o asignare a valorilor de adevăr care satisfac toate clauzele din \mathcal{C} . Definim $S = \{x_C\} \cup \{v \in L : \varphi(v) = 1\}$.
- S este un quasi-kernel în D (**de ce?**).
- " \Leftarrow " Fie S be a quasi-kernel containing x_C ; avem $y_{C_i}, z_{C_i} \notin S$, $\forall i = \overline{1, m}$.
- Definim o asignare a valorilor de adevăr $\varphi : X \rightarrow \{0, 1\}$ considerând $\varphi(v) = 1$ dacă și numai dacă $v \in S$.
- Considerăm o clauză C_i , dacă $\varphi(C_i) = 0$, atunci $\varphi(v) = 0$ adică $v \notin S$, pentru orice literal v din C_i . Astfel y_{C_i} și z_{C_i} nu sunt accesibili din S , o contradicție.³⁹⁴

Exercițiu 6. Soluție.

- Mai întâi observăm că LPL este în NP (de ce?).
- Apoi arătăm că $\text{TR} \leqslant_P \text{LPL}$:
- Fie $G = (V, E)$ un graf (instanță pentru TR). Definim G și $n - 1$ o instanță pentru LPL.
- G este trasabil dacă și numai dacă există un drum în G de lungime cel puțin $n - 1$ (de ce?).

Exerciții rezolvate (partial)

Exercițiu 7. Soluție.

- Fie X mulțimea de variabile din F (\overline{X} este mulțimea literalilor negativi) și C_1, C_2, \dots, C_m clauzele din F .
- F este satisfiabilă dacă și numai dacă G admite un nucleu.
- “ \implies ” Presupunem că F este o formulă satisfiabilă: există o asignare a valorilor de adevăr $t : X \rightarrow \{0, 1\}$, astfel ca $t(C_j) = 1$, $\forall 1 \leqslant j \leqslant m$.
- Fie $S = \{v_u : u \in X \cup \overline{X}, t(u) = 1\}$ - o mulțime stabilă din G .
- Dacă $w \in V(G) \setminus S$, atunci $w = v_{\overline{u}}$ pentru un $u \in S$ sau $w = v_C^i$, pentru o clauză C în care apare cel puțin un literal u ($t(u) = 1$) (de ce?); în amândouă cazurile $v_u w \in E(G)$.
- Astfel, S este un nucleu al lui G .
- “ \Leftarrow ” Reciproc, presupunem că G are un nucleu S .

- Pentru orice clauză C din F , $|\{v_C^1, v_C^2, v_C^3\} \setminus S| \geq 2$ și trebuie să existe un nod $w_C = v_C^i \notin S$ care nu este "văzut" din $S \cap \{v_C^1, v_C^2, v_C^3\}$.
- Pentru orice variabilă $x \in X$, $|S \cap \{v_x, v_{\bar{x}}\}| = 1$ (de ce?), deci putem defini următoarea asignare $t : X \rightarrow \{0, 1\}$

$$t(x) = \begin{cases} 0, & v_{\bar{x}} \in S \\ 1, & v_x \in S \end{cases}.$$

- Fie C o clauză din F ; w_C trebuie să fie văzut și din afara mulțimii $\{v_C^1, v_C^2, v_C^3\}$, deci există un literal u așa că $v_u \in S$ și $v_u w_C \in E$, astfel u apare în C (de ce?) și $t(u) = t(C) = 1$.
- În plus, G poate fi construit în complexitatea timp (polinomială) $\mathcal{O}(|X| + m)$.

Exerciții rezolvate (partial)

Exercițiul 8. Soluție.

- “ \implies ” Presupunem că există $1 \leq i \leq n$ așa încât x_i și \bar{x}_i sunt în aceeași componentă tare conexă a lui G : vor exista în G un $x_i \bar{x}_i$ -drum și un $\bar{x}_i x_i$ -drum

$$x_i = u_0, u_1, \dots, u_k = \bar{x}_i \quad \text{and} \quad \bar{x}_i = v_0, v_1, \dots, v_h = x_i$$

Fie $t : \{x_1, x_2, \dots, x_n\} \rightarrow \{0, 1\}$ o asignare care satisfac toate clauzele din \mathcal{C} . Fie $t(x_i) = 1$; cum $u_{j-1} u_j \in E(G)$, avem $(\bar{u}_{j-1} \vee u_j) \in \mathcal{C}$, deci $t(u_{j-1}) = 0$ sau $t(u_j) = 1$, pentru orice $1 \leq j \leq k$. Astfel

$$t(u_0) = 1 \Rightarrow t(u_1) = 1 \Rightarrow \dots \Rightarrow t(u_k) = ?$$

- Ce contradicție se obține de aici?
- O contradicție similară se obține dacă presupunem că $t(x_i) = 0$ folosind $\bar{x}_i x_i$ -drumul.

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

- “ \Leftarrow ” Să presupunem că pentru orice i , x_i și \bar{x}_i aparțin la componente tari conexe ale lui G . Vom defini o asignare $t : \{x_1, x_2, \dots, x_n\} \rightarrow \{0, 1\}$ care va satisface toate clauzele din \mathcal{C} .
- Fie $\mathcal{G} = (\mathcal{V}, \mathbb{E})$ digraful care are drept noduri componente tari conexe ale lui G iar două astfel de noduri G' și G'' vor fi adiacente dacă există un arc $u'u'' \in E(\mathcal{G})$, cu $u' \in V(G')$ și $u'' \in V(G'')$. Acest digraf este bine definit și este aciclic (**de ce?**).
- Sortăm topologic nodurile lui \mathcal{G} : G_1, G_2, \dots, G_p ; în acest fel, dacă există un arc de la G_i către G_j ($i \neq j$), atunci $i < j$ (**de ce?**).

* C. Croitoru - Graph Algorithms *

Exerciții rezolvate (partial)

```

for i = 1 to n { // t has not yet been defined anywhere
    t[xi] ← -1;
    t[ $\bar{x}_i$ ] ← -1;
}
for j = p down to 1 {
    if ( $\exists u \in V(G_j)$  such that  $t[\bar{u}] \neq -1$ )
        for v ∈ V(Gj)
            t(v) ← 1 - t( $\bar{u}$ );
    else
        for v ∈ V(Gj)
            t(v) ← 1;
}

```

- Se observă că, dacă $u, v \in V$ sunt într-o aceeași componentă tare conexă a lui G , atunci \bar{u} și \bar{v} sunt de asemenea într-o aceeași componentă tare conexă a lui (**de ce?** diferită de prima) - componente tari conexe ale lui G sunt împerecheate.

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

- t este bine definită deoarece asignarea este definită global în fiecare componentă tare conexă și:

- x_i și \bar{x}_i sunt în componente diferite;
 - dacă, pentru $u \in V(G_j)$, $t(\bar{u})$ a fost deja definită, atunci $\bar{u} \in V(G_k)$ cu $k \geq j$ și $\bar{v} \in V(G_k)$, pentru orice $v \in V(G_j)$ (de ce?).

Exercitii rezolvate (partial)

Exercitiul 9. Solutie.

- MAX 2SAT \in NP (de ce?). Vom arăta că $3\text{SAT} \leqslant_P \text{MAX 2SAT}$:
 - Fie $U = \{u_1, u_2, \dots, u_n\}$, ($n \in \mathbb{N}^*$), $C = C_1 \wedge C_2 \dots \wedge C_m$ ($m \in \mathbb{N}^*$) with $C_i = v_{i_1} \vee v_{i_2} \vee v_{i_3}$, $i = \overline{1, m}$, (unde $\forall i_j, \exists h \in \{1, 2, \dots, n\}$ s.t. $v_{i_j} = u_h$ or $v_{i_j} = \bar{u}_h$) o instanță of the 3SAT problem.
 - Definim următoarea instanță pentru MAX 2SAT: $k = 7m$, $U' = U \cup \{x_1, x_2, \dots, x_m\}$, $C' = \bigwedge_{j=1}^m C'_j$, unde

$$C'_j = x_j \wedge v_{j_1} \wedge v_{j_2} \wedge v_{j_3} \wedge (\bar{v}_{j_1} \vee \bar{v}_{j_2}) \wedge (\bar{v}_{j_2} \vee \bar{v}_{j_3}) \wedge (\bar{v}_{j_3} \vee \bar{v}_{j_1}) \wedge$$

$$\wedge (v_{j_1} \vee \bar{x}_j) \wedge (v_{j_2} \vee \bar{x}_j) \wedge (v_{j_3} \vee \bar{x}_j).$$

- Dacă C este satisfăcută de către o asignare $t : U \rightarrow \{\text{true}, \text{false}\}$, atunci cel puțin un literal din $\{v_{j_1}, v_{j_2}, v_{j_3}\}$ este true, pentru orice $1 \leq j \leq m$.

- În toate trei cazurile există o asignare pentru x_j astfel ca exact șapte clauze "mici" (corespunzând lui C_j) să fie true:
 - dacă $t(v_{j_1}) = t(\bar{v}_{j_2}) = t(\bar{v}_{j_3}) = \text{true}$, definim $t(x_j) = \text{false}$;
 - dacă $t(v_{j_1}) = t(v_{j_2}) = t(\bar{v}_{j_3}) = \text{true}$, definim $t(x_j) = \text{false}$;
 - dacă $t(v_{j_1}) = t(v_{j_2}) = t(v_{j_3}) = \text{true}$, definim $t(x_j) = \text{true}$.
- Reciproc, dacă există $7m$ clauze satisfăcute de o asignare, t , pe U' , atunci există cel mult 7 clauze "mici" (corespunzând lui C_j) care sunt satisfăcute:
 - dacă $t(x_j) = \text{false}$, atunci $t(v_{j_k} \vee \bar{x}_j) = \text{true}$, $\forall k = \overline{1, 3}$ și vor mai fi satisfăcute încă trei sau patru (de ce?).
 - dacă $t(x_j) = \text{true}$, atunci $\alpha = |\{k : t(v_{j_k}) = \text{true}, 1 \leq k \leq 3\}|$ este egal cu $\beta = |\{k : t(v_{j_k} \vee \bar{x}_j) = \text{true}, 1 \leq k \leq 3\}|$, și $\alpha + \gamma \in \{3, 4\}$, unde $\gamma = |\{k : t(\bar{v}_{j_k} \vee \bar{v}_{j_h}) = \text{true}, 1 \leq k < h \leq 3\}|$ (de ce?).

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

Exerciții rezolvate (partial)

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

- Dar, dacă $t(v_{j_1}) = t(v_{j_2}) = t(v_{j_3}) = \text{false}$, atunci cel mult șase dintre cele zece clauze "mici" corespunzând lui C_j sunt satisfăcute de (șase dacă $t(x_j) = \text{true}$ și cinci altfel) (verificați!).
- Deci C este satisfăcută de restricția lui t la U .

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algo 399ns * C. Croitoru - Graph Algorithms *

Exercițiu 10. Soluție.

- " \Rightarrow " Presupunem că instanță lui 3SAT este satisfăcută de $t : U \rightarrow \{\text{true}, \text{false}\}$; atunci $t(x) = \text{false}$ și dacă $t(v_{j_1})$ sau $t(v_{j_2})$ sunt *true*, atunci $t(y_j) = \text{false}$, altfel $t(y_j) = \text{true}$. Răspunsul la NAE-3SAT pentru instanță în cauză va fi affirmative (**de ce?**).
- " \Leftarrow " Presupunem că instanță lui NAE-3SAT este satisfăcută de $t : U \cup \{x\} \cup Y \rightarrow \{\text{true}, \text{false}\}$, unde $Y = \{y_1, y_2, \dots, y_m\}$.
- Atunci instanța NAE-3SAT este satisfăcută și de $\bar{t} : U \cup \{x\} \cup Y \rightarrow \{\text{true}, \text{false}\}$, unde $\bar{t}(v) = \overline{t(v)}$, $\forall v \in U \cup \{x\} \cup Y$.
- Dintre cele două asignări o alegem pe aceea în care x este *false*. Restricția acestei asignări la U satisface instanța 3SAT (**de ce?**).

Exerciții rezolvate (partial)**Exercițiu 11. Soluție.**

- Construcția lui G se face în $\mathcal{O}(n + m)$.
- Arătăm că $C = C_1 \wedge C_2 \dots \wedge C_m$ este satisfiabilă aşa încât fiecare clauză are și un literal *true* și unul *false* dacă și numai dacă G are o tăietură de pondere cel puțin $10mn + 2m$.
- " \Rightarrow " Presupunem că $C = C_1 \wedge C_2 \dots \wedge C_m$ este satisfăcută aşa încât fiecare clauză are și un literal *true* și unul *false* de către asignarea t .
- Definim următoarea bipartiție pe $V(G)$: $S = \{v : t(v) = \text{false}\}$, $T = V(G) \setminus S = \{v : t(v) = \text{true}\} \cup \{y_1, y_2, \dots, y_m\}$.
- $u_i \bar{u}_i \in A = \{xy \in E(G) : x \in S, y \in T\}$ pentru orice $1 \leq i \leq n$ (**de ce?**).
- Pentru orice clauză C_j , există două muchii în A : dacă doar v_{j_k} este *true*, atunci $v_{j_k} v_{j_h}, v_{j_k} v_{j_l} \in A$, dacă v_{j_k} și v_{j_h} sunt *true*, atunci $v_{j_k} v_{j_l}, v_{j_h} v_{j_l} \in A$. Toate aceste 400 muchii din A au ponderea $10mn + 2m$.

- "⇒" Fie $A = \{xy \in E(G) : x \in S, y \in T\}$ tăietură de pondere cel puțin $10mn + 2m$.
- Fie $1 \leq i \leq n$, if $u_i \bar{u}_i \notin A$, atunci $c(A) \leq (n-1)10m + 3m < 10nm + 2m$ - contradicție (de ce?).
- Dacă, pentru un j , $|A \cap \{v_{j_1}v_{j_2}, v_{j_2}v_{j_3}, v_{j_3}v_{j_1}\}| = |A_j| \neq 2$, atunci $A_j = 0$ și $c(A) < 10nm + m$ - din nou o contradicție.
- Astfel, pentru orice $1 \leq j \leq m$, există un $k \in \{1, 2, 3\}$ astfel ca $A_j = \{v_{j_k}v_{j_h}, v_{j_k}v_{j_l}\}$.
- Cum (S, T) separă literalii pozitivi de cei negativi, putem defini asignarea: $t(v) = \text{true}$, pentru orice $v \in S$ care satisface toate clauzele (de ce?) și fiecare clauză va conține un literal false și unul true (de ce?).

11 CURS 11: Reduceri polinomiale pentru probleme de decizie pe grafuri. Abordari ale problemelor NP-hard pe grafuri.

Cuplaj -> multime de muchii neadiciente(nu au noduri in comun)

Cuplaj perfect -> cuplaj care contine toate nodurile disponibile

Cuplaj perfect cu pondere maxima/minima -> suma de pe muchiile din cuplaj este maxima/minima

Parcurs eulerian inchis -> circuit care trece prin toate muchiile o singura data

Circuit hamiltonian -> trece prin toate nodurile o singura data

Colorare greedy:

-> mergem in ordinea aia care se da si plecam de la culoarea 1

-> vedem ce culori au vecinii deja colorati si punem cel mai mic numar disponibil

Colorare optima: cel mai mic numar de culori posibil

Algoritmica Grafurilor - Cursul 11

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

18 decembrie 2020

Cuprins

- 1 **Reduceri în timp polinomial pentru probleme pe grafuri**
 - Probleme Hamiltoniene
 - Problema comis-voiajorului
- 2 **Abordări ale problemelor NP-hard**
 - TSP metrică: Algoritmul lui Christofides
 - Colorarea nodurilor: Algoritmul de colorare greedy
- 3 **Exerciții pentru seminarul din următoarea săptămână**
- 4 **Exerciții rezolvate (parțial)**

Teorema 1

(Karp, 1972) SM \leqslant_P CH.

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

Demonstrație. Fie $G = (V, E)$ și $j \in \mathbb{N}$ o instanță a problemei SM. Vom construi în timp polinomial (relativ la $n = |V|$) un graf H astfel încât există o mulțime stabilă S în G cu $|S| \geq j$ dacă și numai dacă H este Hamiltonian.

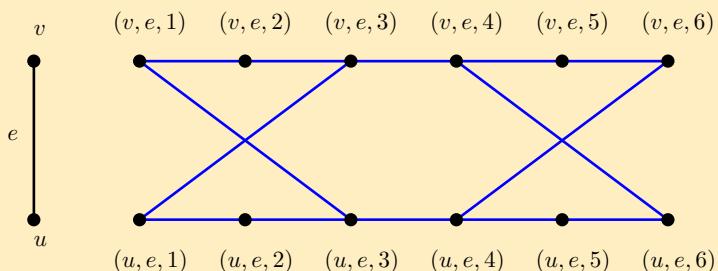
Fie $k = |V| - j$. Să presupunem că $k > 0$ (pentru a evita cazurile banale)

- (i) Fie $A = \{a_1, a_2, \dots, a_k\}$ o mulțime de k de noduri distincte.
- (ii) Pentru fiecare muchie $e = uv \in E(G)$, considerăm graful $G'_e = (V'_e, E'_e)$ cu $V'_e = \{(w, e, i) : w \in \{u, v\}, i = \overline{1, 6}\}$ și $E'_e = \{(w, e, i)(w, e, i+1) : w \in \{u, v\}, i = \overline{1, 5}\} \cup \{(u, e, 1)(v, e, 3), (u, e, 3)(v, e, 1), (u, e, 4)(v, e, 6), (u, e, 6)(v, e, 4)\}$.

Reduceri în timp polinomial - Probleme Hamiltoniene

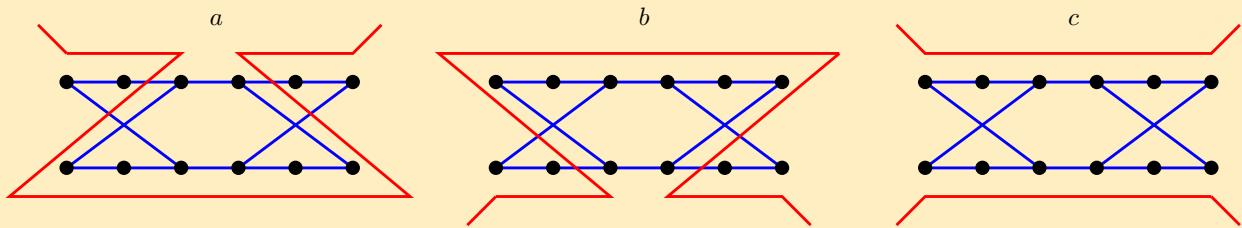
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Demonstrație (continuare).



Graful are proprietatea că, dacă este un subgraf inducător al unui graf Hamiltonian, H , și nici unul dintre nodurile (w, e, i) cu $w \in \{u, v\}$ și $i = \overline{2, 5}$ nu are alți vecini în H , atunci singurele posibilități de parcursere a lui G'_e pe un circuit Hamiltonian sunt:

Graph Algorithms * C. Croitoru - Graph Algo404ns * C. Croitoru - Graph Algorithms *

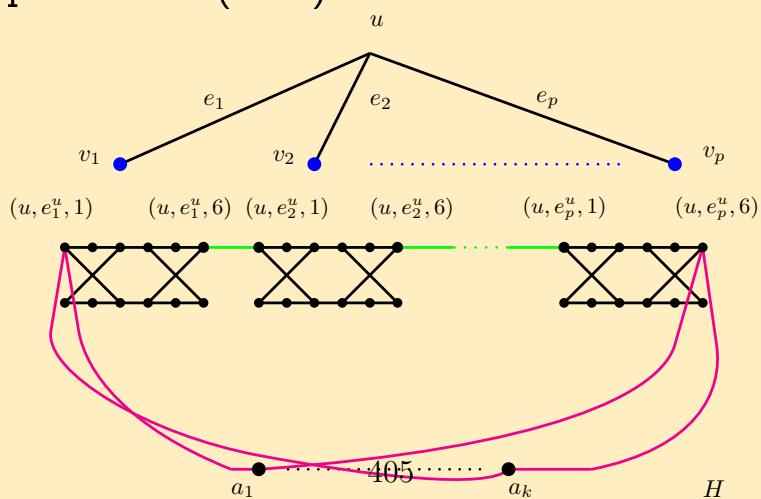


Demonstrație (continuare). Astfel, dacă circuitul Hamiltonian “intră” în G'_e printr-un nod corespunzând lui u ($(u, e, 1)$ sau $(u, e, 6)$) atunci “părăsește” graful G'_e de asemenei printr-un nod corespunzând lui u .

- (iii) Pentru fiecare nod $u \in V$, considerăm (într-o ordine arbitrară) muchiile incidente în G cu u : $e_1^u = uv_1, e_2^u = uv_2, \dots, e_p^u = uv_p$ ($p = d_G(u)$). Fie $E''_u = \{(u, e_i^u, 6)(u, e_{i+1}^u, 1) : i = \overline{1, p-1}\}$ și $E'''_u = \{a_i(u, e_1^u, 1), a_i(u, e_p^u, 6) : i = \overline{1, k}\}$

Reduceri în timp polinomial - Probleme Hamiltoniene

Demonstrație (continuare). Graful H are $V(H) = A \cup \left(\bigcup_{e \in E} V'_e \right)$ și $E(H) = \left(\bigcup_{e \in E} E'_e \right) \cup \left(\bigcup_{u \in V} (E''_u \cup E'''_u) \right)$. Evident, poate fi construit din G în timp polinomial (în n).



Demonstrație (continuare). Acum arătăm că există o mulțime stabilă în G cu cel puțin j noduri dacă și numai dacă H este Hamiltonian.

“ \Leftarrow ” Dacă H este Hamiltonian, atunci există C un circuit Hamiltonian în H . Deoarece A este o mulțime stabilă în H , A descompune circuitul C în exact k drumuri intern disjuncte: $D_{a_{i_1} a_{i_2}}, D_{a_{i_2} a_{i_3}}, \dots, D_{a_{i_k} a_{i_1}}$.

Fie $D_{a_{i_j} a_{i_{j+1}}}$ un astfel de drum ($j+1 = 1 + (j \pmod k)$). Din construcția lui H , urmează că primul nod după a_{i_j} pe acest drum va fi $(v_{i_j}, e_1^{v_{i_j}}, 1)$ sau $(v_{i_j}, e_p^{v_{i_j}}, 6)$, unde $p = d_G(v_{i_j})$, $v_{i_j} \in V$.

După aceasta, $D_{a_{i_j} a_{i_{j+1}}}$ va intra în componenta G'_{e_1} sau G'_{e_p} care va fi părăsită printr-un nod corespunzător lui v_{i_j} . Dacă următorul nod nu este $a_{i_{j+1}}$, va intra în componenta corespunzând următoarei muchii incidente cu v_{i_j} , care va fi părăsită de asemenei printr-un nod corespunzător lui v_{i_j} .

Reduceri în timp polinomial - Probleme Hamiltoniene

Demonstrație (continuare). Urmează că fiecare drum $D_{a_{i_j} a_{i_{j+1}}}$ corespunde unui singur nod $v_{i_j} \in V$, astfel încât prima și ultima muchie a lui $D_{a_{i_j} a_{i_{j+1}}}$ sunt $a_{i_j}(v_{i_j}, e_t^{v_{i_j}}, x)$, $a_{i_{j+1}}(v_{i_j}, e_{t'}^{v_{i_j}}, x')$, cu $t = 1$ și $t' = d_G(v_{i_j})$, $x = 1$, $x' = 6$ sau $t = d_G(v_{i_j})$, $t' = 1$, $x = 6$, $x' = 1$.

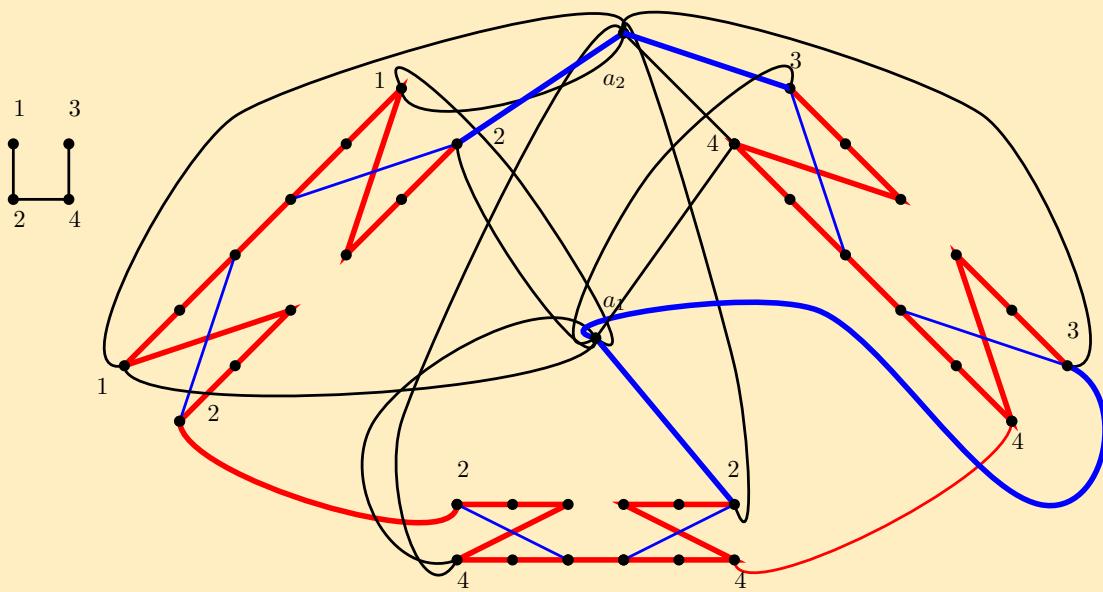
Urmează că nodurile v_{i_j} sunt distințte.

Fie $V^* = \{v_{i_1}, v_{i_2}, \dots, v_{i_k}\}$. Deoarece C este un circuit Hamiltonian în H , urmează că, $\forall e \in E$, există un drum $D_{a_{i_j} a_{i_{j+1}}}$ care traversează G'_{e_j} , astfel există $v \in V^*$ incident cu e .

Deci $S = V \setminus V^*$ este o mulțime stabilă în G și $|S| = j$.

Astfel, am arătat că, dacă H este Hamiltonian, atunci în G există o mulțime stabilă cu j noduri și răspunsul la SM pentru instanța G, j este da.

Demonstrație (continuare).



Reduceri în timp polinomial - Probleme Hamiltoniene

Demonstrație (continuare). “ \Rightarrow ” Să presupunem că răspunsul la SM pentru instanța G, j este da, astfel există S_0 o mulțime stabilă în G cu $|S_0| \geq j$. Există $S \subseteq S_0$ cu $|S| = j$. Fie $V^* = V \setminus S = \{v_1, v_2, \dots, v_k\}$. Considerăm în H pentru fiecare $e = uv \in E$:

- cele două drumuri din cazul (c) în G'_e (desenate mai sus) dacă $u, v \in V^*$.
- drumul din cazul (b) în G'_e (desenat mai sus) dacă $u \in V^*$ și $v \notin V^*$.
- drumul din cazul (a) în G'_e (desenat mai sus) dacă $u \notin V^*$ și $v \in V^*$.

La reuniunea tuturor acestor drumuri adăugăm muchiile $a_i(v_i, e_1^{v_i}, 1), (v_i, e_1^{v_i}, 6)(v_i, e_2^{v_i}, 1), \dots, (v_i, e_{p-1}^{v_i}, 6)(v_i, e_p^{v_i}, 1), (v_i, e_p^{v_i}, 6)a_{i+1}$, (cu $p = d_G(v_i)$), pentru $i = \overline{1, k}$.

Ceea ce se obține este un circuit Hamiltonian în H . \square

Traveling Salesman Problem - TSP Dat $G = (V, E)$ un graf și $d : E \rightarrow \mathbb{R}_+$ o funcție de pondere nenegativă pe muchiile sale, să se determine un circuit Hamiltonian, H_o , a. i. suma ponderilor pe muchiile lui H_o să fie minimă (printre toate circuitele Hamiltoniene ale lui G).

Fie graful G poate fi o rețea constând dintr-o mulțime, V , de orașe împreună cu o mulțime, E , de rute directe între orașe, funcția d oferind, pentru fiecare muchie $uv \in E$, $d(uv) =$ distanța pe ruta directă între orașele u și v . Fixând un oraș de start v_0 , circuitul Hamiltonian H_o reprezintă cel mai scurt traseu care vizitează toate orașele exact odată (cu excepția lui v_0) pe care îl poate parurge un comis-voiajor plecând din v_0 și întorcându-se în v_0 .

Aceasta este probabil, cea mai studiată problemă de optimizare NP-hard!

Reduceri în timp polinomial - Problema comis-voiajorului (TSP)

Considerăm următoarea formulare echivalentă a acestei probleme.

TSP Dat $n \in \mathbb{N}$ ($n \geq 3$) și $d : E(K_n) \rightarrow \mathbb{R}_+$, să se determine un circuit Hamiltonian, H_o , în K_n , cu $d(H_o)$ minim printre toate circuitele Hamiltoniene ale lui K_n , unde

$$d(H_o) = \sum_{e \in E(H_o)} d(e).$$

Dacă graful G , pe care trebuie să rezolvăm TSP, nu este graful complet K_n , atunci putem introduce muchiile lipsă cu o pondere foarte mare, $M \in \mathbb{R}_+$, unde $M > |V| \cdot \max_{e \in E(G)} d(e)$.

Aceasta este formularea problemei TSP simetrice, o problemă similară (asimetrică) poate fi considerată pentru cazul când G este un digraf.

În studiul complexității timp a acestei probleme, vom considera $d(e) \in \mathbb{N}$, pentru fiecare muchie e .

Problema de decizie asociată este

DTSP

Instanță: $n \in \mathbb{N}$ ($n \geq 3$), $d : E(K_n) \rightarrow \mathbb{N}$ și $B \in \mathbb{N}$.

Întrebare: Există un circuit Hamiltonian, H_o , în K_n , a. i. $d(H_o) \leq B$?

Teorema 2

$\text{CH} \leq_P \text{DTSP}$.

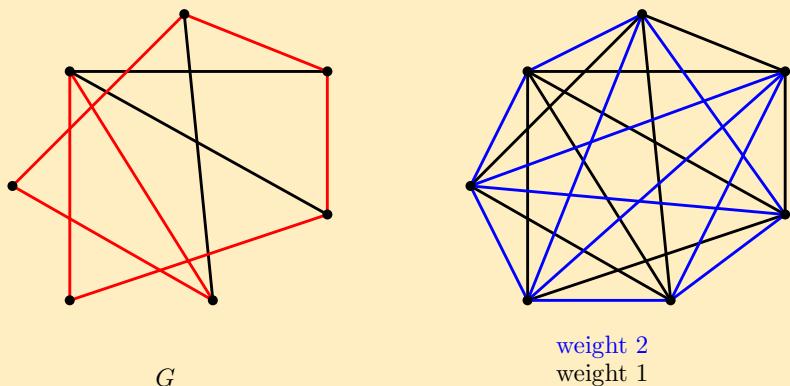
Demonstrație. Fie $G = (V, E)$ ($|V| = n$) o instanță a problemei CH. Construim în timp polinomial o instanță, $d : E(K_n) \rightarrow \mathbb{N}$ și $B \in \mathbb{N}$, a problemei DTSP astfel încât există un circuit Hamiltonian în K_n de pondere totală cel mult B dacă și numai dacă G este graf Hamiltonian.

Reduceri în timp polinomial - Problema comis-voiajorului (TSP)

Fie

$$d(vw) = \begin{cases} 1, & \text{dacă } vw \in E(G) \\ 2, & \text{dacă } vw \in E(\overline{G}) \end{cases} \text{ și } B = n.$$

Atunci, există în K_n un circuit Hamiltonian de pondere $\leq n$ dacă și numai dacă există un circuit Hamiltonian C în K_n astfel încât $\forall e \in E$ $d(e) = 1$, adică, dacă și numai dacă G are un circuit Hamiltonian:



Urmează că TSP este o problemă NP-hard. \square

O abordare posibilă este să considerăm un algoritm de aproximare \mathcal{A} , care să construiască, în timp polinomial, pentru fiecare instanță TSP, un circuit Hamiltonian $H_{\mathcal{A}}$ al lui K_n , o aproximare a soluției optime H_o . Calitatea aproximării poate fi exprimată folosind următorul raport:

$$R_{\mathcal{A}}(n) = \sup_{d: E(K_n) \rightarrow \mathbb{R}_+, d(H_o) \neq 0} \frac{d(H_{\mathcal{A}})}{d(H_o)}$$

$$R_{\mathcal{A}} = \sup_{n \geq 3} R_{\mathcal{A}}(n).$$

Evident, algoritmul de aproximare \mathcal{A} este util numai dacă $R_{\mathcal{A}}$ este finită. Din nefericire, dacă funcția de pondere d este arbitrară, condiția ca $R_{\mathcal{A}}$ să fie finită este la fel de dificilă ca și rezolvarea exactă a TSP. Mai precis, avem următorul rezultat:

Reduceri în timp polinomial - Problema comis-voiajorului (TSP)

Teorema 3

Dacă există un algoritm de aproximare în timp polinomial A pentru TSP a. î. $R_{\mathcal{A}} < \infty$, atunci problema CH poate fi rezolvată în timp polinomial.

Demonstrație. Fie A un algoritm de aproximare în timp polinomial cu $R_{\mathcal{A}} < \infty$. Există $k \in \mathbb{N}$ astfel încât $R_{\mathcal{A}} \leq k$.

Fie $G = (V, E)$ un graf arbitrar, instanță a CH. Dacă $n = |V|$, atunci considerăm $d : E(K_n) \rightarrow \mathbb{N}$ definită prin

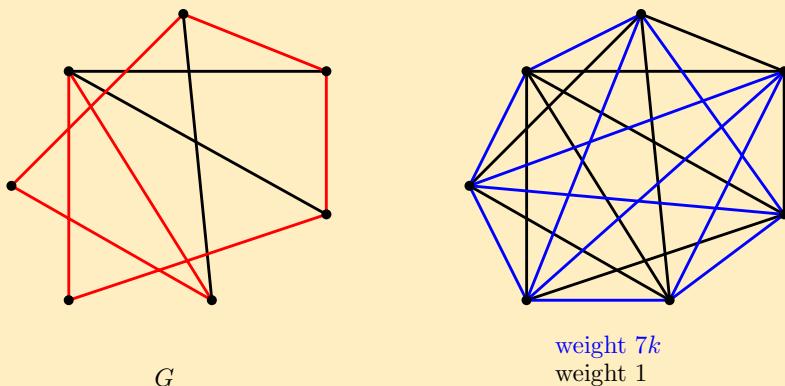
$$d(uv) = \begin{cases} 1, & \text{dacă } uv \in E(G) \\ kn, & \text{dacă } uv \notin E(G) \end{cases}.$$

Evident, G este Hamiltonian dacă și numai dacă H_o , soluția optimă a acestei instanțe a TSP, satisface $d(H_o) = n$.

Aplicăm \mathcal{A} pentru a rezolva aproximativ această instanță a TSP.

- Dacă $d(H_{\mathcal{A}}) \leq kn$, atunci $d(H_{\mathcal{A}}) = n$ și $H_{\mathcal{A}}$ este optim.
- Dacă $d(H_{\mathcal{A}}) > kn$, atunci $d(H_o) > n$. Într-adevăr, presupunând că $d(H_o) = n$, avem $\frac{d(H_{\mathcal{A}})}{d(H_o)} \leq k$, astfel $d(H_{\mathcal{A}}) \leq kd(H_o) = kn$, contradicție.

Urmează că G este Hamiltonian dacă și numai dacă $d(H_{\mathcal{A}}) \leq kn$, și, deoarece \mathcal{A} rulează în timp polinomial, urmează că CH poate fi rezolvată în timp polinomial. \square



Reduceri în timp polinomial - Problema comis-voiajorului (TSP)

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

Teorema 3 poate fi formulată echivalent astfel:

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

Teoremă

Dacă $P \neq NP$, atunci nu există algoritm de aproximare în timp polino-mial \mathcal{A} cu $R_{\mathcal{A}} < \infty$.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
 Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
 C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algo411ns * C. Croitoru - Graph Algorithms *

Teorema 4

(Christofides,1976) Dacă în TSP funcția de pondere d satisfacă

$$\forall u, v, w \in V(K_n) \text{ distințe}, d(uv) \leq d(uw) + d(wv),$$

atunci există un algoritm polinomial de aproximare \mathcal{A} cu $R_{\mathcal{A}} = 3/2$.

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

Demonstrație. Fie \mathcal{A} următorul algoritm:

- Determină T^0 , mulțimea de muchii a unui arbore parțial de cost minim din K_n (costul unei muchii e va fi $d(e)$) (acest pas ia un timp polinomial folosind orice algoritm pentru MST).
- Determină M^0 , un cuplaj perfect de pondere minimă în subgraful induș în K_n de mulțimea de noduri de grad impar din arborele parțial de cost minim T^0 (acest pas ia un timp polinomial folosind orice algoritm pentru determinarea unui cuplaj de cardinal maxim).

Abordări ale problemelor NP-hard - Algoritmul lui Christofides

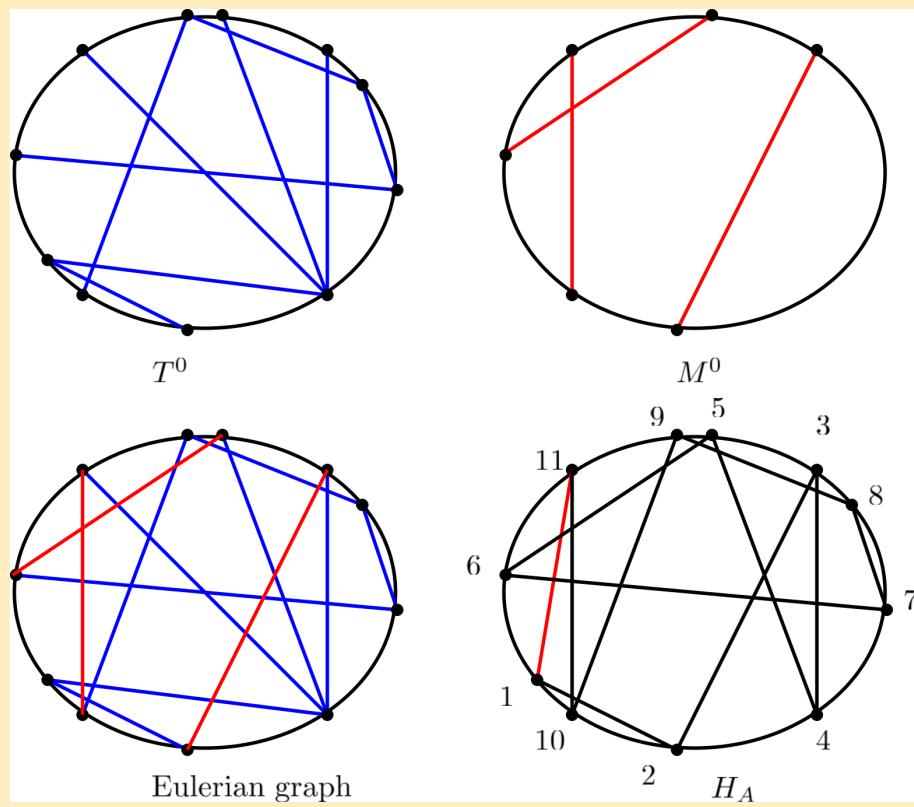
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstrație (continuare)

- În multigraful obținut din $\langle T^0 \cup M^0 \rangle_{K_n}$, prin duplicarea muchiilor din $T^0 \cap M^0$ (este conex și are toate nodurile de grad par) determină un parcurs Eulerian închis, $(v_{i_1}, v_{i_2}, \dots, v_{i_n})$. Eliminăm toate aparițiile multiple ale nodurilor interne pentru a obține un circuit Hamiltonian $H_{\mathcal{A}}$ în K_n cu mulțimea de muchii $H_{\mathcal{A}} = \{v_{j_1}v_{j_2}, v_{j_2}v_{j_3}, \dots, v_{j_n}v_{j_1}\}$ (amândouă construcțiile se fac în $\mathcal{O}(n^2)$, parcursul Eulerian închis poate fi găsit cu algoritmul lui Hierholzer).

Fie $H_{\mathcal{A}}$ soluția aproximativă a TSP dată de algoritmul lui Christofides. Fie $m = \lfloor n/2 \rfloor$ și H_o o soluție optimă. Arătăm (după Cornuejols & Nemhauser) că

$$\forall n \geq 3, d(H_{\mathcal{A}}) \leq \frac{3m-1}{2m} d(H_o).$$



Abordări ale problemelor NP-hard - Algoritmul lui Christofides

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

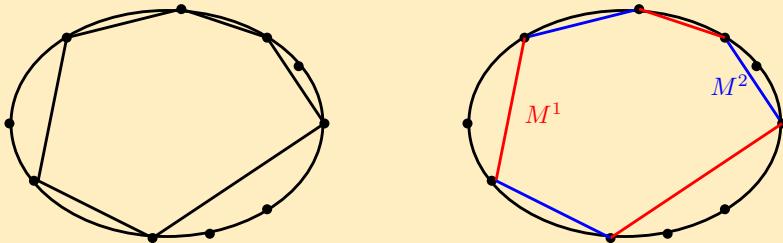
Demonstrație (continuare) Fie $H_o = \{v_1 v_2, v_2 v_3, \dots, v_n v_1\}$ (dacă e necesar, redenumim nodurile).

Fie $W = \{v_{i_1}, v_{i_2}, \dots, v_{i_{2k}}\}$ mulțimea nodurilor de grad impar din $\langle T^0 \rangle_{K_n}$, $i_1 < i_2 < \dots < i_{2k}$. Fie $H = \{v_{i_1} v_{i_2}, v_{i_2} v_{i_3}, \dots, v_{i_{2k-1}} v_{i_{2k}}, v_{i_{2k}} v_{i_1}\}$ circuitul generat de W în K_n . Aplicând în mod repetat inegalitatea triunghiulară, obținem $d(H) \leq d(H_o)$, (ponderea fiecărei corzi, $d(v_{i_j} v_{i_{j+1}})$ este majorată de suma ponderilor de pe muchiile lui H_o care unesc extremitățile corzii $v_{i_j} v_{i_{j+1}}$).

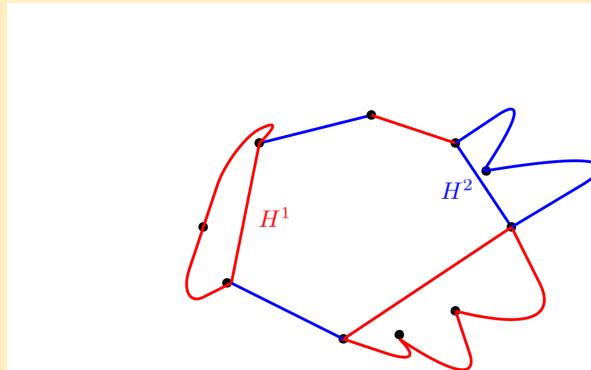
Deoarece H este un circuit par, poate fi scris ca reuniunea a două cuplaje perfecte în $[W]_{K_n}$, $M^1 \cup M^2$. Să presupunem că $d(M^1) \leq d(M^2)$.

Din modul de alegere al lui M^0 , avem $d(M^0) \leq d(M^1) \leq (1/2)[d(M^1) + d(M^2)] = (1/2)d(H) \leq (1/2)d(H_o)$. Fie $\alpha \in \mathbb{R}_+$ a. î. $d(M^0) = \alpha d(H_o)$. Evident, $0 < \alpha \leq 1/2$.

Demonstrație (continuare)



Descompunem H_o în $H^1 \cup H^2$ luând în H^i muchiile din H_o care conectează extremitățile fiecărei corzi din M^i : $(v_{i_j} v_{i_j+1} \in M^i \Rightarrow v_{i_j} v_{i_j+1}, \dots, v_{i_j+1-1} v_{i_j+1} \in H^i)$.



Abordări ale problemelor NP-hard - Algoritmul lui Christofides

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

Demonstrație (continuare) Din inegalitatea triunghiulară, $d(H^i) \geq d(M^i)$, $i = 1, 2$. Cel puțin unul dintre H^1 sau H^2 are cel mult $m = \lfloor n/2 \rfloor$ muchii. Să presupunem că H^1 are această proprietate. Deoarece $d(H^1) \geq d(M^1) \geq d(M^0) = \alpha d(H_o)$, urmează că există $e \in H^1$ astfel încât $d(e) \geq (\alpha/m)d(H_o)$.

Fie T un arbore parțial obținut din H_o prin ștergerea unei muchii de pondere maximă. Avem $d(T) = d(H_o) - \max_{e \in E(H_o)} d(e) \leq d(H_o) - (\alpha/m)d(H_o)$.

Deoarece T^0 este arbore parțial de cost minim în K_n urmează că $d(T^0) \leq d(H_o)(1 - \alpha/m)$.

Folosind inegalitatea triunghiulară obținem

$$\begin{aligned} d(H_A) &\leq d(T^0) + d(M^0) \leq d(H_o) \left(1 - \frac{\alpha}{m}\right) + \alpha d(H_o) = \\ &= \left(1 + \frac{\alpha(m-1)}{m}\right) d(H_o) \stackrel{\alpha \leq 1/2}{\ll} \frac{3m-1}{2m} d(H_o), \forall n \geq 3. \quad \square \end{aligned}$$

Fie $G = (V, E)$ un graf, $V = \{1, 2, \dots, n\}$ și fie π o permutare a lui V .

Construim o colorare a nodurilor $c : V \rightarrow \{1, \dots, \chi(G, \pi)\}$.

```

 $c(\pi_1) \leftarrow 1; \chi(G, \pi) \leftarrow 1; S_1 \leftarrow \{\pi_1\};$ 
for ( $i = \overline{2, n}$ ) do
     $j \leftarrow 0;$ 
    repeat
         $j++;$   $v \leftarrow$  primul nod (în  $\pi$ ), din  $S_j$  a. î.  $\pi_i v \in E(G)$ ;
        if ( $\exists v$ ) then
             $first(\pi_i, j) \leftarrow v;$ 
        else
             $first(\pi_i, j) \leftarrow 0;$   $c(\pi_i) \leftarrow j;$   $S_j \leftarrow S_j \cup \{\pi_i\};$ 
        end if
    until ( $first(\pi_i, j) = 0$  sau  $j = \chi(G, \pi)$ )
    if ( $first(\pi_i, j) \neq 0$ ) then
         $c(\pi_i) \leftarrow j + 1;$   $S_{j+1} \leftarrow \{\pi_i\}; \chi(G, \pi) \leftarrow j + 1;$ 
    end if
end for

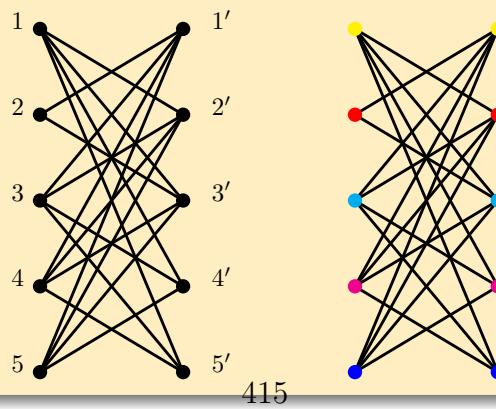
```

Abordări ale problemelor NP-hard - Algoritmul de colorare greedy

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

Să observăm că numărul de culori folosite de algoritm nu este mai mare decât $1 + \Delta(G)$. Urmează că $\chi(G) \leq 1 + \Delta(G)$.

$\chi(G, \pi)$ numărul de culori returnate de algoritmul de colorare greedy, poate fi cu oricât mai mare decât $\chi(G)$. De exemplu, fie G graful obținut din graful complet bipartit $K_{n,n}$, cu multimea de noduri $\{1, 2, \dots, n\} \cup \{1', 2', \dots, n'\}$, prin stergerea muchiilor $11', 22', \dots, nn'$.



Dacă $\pi = (1, 1', 2, 2', \dots, n, n')$, atunci algoritmul de colorare greedy returnează $c(1) = c(1')$, $c(2) = c(2') = 2$, $c(n) = c(n') = n$. Astfel, $\chi(G, \pi) = n$, pe când $\chi(G) = 2$.

Pe de altă parte, pentru orice graf G , există o permutare π a nodurilor sale astfel încât $\chi(G, \pi) = \chi(G)$.

Într-adevăr, fie $S_1, S_2, \dots, S_{\chi(G)}$ clasele de colorare ale unei colorări optimale, astfel încât fiecare S_i este o mulțime stabilă maximală (relativ la incluziune) în $G - \bigcup_{j=1}^{i-1} S_j$. Fie π o permutare care induce o ordonare

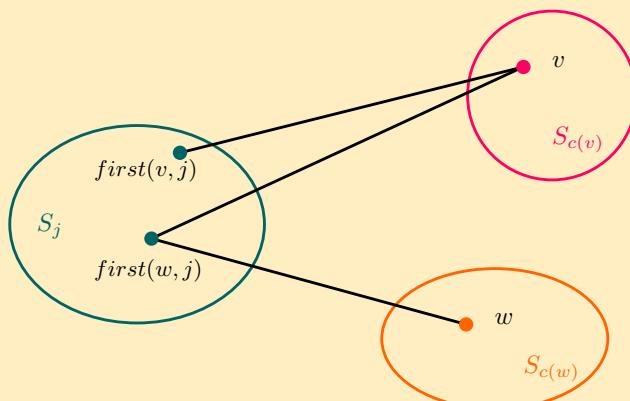
astfel încât nodurile apar în ordinea crescătoare a culorilor lor. Atunci, $\chi(G, \pi) = \chi(G)$.

O condiție suficientă pentru corectitudinea algoritmului de colorare greedy:

Abordări ale problemelor NP-hard - Algoritmul de colorare greedy

Teoremă

Dacă, $\forall vw \in E$ și $\forall j < \min\{c(v), c(w)\}$ astfel încât $\text{first}(v, j) < \text{first}(w, j)$ în ordonarea dată de π , avem $\text{first}(w, j)v \in E$, atunci $\chi(G, \pi) = \chi(G)$.



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

Condiția de mai sus poate fi testată în $\mathcal{O}(m)$ ca un ultim pas al algoritmului. Dacă este îndepință, atunci avem un certificat pentru optimialitatea numărului găsit de culori.

Teorema de mai sus se poate demonstra arătând că dacă condiția enunțată are loc, atunci $\chi(G, \pi) = \omega(G) \leq \chi(G)$. Pe de altă parte, se știe că există grafuri G pentru care $\chi(G) - \omega(G)$ este arbitrar de mare; pentru un astfel de graf G nici o permutare π nu satisface condiția din teoremă.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 din următoarea săptămână

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

Exercițiul 1. Regiunile (inclusiv cea infinită) formate de n cercuri în plan pot fi colorate cu două culori astfel ca două regiuni care au în comun un arc de cerc (nedegenerat) să fie colorate diferit.

Exercițiul 2. Fie $G = (V, E)$ un graf fără circuite impare disjuncte. Arătați că G este 5-colorabil.

Exercițiul 3. Pentru un graf dat H definim gradul mediu ca $ad(H) = \frac{2|E(H)|}{|V(H)|}$. Pentru un graf G , gradul mediu maxim este

$$mad(G) = \max \{ ad(H) : H \text{ subgraf inducator al lui } G \}$$

Fie $k \geq 3$ un întreg. Arătați că, dacă un graf G are numărul cromatic strict mai mare decât k iar gradul său mediu maxim cel mult k , atunci G conține un subgraf inducator k -regulat.

Exercițiu 4.

- (a) Arătați că orice graf poate fi colorat (pe noduri) cu $\Delta(G) + 1$ culori.
 (b) Considerăm următorul algoritm recursiv pentru colorarea nodurilor unui graf 3-colorabil cu n noduri:

```

color( $G$ ) {
  if ( $\Delta(G) \leq \sqrt{n}$ ) then
    colorează toate nodurile lui  $G$  cu  $(\Delta(G) + 1)$  culori noi;
  else
    fie  $x_0 \in V(G)$  a. i.  $d_G(x_0) = \Delta(G)$ ;
    colorează  $x_0$  cu o culoare nouă;
    colorează nodurile lui  $[N_G(x_0)]_G$  cu două culori noi;
     $G \leftarrow G - (\{x_0\} \cup N_G(x_0))$ ;
    color( $G$ );
  end if }
```

Arătați că algoritmul de mai sus folosește $\mathcal{O}(\sqrt{n})$ culori.

Exerciții pentru seminarul din următoarea săptămână**Exercițiu 5.** Considerăm următoarea problemă:**Set Cover**

Instanță: O mulțime nevidă X , o familie de submulțimi ale lui X : $\mathcal{F} = \{X_1, \dots, X_n\}$ și $k \in \mathbb{N}^*$.

Întrebare: X poate fi acoperit cu cel mult k submulțimi din \mathcal{F} ?

Considerăm următoarea heuristică (de tip greedy) pentru rezolvarea acestei probleme

```

 $\mathcal{F}' \leftarrow \emptyset$ ;
while ( $\exists x \in X$  neacoperit de  $\mathcal{F}'$ ) do
  fie  $X_i$  din  $\mathcal{F}$  cu un număr maxim de elemente neacoperite;
   $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{X_i\}$ ;
end while
return  $\mathcal{F}'$ ;
```

Arătați că dacă m este cardinalul unei subfamilii optime a lui \mathcal{F} , atunci algoritmul de mai sus returnează $\mathcal{O}(\frac{m}{k} \ln n)$ subfamilie cu cel mult $m \ln n$ submulțimi ($n = |X|$).

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

Exercițiu 6. Fie $G = (V, E)$ un graf cu n noduri și m muchii. O ordonare $\{x_1, x_2, \dots, x_n\}$ a nodurilor lui G este k -mărginită dacă în digraful \vec{G} , obținut din G prin înlocuirea fiecărei muchii $x_i x_j$ cu arcul $x_{\min\{i,j\}} x_{\max\{i,j\}}$, avem $d_{\vec{G}}^+(x) \leq k, \forall x \in V$.

- (a) Descrieți un algoritm care să testeze în $\mathcal{O}(n + m)$ dacă G , are o ordonare k -mărginită, pentru un $k \in \mathbb{N}^*$.

(b) Folosiți algoritmul de mai sus pentru a determina în $\mathcal{O}((n + m) \log n)$ numărul

$o(G) = \min\{k \in \mathbb{N} : G \text{ are o ordonare } k\text{-mărginită.}\}$

- (c) Arătați că orice graf G are o colorare a nodurilor cu $\omega(G) + 1$ culori.

Exercitii pentru seminarul din următoarea săptămână

Exercițiu 7. Un algoritm greedy pentru colorarea nodurilor unui graf $G = (V, E)$ este următorul: mai întâi, alege o D -ordonare a nodurilor $V = \{v_1, v_2, \dots, v_n\}$, i. e., $d_G(v_1) \geq d_G(v_2) \geq \dots \geq d_G(v_n)$, apoi v_i primește cea mai mică culoare nefolosită de vecinii săi deja colorați. Considerăm următoarea problemă de decizie

3GCOL

instantă: $G \equiv (V, E)$ un graf.

întrebare: Are G o D -ordonare a. î. euristică de mai sus folosește cel mult 3 culori?

Arătați că $3\text{COL} \leqslant_P 3\text{GCOL}$.

Exercițiu 8. Fie $G = (V; E)$ un graf cu $V = \{1, 2, \dots, n\}$ și $\omega(G) = 2$. Definim un graf nou $M(G)$ considerând reuniune disjunctă a lui G cu $K_{1,n}$ (a cărui bipartiție este $(\{0\}, \{1', 2', \dots, n'\})$) și adăugând toate muchiile $\{i'j, ij' : ij \in E(G)\}$.

- (a) Arătați că $\omega(M(G)) = 2$ și $\chi(M(G)) = \chi(G) + 1$.
- (b) Arătați că, pentru orice $p \in \mathbb{N}^*$, există un graf K_3 -free cu numărul cromatic p .

Exercițiu 9. Fie \mathcal{S} o societate formată din n indivizi. Fiecare individ, $i \in \mathcal{S}$, cunoaște o submulțime $c(i) \subseteq \mathcal{S} \setminus \{i\}$ formată din alți indivizi. Doi indivizi diferiți $i, i' \in \mathcal{S}$ nu pot face parte din același juriu dacă unul dintre ei îl cunoaște pe celălalt (putem avea jurii formate dintr-un singur membru).

Arătați că dacă fiecare individ cunoaște cel mult k alți indivizi ($|c(i)| \leq k, \forall i \in \mathcal{S}$), atunci există o familie de cel mult $(2k + 1)$ jurii disjuncte care acoperă întreaga societate.

Exerciții rezolvate (partial)

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

Exercițiu 1. Soluție.

- Inducție după n :
- Pentru $n = 1$ avem doar două regiuni, să presupunem ca proprietatea este adevărată pentru $n - 1$ cercuri și alegem n cercuri în plan.
- Dacă C este unul dintre aceste cercuri atunci colorăm mai întâi (cu două culori) regiunile obținute prin intersecția celorlalte $n - 1$ cercuri.
- În ceea ce privește cercul C : culorile regiunilor din afara lui C pot rămașe aceleași în timp ce culorile regiunilor interioare lui C se schimbă (de ce?).

Graph Algorithms * C. Croitoru - Graph Algo420ns * C. Croitoru - Graph Algorithms *

Exercițiu 2. Soluție.

- Dacă G nu conține circuite impare, $\chi(G) \leq 2$ (**de ce?**).
- Altfel, fie C un circuit impar induș, $\chi(C) = 3$; $G - C$ este bipartit (**de ce?**).
- Să presupunem că multimea nodurilor sale poate fi partionată: $V \setminus V(C) = S_1 \dot{\cup} S_2$, unde S_1 și S_2 sunt stabile.
- Există o 3-colorare a lui C (**de ce?**), c' iar $c : V \rightarrow \{1, 2, 3, 4, 5\}$, unde

$$c(x) = \begin{cases} c'(x), & x \in V(C) \\ 4, & x \in S_1 \\ 5, & x \in S_2 \end{cases},$$

este o 5-colorare a lui G (**de ce?**).

Exerciții rezolvate (partial)**Exercițiu 3. Soluție.**

- Demonstrăm mai întâi că, dacă un subgraf induș H al lui G are $\delta(H) \geq k$, atunci H este un graf k -regulat:

$$k \geq mad(G) \geq ad(H) = \frac{1}{|H|} \sum_{x \in V(H)} d_H(x) \geq$$

$$\geq \frac{1}{|H|} \sum_{x \in V(H)} \delta(H) = \delta(H) \geq k. \quad (\text{de ce?})$$

- Să presupunem prin reducere la absurd că pentru orice subgraf induș H al lui G avem $\delta(H) < k$. Ordăm nodurile lui G astfel:

```

 $i \leftarrow n;$ 
 $H \leftarrow G;$ 
while( $|H| \geq 1$ ){
    let  $x \in V(H)$  s.t.  $d_H(x) = \delta(H)$ ; //  $d_H(x) < k$ 
     $v_i \leftarrow x;$ 
     $i --;$ 
     $H \leftarrow H - x;$ 
}

```

- Evident, $|N_G(v_i) \cap \{v_1, v_2, \dots, v_{i-1}\}| \leq k - 1$.
 - Astfel, algoritmul greedy, folosind această ordonare, va oferi o k -colorare a nodurilor lui G (de ce?).

Exerciții rezolvate (partial)

Exercițiu 4. Soluție.

- La fiecare iterație ștergem din graf cel puțin $(\sqrt{n} + 1)$ noduri (de ce?).
 - Până când $\Delta(G)$ devine cel mult \sqrt{n} , astfel, numărul de iterări este mai mic decât $n/\sqrt{n} = \sqrt{n}$ (de ce?).
 - Fiecare astfel de pas utilizează trei culori noi; numărul total de culori va fi cel mult $3\sqrt{n} + \sqrt{n} = \mathcal{O}(\sqrt{n})$.

Exercițiu 5. Soluție.

- Fie k_j numărul de elemente rămase neacoperite după j iterații ($k_0 = n$).
- Aceste k_j elemente pot fi acoperite de o subfamilie optimă a lui \mathcal{F} , \mathcal{F}_0 , cu $|F_0| = m$ (de ce?).
- Astfel una dintre submulțimile din \mathcal{F}_0 are cardinalul cel puțin k_j/m (de ce?), de unde se obține $k_{j+1} \leq k_j - k_j/m = k_j(1 - 1/m)$ și

$$k_j \leq k_o(1 - 1/m)^j, \forall i \geq 1.$$

- Dar $(1 - 1/m)^j \leq \exp(-j/m)$, deoarece $1 - x \leq \exp(-x), \forall x \in \mathbb{R}$.
- Astfel, $k_j < n \exp(-j/m)$ și $\exp(-j/m)$ devine 1 (de unde $k_j < 1$) pentru $j = m \ln n$.

Exerciții rezolvate (partial)

Exercițiu 6. Soluție.

- (a) Reformulăm problema: un graf G are o ordonare k -mărginită dacă și numai dacă îi putem ordona nodurile $V = \{v_{i_1}, v_{i_2}, \dots, v_{i_n}\}$ astfel încât $d_{G - \{v_{i_1}, v_{i_2}, \dots, v_{i_{k-1}}\}}(v_{i_k}) \leq k, \forall 2 \leq k \leq n$. (de ce?)
 - (Aceasta deoarece într-o ordonare k -mărginită primul nod are gradul $\leq k$, iar proprietatea este ereditară - de ce?).
 - Următorul algoritm construiește și întreține multimea S a nodurilor de grad $\leq k$.
 - Stergem nodurile din această multime unul câte unul și decrementăm gradele vecinilor corespunzători.

```
 $\mathcal{S} \leftarrow \emptyset;$  // a list which contains all the vertices having degrees  $\leq k$  in the current graph;  
 $i \leftarrow 0;$   
for  $v \in V(G)$ { // this loop has  $\mathcal{O}(n + m)$  time complexity;  
     $index[v] \leftarrow 0;$   
    find  $d_G(v);$   
     $\mathcal{S}[v] \leftarrow 0;$   
    if  $d_G(v) \leq k$  {  
        push( $\mathcal{S}, v);$   
         $\mathcal{S}[v] \leftarrow 1;$   
    }  
}
```

Algoritms * C. Croitoru - Graph Algorithms *

Exerciții rezolvate (partial)

```
while( $\mathcal{S} \neq \emptyset$ ) { // this loop has  $\mathcal{O}(n + m)$  time complexity;  
     $x \leftarrow \text{pop}(\mathcal{S});$   
     $i \leftarrow i + 1;$   
     $index[x] \leftarrow i;$   
    for  $y \in \mathcal{A}(x)$ { //  $G \leftarrow G - x;$   
        if ( $index[y] == 0$  ){  
             $d_G(y) \leftarrow d_G(y) - 1;$   
            if ( $\mathcal{S}[y] == 0$  and  $d_G(y) \leq k$ ){  
                push( $\mathcal{S}, y);$   
                 $\mathcal{S}[y] \leftarrow 1;$   
            }  
        }  
    }  
}  
return ( $i == n$ );
```

(b) Presupunem că funcția de mai sus se numește $\text{BoundedOrder}(G, k)$ (returnează true dacă și numai dacă G are o ordonare k -mărginită). Următorul algoritm este o căutare binară a lui $o(G)$.

```

if ( $E(G) == \emptyset$ )
    return 0;
 $a \leftarrow ?$ ; // invariant:  $\text{BoundedOrder}(G, a) == \text{false}$ 
 $b \leftarrow ?$ ; // invariant:  $\text{BoundedOrder}(G, b) == \text{true}$ 
while(?)
    if( $\text{BoundedOrder}(G, \left\lceil \frac{a + b}{2} \right\rceil)$ )
         $b \leftarrow \left\lceil \frac{a + b}{2} \right\rceil$ ;
    else
         $a \leftarrow \left\lceil \frac{a + b}{2} \right\rceil$ ;
return  $b$ ;

```

Exerciții rezolvate (partial)

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

(c) Fie $V = \{v_{i_1}, v_{i_2}, \dots, v_{i_n}\}$ o ordonare $o(G)$ -mărginită a lui G . Construim o $o(G) + 1$ colorare a lui G folosind algoritmul de mai jos (de ce?)

for $k = n$ down to 1

$$c(v_{i_k}) \leftarrow \min (\{1, 2, \dots, o(G) + 1\} \setminus \{c(v_{i_j}) : v_{i_k} v_{i_j} \in E(G), j > k\})$$

- (Cum v_{i_k} are cel mult $o(G)$ vecini cu index mai mare putem alege întotdeauna o culoare din mulțimea de culori de mai sus.)

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
 C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algo425ns * C. Croitoru - Graph Algorithms *

Exercițiu 7. Soluție.

- Fie G un graf (an instanță pentru problema 3COL); construim în timp polinomial o instanță H pentru 3GCOL.
- Când $\Delta(G) \in \{0, 1\}$ definim $H = G$; H are o 3-colorare (de ce?) și pentru orice D-ordonare euristică oferă o 3-colorare (de ce?).
- Când $\Delta(G) \geq 2$ definim H aşa încât G este un subgraf induș al lui H , $\Delta(H) = \Delta(G)$, și $d_H(x) = \Delta(G)$, $\forall x \in V(G)$, folosind următorul algoritm:

```

 $H \leftarrow G;$ 
while ( $\exists x \in V(G)$ ,  $d_H(x) < \Delta(G)$ ) {
    let  $x$  such that  $d_H(x) < \Delta(G)$ ;
     $k \leftarrow \Delta(G) - d_H(x)$ ;
     $V(H) \leftarrow V(H) \cup \{v_1^x, v_2^x, \dots, v_k^x\}$ ; // these are new nodes;
     $E(H) \leftarrow E(H) \cup \{xv_1^x, xv_2^x, \dots, xv_k^x\}$ ;
}

```

Exerciții rezolvate (partial)

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

- Instanța H a fost construită în timpul $\mathcal{O}(|V(G)|^2)$ plecând de la instanța G .
- Rămâne să arătăm că G admite o 3-colorare dacă și numai dacă există o D-ordonare of $V(H)$ astfel încât, plecând de la ea, euristică folosește cel mult 3 culori.
- "⇒" Fie $c : V(G) \rightarrow \{1, 2, 3\}$ a 3-colorare of $V(G)$, and $S_i = c^{-1}(i)$, $i = \overline{1, 3}$.
- Definim următoarea D-ordonare: mai întâi sunt nodurile din S_1 , apoi cele din S_2 , apoi cele din S_3 iar la final nodurile pendante adăugate - dacă există astfel de noduri în H .
- Cu această ordonare aplicăm algoritmul greedy de colorare și obținem o colorare c' :

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

- $c'(v) = 1, \forall v \in S_1;$
- $c'(v) \in \{1, 2\}, \forall v \in S_2;$
- $c'(v) \in \{1, 2, 3\}, \forall v \in S_3;$
- $c'(v) \in \{1, 2\}, \forall v \in V(H) \setminus V(G);$
- c' este o 3-colorare a lui $V(H)$ (**de ce?**).
- " \Leftarrow " dacă euristică oferă o 3-colorare a lui $V(H)$, atunci G este un graf 3-colorabil (**de ce?**).

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolvate (partial)

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. Soluție.

- (a) Dacă $M(G)$ conține trei noduri x, y, z care induc un K_3 , atunci trebuie ca $\{x, y\} \subseteq V(G)$ sau $\{x, y\} \subseteq V(K_{1,n})$ (**de ce?**).
- În primul caz $xy \in E(G)$, $z = u'$ și $xu, yu \in E(G)$ ($u \in V(G)$) adică $\omega(G) \geq 3$, în al doilea caz se obține o contradicție similară în $K_{1,n}$. Astfel, $\omega(M(G)) = 2$.
 - Putem presupune fără a restrânge generalitatea că G nu are noduri izolate (**de ce?**).
 - Fie $c : V(G) \rightarrow \{1, 2, \dots, \chi(G)\}$ o colorare optimă a lui G . $\chi(M(G)) \geq \chi(G)$ și $c' : V(M(G)) \rightarrow \mathbb{N}^*$, definită prin $c'(i') = c'(i) = c(i), \forall i \in V(G)$, $c(0) = \chi(G) + 1$, este o colorare a lui $M(G)$ (**de ce?**).

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

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

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

- Deci $\chi(M(G)) \leq \chi(G) + 1$.

- Să presupunem prin reducere la absurd că $\chi(M(G)) = \chi(G)$ și fie $c'_0 : V(M(G)) \rightarrow \{1, 2, \dots, \chi(G)\}$ o colorare optimă a lui $M(G)$, $c'_0(0) = p_0$.

- Definim $c_0 : V(G) \rightarrow \{1, 2, \dots, \chi(G)\} \setminus \{p_0\}$ luând

$$c_0(i) = \begin{cases} c'_0(i), & \text{if } c'_0(i) \neq p_0 \\ c'_0(i'), & \text{if } c'_0(i) = p_0 \end{cases}$$

- c_0 este o colorare a lui G (de ce?) - contradicție.
- Astfel, $\chi(M(G)) = \chi(G) + 1$.

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

Exerciții rezolvate (partial)

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

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

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

Exercițiul 9. Soluție.

- Fie $G = (V, E)$ următorul graf:

$$V = \mathcal{S}, \forall i, j \in \mathcal{S}, ij \in E \text{ if } i \in c(j) \text{ or } j \in c(i).$$

- Numărul de muchii din G este cel mult $k|\mathcal{S}|$ (de ce?).
- Astfel, $2k|\mathcal{S}| \geq 2|E| = \sum_{i \in \mathcal{S}} d_G(i) \geq |\mathcal{S}|\delta(G)$ și trebuie să existe un nod $i \in \mathcal{S}$ așa încât $d_G(i) \leq 2k$ (de ce?).
- Orice subgraf al lui G are această (de ce?). Următorul algoritm construiește o $(2k + 1)$ -colorare a lui G :

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
- Graph Algorithms * C. Croitoru - Graph Algo428ns * C. Croitoru - Graph Algorithms *

```
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms  
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph  
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -  
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru  
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.  
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *  
while( $|G| > 0$ ) {  
    let  $i \in G$  cu  $d_G(i) \leq 2k$  and  $1 \leq h \leq 2k + 1$  a color not used by its  
already coloured neighbours;  
     $c(i) \leftarrow h$ ;  
}
```

- Clasele de colorare ale lui c sunt juriile cu proprietatea cerută.

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

12 CURS 12: Grafuri planare.

$$f = m - n + 2, m = f + n - 2.$$

Obs 1: $m \leq 3n - 6, f \leq 2n - 4$ (graf planar conex)

Obs 2: $m \leq 2n - 4, f \leq n - 2$ (graf planar conex si bipartit)

Graf plan : $f = m - n + 2$

Graf planar : $m \leq 3n - 6$

Graf planar bipartit : $m \leq 2n - 4$

Numarul de fete creste cand numarul de muchii creste sau cand numarul de noduri scade.

Algoritmica Grafurilor - Cursul 12

Cuprins

- 1 **Grafuri planare**

 - **Proprietăți elementare**
 - **Desenarea grafurilor planare**
 - **Separatori mici**

2 **Exerciții pentru seminarul din următoarea săptămână**

Fie $G = (V, E)$ un graf și \mathcal{S} o suprafață (e.g., plan, sferă) din \mathbb{R}^3 . O reprezentare a lui G pe \mathcal{S} este un graf $G' = (V', E')$ astfel încât:

- a) $G \cong G'$;
- b) V' este o mulțime puncte distincte ale lui \mathcal{S} ;
- c) Orice muchie $e' \in E'$ este o curbă simplă (arc Jordan) conținută în \mathcal{S} unindu-i extremitățile;
- d) Orice punct din \mathcal{S} este fie un nod al lui G' fie este conținut în cel mult o muchie a lui G' .

Dacă \mathcal{S} este un plan, atunci G este un **graf planar** și G' este o **reprezentare planară** a lui G .

Dacă \mathcal{S} este un plan și G' este un graf satisfăcând constrângerile b), c) și d) de mai sus, atunci G' se numește **graf plan**.

Grafuri planare - Proprietăți elementare - Proiecția stereografică

Lemă

Un graf este planar dacă și numai dacă are o reprezentare pe o sferă.

Demonstrație. Dacă G este planar, fie G' o reprezentare planară a lui G în planul π . Luăm un punct x în π și considerăm o sferă \mathcal{S} tangentă la π în x . Fie y punctul diametral opus lui x în \mathcal{S} . Considerăm $\varphi : \pi \rightarrow \mathcal{S} \setminus \{y\}$ dată prin $\varphi(M) =$ punctul diferit de y în care dreapta My intersectează sferă, $\forall M \in \pi$. φ este o bijecție și astfel $\varphi(G')$ este o reprezentare a lui G pe \mathcal{S} .

Reciproc, dacă G are o reprezentare pe o sferă \mathcal{S} : luăm un punct y în \mathcal{S} ^a, considerăm x , punctul diametral opus lui y pe \mathcal{S} , construim un plan tangent π la \mathcal{S} în x , și definim $\psi : \mathcal{S} \rightarrow \pi$ by $\psi(M) =$ punctul în care dreapta ym intersectează planul π , pentru orice $M \in \mathcal{S}$. Imaginea prin ψ a reprezentării lui G pe sferă, $\psi(G)$, este reprezentarea planară dorită a lui G . \square

^a y se alege a. i. să nu se afle pe vreo muchie sau în vreun nod al lui G .

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 un graf plan. Dacă stergem punctele lui G din plan (nodurile și muchiile sale), acesta este descompus într-o reuniune finită de regiuni conexe maximale din plan (oricare două puncte pot fi unite printr-o curbă simplă conținută în acea regiune), care sunt numite fețele lui G . Exact una dintre aceste fețe este nemărginită și este numită față **exterioră**.

Fiecare față este caracterizată de mulțimea muchiilor care-i formează **frontiera**. Fiecare circuit al lui G împarte planul în exact două regiuni conexe, astfel fiecare muchie a unui circuit aparține la exact două frontiere (la exact două fețe).

Un graf planar poate avea diferite reprezentări planare.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Grafuri planare - Proprietăți elementare - Fețe

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

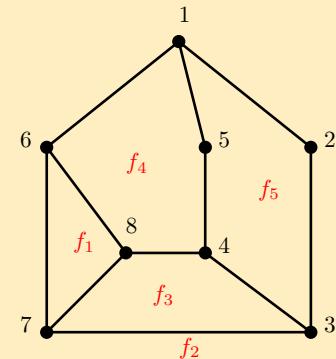
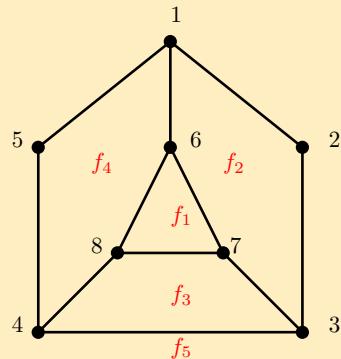
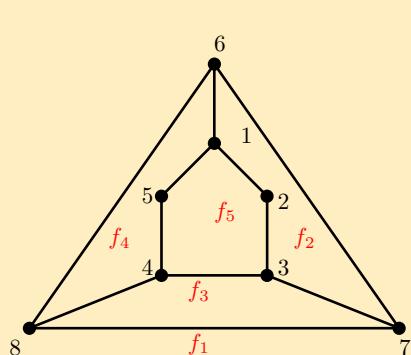
Lemă

Orice reprezentare planară a unui graf planar poate fi transformată într-o reprezentare planară diferită în care o față fixată a primei reprezentări să devină față exterioră a celei de-a doua.

Demonstrație. Fie G' o reprezentare planară a lui G și F' o față a lui G' . Fie G^0 o reprezentare a lui G' pe o sferă și F^0 față a lui G^0 corespunzând lui F . Alegem un punct y în interiorul lui F^0 , x punctul său diametral opus pe sferă, și π planul tangent în x la sferă.

$G'' = \psi(G^0)$ este o reprezentare a lui G în planul π având ca față exterioră $\psi(F^0)$. \square

Graph Algorithms * C. Croitoru - Graph Algo433ns * C. Croitoru - Graph Algorithms *

**Teoremă**

(Formula lui Euler) Fie $G = (V, E)$ un graf conex plan cu n noduri, m muchii și f fețe. Atunci,

$$f = m - n + 2.$$

Demonstrație. Inducție după f .

Demonstrație (continuare). Dacă $f = 1$, atunci G nu are circuite și, deoarece este conex, este un arbore. Urmează că $m = n - 1$ și teorema este adevărată.

În pasul inductiv să presupunem că teorema are loc pentru orice graf conex plan cu mai puțin de $f (\geq 2)$ fețe. Fie e o muchie de pe un circuit al lui G (există un astfel de circuit, deoarece $f \geq 2$). Atunci e aparține frontierei a exact două fețe a lui G . Urmează că $G_1 = G - e$ este un graf conex plan cu n noduri, $m - 1$ muchii și $f - 1$ fețe. Teorema are loc pentru G_1 , deci $f - 1 = m - 1 - n + 2$, i. e., $f = m - n + 2$. \square

Remarcă

Din punct de vedere algoritmic, teorema de mai sus implică (vezi următoarele două corolarii) că orice planar graf este rar: dacă m este numărul de muchii și n este numărul de noduri, atunci $m = \mathcal{O}(n)$.

Corolarul 1

Fie $G = (V, E)$ un graf conex planar cu $n \geq 3$ noduri și m muchii. Atunci,

$$m \leq 3n - 6.$$

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

Demonstrație. Fie G' o reprezentare planară a lui G . Dacă G' are doar o față, atunci G este un arbore, $m = n - 1$, și pentru $n \geq 3$ inegalitatea are loc.

Dacă G' are cel puțin două fețe, atunci fiecare față F a lui G' are în frontieră să muchiile unui circuit C_F , și fiecare astfel de muchie aparține la exact două fețe. Orice circuit al lui G' are cel puțin trei muchii, astfel

$$2m \geq \sum_{F \text{ față a lui } G'} \text{length}(C_F) \geq \sum_{F \text{ față a lui } G'} 3 = 3f = 3(m-n+2),$$

Adică inegalitatea dorită. \square

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

Remarcă

Graful K_5 nu este planar (numărul său de noduri este $n = 5$, numărul său de muchii este $m = 10$ și $10 > 3 \cdot 5 - 6$).

Corolarul 2

Fie $G = (V, E)$ un graf bipartit, planar și conex cu $n \geq 3$ noduri și $m \geq 3$ muchii. Atunci,

$$m \leq 2n - 4.$$

Demonstrație. Aceeași demonstrație ca pentru Corolarul 1, dar folosind faptul că orice circuit al lui G' are cel puțin patru muchii. \square

Remarcă

Graful $K_{3,3}$ nu este planar (numărul său de noduri este $n = 6$, numărul său de muchii este $m = 9$ și $9 > 2 \cdot 6 - 4$).

Corolarul 3

Dacă $G = (V, E)$ este un graf conex planar, atunci există $v_0 \in V$ astfel încât

$$d_G(v_0) \leq 5.$$

Demonstrație. Putem să presupunem că G are cel puțin două muchii (altfel e banal). Fie G' o reprezentare planară a lui G cu n noduri și m muchii. Dacă notăm cu n_i numărul de noduri de grad i ($1 \leq i \leq n-1$) atunci

$$\sum_{i=1}^{n-1} i \cdot n_i = 2m \leq 2(3n-6) = 6 \left(\sum_i n_i \right) - 12 \Rightarrow \sum_i (i-6)n_i + 12 \leq 0.$$

Dacă am avea $i \geq 6$, pentru orice i , toți termenii din această sumă sunt ≥ 0 , deci există $i_0 \leq 5$ astfel încât $n_{i_0} > 0$. \square

Grafuri planare - Proprietăți elementare - Teorema lui Kuratowski

Fie $G = (V, E)$ un graf și $v \in V$ astfel încât $d_G(v) = 2$ și $vw_1, vw_2 \in E$, $w_1 \neq w_2$.

Fie $h(G) = (V \setminus \{v\}, E \setminus \{vw_1, vw_2\} \cup \{w_1 w_2\})$.

Lemă

G este planar dacă și numai dacă $h(G)$ este planar.

Demonstrație. “ \Leftarrow ” Presupunem că $h(G)$ e planar.

Dacă $w_1 w_2 \notin E$, atunci pe curba simplă care unește punctele corespunzând lui w_1 și w_2 într-o reprezentare planară a lui $h(G)$ inserăm un punct nou corespunzând lui v ; dacă $w_1 w_2 \in E$ considerăm un punct nou corespunzând lui v “destul de aproape” de curba reprezentând $w_1 w_2$ pe una dintre fețele reprezentării planare a lui $h(G)$ și “unim” acest punct nou cu punctele corespunzând lui w_1 și w_2 prin curbe simple care nu le intersectează pe cele deja existente.⁴³⁶

Demonstrație (continuare) “ \Rightarrow ” Reciproc, presupunem că G este planar.

În reprezentarea sa planară, stergem punctul corespunzând lui v și cele două curbe corespunzând muchiilor vw_1 și vw_2 sunt înlocuite cu reuniunea lor; dacă $w_1w_2 \in E$, atunci curba simplă care-i corespunde este sătarsă. \square

Notăm cu $h^*(G)$ graful obținut din G prin aplicarea repetată a transformării h până se obține un graf fără noduri de grad 2.

Urmează că G este planar dacă și numai dacă $h^*(G)$ este planar.

Două grafuri G_1 și G_2 sunt homeomorfe dacă $h^*(G_1) \cong h^*(G_2)$.

Teoremă

(Kuratowski, 1930) Un graf este planar dacă și numai dacă nu conține subgrafuri homeomorfe cu K_5 sau cu $K_{3,3}$.

Desenarea grafurilor planare

Teoremă

(Fary, 1948, independent Wagner & Stein) Orice graf planar are o reprezentare planară cu toate muchiile segmente de dreaptă (reprezentare Fary).

Problemă: Să se determine o reprezentare Fary cu punctele reprezentând nodurile de coordonate întregi și aria suprafeței ocupată de reprezentare polinomială în n , numărul de noduri.

Teoremă

(Fraysseix, Pach, Pollack, 1988) Orice graf planar G cu n noduri are o reprezentare planară cu noduri în puncte cu coordonate întregi din $[0, 2n - 4] \times [0, n - 2]$ și cu toate muchiile segmente de dreaptă.

*C. Crițoiu - Graph Algorithms * C. Crițoiu - Graph Algorithms * C. Crițoiu - Graph Algorithms*

Demonstrație algoritmică. Vom schița o desenare în $\mathcal{O}(n \log n)$.

Fără a restrânge generalitatea, vom presupune că G este maximal planar: $\forall e \in E(G)$, $G + e$ nu este planar (adăugăm muchii la G pentru a-l face maximal planar și când aceste muchii (segmente) vor fi desenate le facem invizibile). Observăm că orice față a unui graf maximal planar este un triunghi și are $3n - 6$ muchii, unde n este numărul său de noduri.

Lema 1

Fie G un graf planar și G' o reprezentare planară a lui G . Dacă C' este un circuit al lui G' care trece prin muchia $uv \in E(G')$, atunci există $w \in V(C')$ astfel încât $w \neq u, v$ și nu există nicio coardă interioară a lui C' cu o extremitate în w .

Demonstrație. Fie v_1, v_2, \dots, v_n nodurile lui C' întâlnite într-o parcursare de la u la v ($v = v_1$, $u = v_n$).

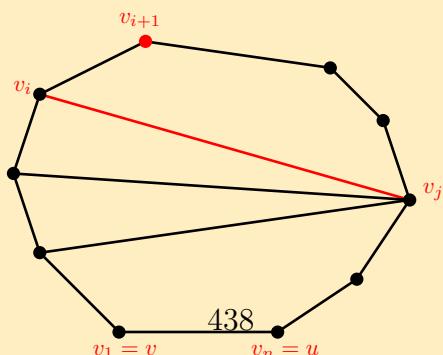
Desenarea grafurilor planare

*C. Crițoiu - Graph Algorithms * C. Crițoiu - Graph Algorithms * C. Crițoiu - Graph Algorithms*

Demonstrație (continuare). Dacă C' nu are corzi interioare, atunci lemma este adevărată. Altfel, alegem o pereche (i, j) astfel încât $v_i v_j$ este o coardă interioară a lui C' și

$j - i = \min \{k - l : k > l + 1, v_k v_l \in E(G'), v_k v_l$ coardă interioară a lui $C'\}$

Atunci, $w = v_{i+1}$ nu este incident cu o coardă interioară: $v_{i+1} v_p$ cu $i + 1 < p < j$ nu poate fi o coardă interioară - din modul de alegere a perechii (i, j) , și $v_{i+1} v_l$ cu $l < i$ sau $l > j$ nu este o coardă interioară deoarece ar trebui să intersecteze $v_i v_j$. \square



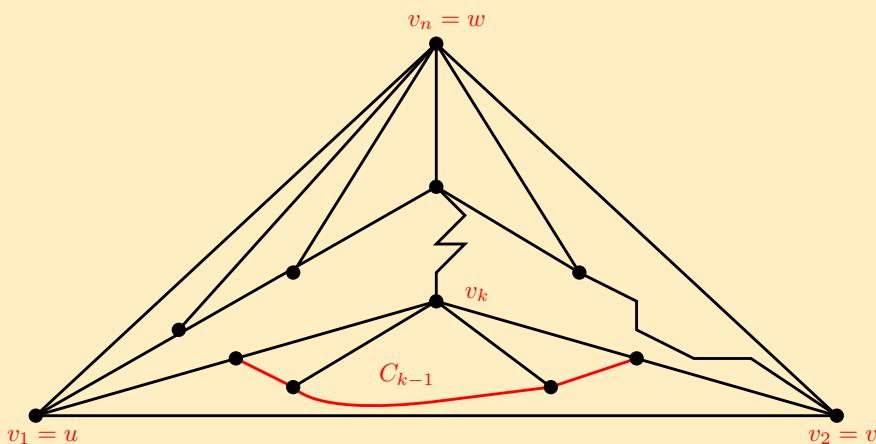
Lema 2

Fie G un graf maximal planar cu $n \geq 4$ noduri și G' o reprezentare planară a lui G având fața exterioară triunghiul u, v, w . Atunci, există o etichetare v_1, v_2, \dots, v_n a nodurilor lui G' astfel încât $v_1 = u$, $v_2 = v$, $v_n = w$ și, pentru fiecare $k \in \{4, \dots, n\}$, avem:

- (i) Subgraful induz $G'_{k-1} = [\{v_1, \dots, v_{k-1}\}]_G$ este 2-conex și fața sa exterioară este determinată de circuitul C'_{k-1} conținând uv .
- (ii) În subgraful induz G'_k nodul v_k este în fața exterioară a lui G'_{k-1} și $N_{G'_k}(v_k) \cap \{v_1, \dots, v_{k-1}\}$ este un drum de lungime ≥ 1 pe circuitul $C'_{k-1} - uv$.

Demonstrație. Fie $v_1 = u$, $v_2 = v$, $v_n = w$, $G'_n = G$, $G'_{n-1} = G - v_n$.

Desenarea grafurilor planare



Demonstrație (continuare). Observăm că $N_{G'_n}(w)$ este un circuit conținând uv (după o sortare a lui $N_{G'_n}(w)$ pe coordonata x și folosind planaritatea maximală). Urmează că i) și ii) au loc pentru $k = n$.

Dacă v_k a fost deja ales ($k \leq n$) atunci în $G'_{k-1} = G' - \{v_n, \dots, v_k\}$, vecinii lui v_k determină un circuit C'_{k-1} conținând uv și formând frontieră feței exterioare a lui G'_{k-1} . 439

Demonstrație (continuare). Din Lema 1, există v_{k-1} pe C'_{k-1} astfel încât v_{k-1} nu este extremitatea vreunei corzi interioare a lui C'_{k-1} . Prin construcție, v_{k-1} nu este incident cu vreo coardă externă a lui C'_{k-1} (din planaritatea maximală). Urmează că G'_{k-2} conține un circuit C'_{k-2} cu proprietățile (i) și (ii). \square

Demonstrația Teoremei (Fraysseix, Pach, Pollack). Fie G un graf maximal planar cu n noduri, G' o reprezentare planară cu nodurile etichetate v_1, \dots, v_n ca în Lema 2, și u, v, w față sa exterioară. Vom construi o reprezentare Fary a lui G având nodurile puncte de coordonate întregi.

Presupunem că în pasul k (≥ 3) al construcției avem o astfel de reprezentare a lui G_k și sunt îndeplinite următoarele trei condiții:

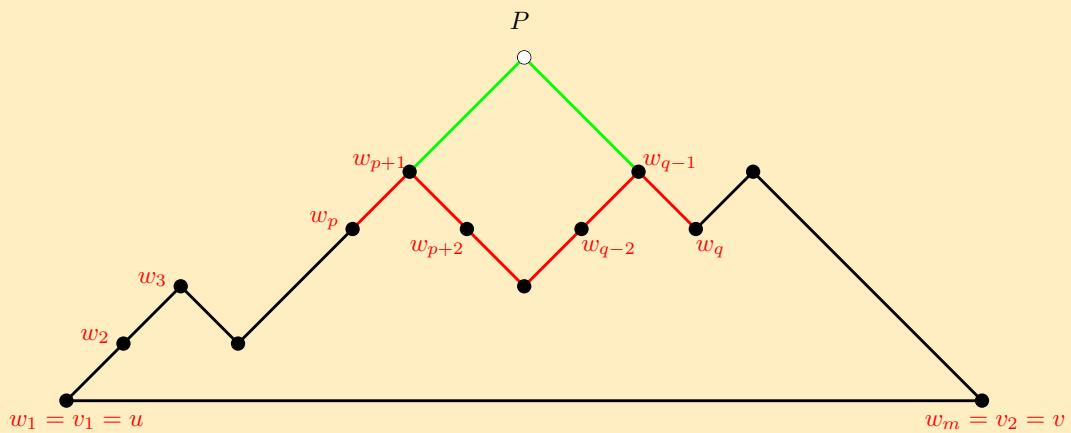
Desenarea grafurilor planare

Demonstrație (continuare).

- (1) v_1 are coordonatele $(0, 0)$; v_2 are coordonatele $(i, 0)$, $i \leq 2k - 4$.
 - (2) Dacă w_1, w_2, \dots, w_m sunt nodurile circuitului care corespunde feței exterioare a lui G_k , într-o parcurgere de la v_1 la v_2 ($w_1 = v_1$, $w_m = v_2$), atunci
- $x_{w_1} < x_{w_2} < \dots < x_{w_m}$.
- (3) Muchiile $w_1w_2, w_2w_3, \dots, w_{m-1}w_m$ sunt segmente de dreaptă paralele cu una dintre cele două bisectoare ale axelor de coordonate.

Condiția (3) implică faptul că $\forall i < j$, paralela prin w_i la prima bisectoare intersectează paralela prin w_j la cea de-a doua bisectoare într-un un punct de coordonate întregi (w_i și w_j au coordonate întregi).

Construcția lui G'_{k+1} . Fie w_p, w_{p+1}, \dots, w_q vecinii din G'_k ai lui v_{k+1} în G'_{k+1} ($1 \leq p < q \leq m$).



Demonstrație (continuare) Paralela prin w_p la prima bisectoare intersectează paralela prin w_q la cea de-a două bisectoare în punctul P . Dacă din P putem trasa segmentele Pw_i , $p \leq i \leq q$ astfel încât toare sunt distincte, atunci putem lua $v_{k+1} = P$ pentru a obține o reprezentarea Fary a lui G_{k+1} cu toate noduri de coordonate întregi, satisfăcând condițiile (1) - (3).

Desenarea grafurilor planare

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 segmentul $w_p w_{p+1}$ este paralel cu prima bisectoare, atunci translăm către dreapta cu 1 toate nodurile lui G_k care au $x \geq x_{w_{p+1}}$. Facem o altă translație la dreapta cu 1 a tuturor nodurilor lui G_k având coordonata $x \geq x_{w_q}$.

Acum, toate segmentele $P'w_i$, cu $p \leq i \leq q$, sunt distincte, segmentele $w_i w_{i+1}$ cu $i = \overline{q, m-1}$ au pantele ± 1 și de asemenea $w_p P'$ și $P'w_q$ au pantele ± 1 (unde P' este intersecția paralelei la prima bisectoare prin w_p cu paralela la cea de-a două bisectoare prin w_q).

Luăm $v_{k+1} = P'$ și pasul k al construcției este terminat. \square

Algoritmul poate fi implementat în $\mathcal{O}(n \log n)$.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
 Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
 - Graph Algorithms * C. Croitoru - Graph Algo441ns * C. Croitoru - Graph Algorithms *

Teoremă

(Tarjan & Lipton, 1979) Fie G un graf planar cu n noduri. Există o partitie (A, B, S) a lui $V(G)$ astfel încât:

- S separă A de B în G : $G - S$ nu conține muchii de la A la B ,
- $|A| \leq (2/3)n$, $|B| \leq (2/3)n$,
- $|S| \leq 4\sqrt{n}$.

Această partitie poate fi găsită în $\mathcal{O}(n)$.

Ideeă demonstrației. Fie G un graf conex plan. Executăm o parcurgere bfs dintr-un nod oarecare s , etichetând fiecare nod v cu nivelul corespunzător din arborele bfs obținut. Fie $L(t)$, mulțimea tuturor nodurilor de pe nivelul t , pentru $0 \leq t \leq l + 1$. Ultimul nivel $L(l + 1)$ este - din rațiuni tehnice - vid (ultimul nivel este în fapt l).

Grafuri planare - Separatori mici

Demonstrație (continuare). Fiecare nivel intern este un separator în G (avem muchii doar între nivele consecutive). Fie t_1 nivelul de mijloc, adică nivelul care conține cel de-al $\lfloor n/2 \rfloor$ -lea nod întâlnit în timpul parcurgerii. Mulțimea $L(t_1)$ satisface:

$$\left| \bigcup_{t < t_1} L(T) \right| < \frac{n}{2} \text{ și } \left| \bigcup_{t > t_1} L(T) \right| < \frac{n}{2}.$$

Dacă $|L(t_1)| \leq 4\sqrt{n}$, teorema este demonstrată.

Lemă

Există nivelele $t_0 \leq t_1$ și $t_2 \geq t_1$ astfel încât $|L(t_0)| \leq \sqrt{n}$, $|L(t_2)| \leq \sqrt{n}$ și $t_2 - t_0 \leq \sqrt{n}$.

Demonstrație. Considerăm t_0 cel mai mare întreg care satisface $t \leq t_1$ și $|L(t)| \leq \sqrt{n}$ (există un astfel de nivel deoarece $|L(0)| = 1$). Există t_2 un cel mai mic întreg care satisface $t > t_1$ și $|L(t_2)| \leq \sqrt{n}$ (se observă că $|L(l+1)| = 0$).

Orice nivel dintre t_0 și t_2 are mai mult de \sqrt{n} noduri, deci numărul acestor nivele nu poate depăși \sqrt{n} (altfel, numărul de noduri ar fi $> n$).

□

Demonstrație (continuare la Teorema separatorilor). Fie

$$C = \bigcup_{t < t_0} L(t), D = \bigcup_{t_0 < t < t_2} L(t), E = \bigcup_{t > t_2} L(t).$$

- $|D| \leq (2/3)n$. Teorema are loc cu $S = L(t_0) \cup L(t_2)$, A mulțimea de cardinal maxim dintre C , D și E , iar B reuniunea celor două mulțimi rămase (C și E au cel mult $n/2$ elemente).

Demonstrație (continuare la Teorema separatorilor).

- $n_1 = |D| > (2/3)n$. Dacă putem găsi un separator de tipul $1/3 \leftrightarrow 2/3$ pentru D cu cel mult $2\sqrt{n}$ noduri, atunci îl adăugăm la $L(t_0) \cup L(t_2)$ pentru a obține un separator de cardinal cel mult $4\sqrt{n}$, pentru A luăm reuniunea mulțimii de cardinal maxim dintre C și E cu partea mai mică rămasă din D , și pentru B luăm reuniunea celor două mulțimi rămase.

Separatorul pentru (graful induș de) D poate fi construit astfel: ștergem toate nodurile lui G care nu sunt din D mai puțin s care este unit cu toate nodurile de pe nivelul $t_0 + 1$. Graful obținut se notează cu D' și este planar și conex. Are un arbore parțial de diametru cel mult $2\sqrt{n}$ (orice nod este accesibil din s pe un drum de lungime cel mult \sqrt{n} , după cum am demonstrat în Lema de mai sus). Acest arbore este parcurs **dfs** pentru a obține separatorul dorit. Detaliile (foarte interesante) sunt omise. □

Considerăm problema de a decide dacă un graf planar dat are o 3-colorare (a nodurilor), care este o problemă **NP**-completă.

În cazul unui graf G cu număr mic de noduri (pentru o constantă c , putem verifica toate cele $\mathcal{O}(3^c) = \mathcal{O}(1)$ funcții de la $V(G)$ la $\{1, 2, 3\}$) putem decide dacă există o 3-colorare.

Pentru grafuri planare cu $n > c$ noduri construim în timp liniar, $\mathcal{O}(n)$, partitia (A, B, C) a nodurilor sale, cu $|A|, |B| \leq (2n/3)$ și $|C| \leq \sqrt{n}$.

Se testează fiecare $3^{|C|} = 2^{\mathcal{O}(\sqrt{n})}$ funcție posibilă de la C la $\{1, 2, 3\}$ dacă este o 3-colorare a subgrafului induș de C și dacă această colorare poate fi extinsă la o 3-colorare a subgrafului induș de $A \cup C$ în G și de asemenei la o 3-colorare a subgrafului induș de $B \cup C$ în G (recursiv).

O aplicație a Teoremei Separatorilor

Timpul de execuție, $T(n)$, al acestui algoritm, satisfacă recursia

$$T(n) = \begin{cases} \mathcal{O}(1), & \text{dacă } n \leq c, \\ \mathcal{O}(n) + 2^{\mathcal{O}(\sqrt{n})} (\mathcal{O}(\sqrt{n}) + 2 T(2n/3)), & \text{dacă } n > c. \end{cases}$$

Urmează că $T(n) = 2^{\mathcal{O}(\sqrt{n})}$ (este posibil ca acele constante din spatele notației $\mathcal{O}(\cdot)$ să fie foarte mari).

Exercițiu 1. Fie $G = (V, E)$ un graf plan cu n noduri și m muchii.

- (a) Dacă lungimea circuitului din fontiera oricărei fețe este cel puțin $k \geq 3$ pentru un întreg k , atunci $m \leq \frac{k(n-2)}{k-2}$.
- (b) Arătați că graful lui Petersen nu este planar.

Exercițiu 2. Fie $G = (V, E)$ un graf plan cu n noduri, m muchii și p componente conexe. Determinați o formulă pentru numărul fețelor lui G în termenii lui n , m și p .

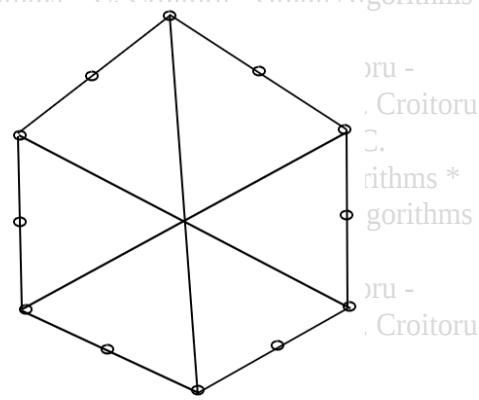
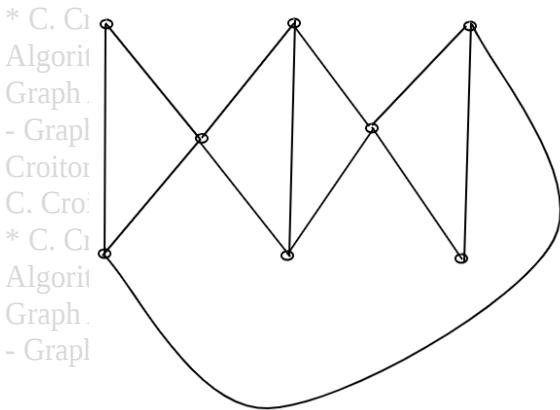
Exercițiu 3. Care dintre următoarele grafuri are proprietatea că prin stergerea oricărui nod se obține un graf planar?

$K_5, K_6, K_{4,3}, K_{3,3}$, graful lui Petersen.

Exerciții pentru seminarul din următoarea săptămână

Exercițiu 4. **Crossing number**, $cr(\cdot)$, al unui graf este numărul minim intersecții ale muchiilor (în alte puncte decât în noduri) atunci când graful este desanat în plan (presupunem că trei muchii nu se pot intersecta într-un același punct care nu este nod al grafului). Determinați $cr(K_{3,3})$, $cr(K_5)$, $cr(K_6)$ și cr (graful lui Petersen).

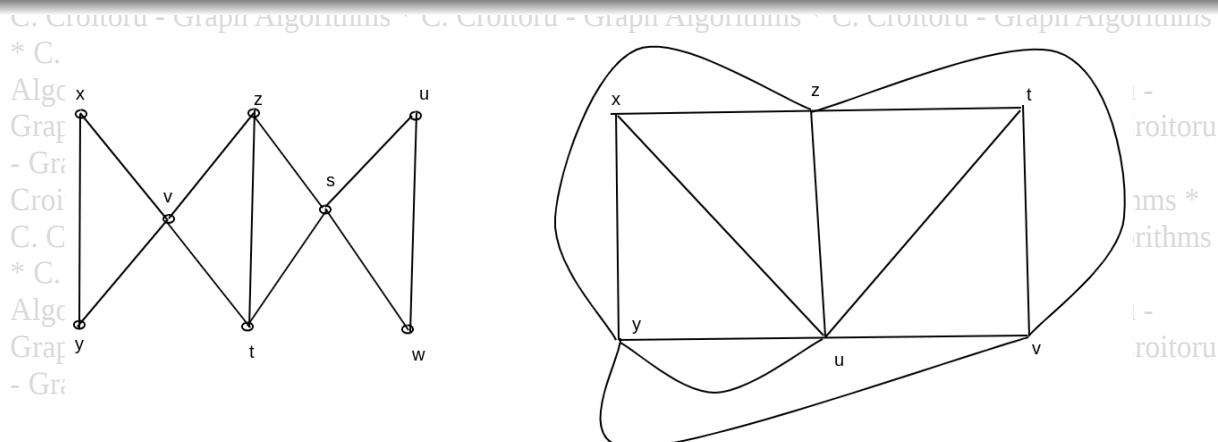
Exercițiu 5. Folosiți teorema lui Kuratowski pentru a afla care dintre următoarele grafuri este planar:



Exercițiu 6. Fie G un (multi-) graf plan, definim un multi-graph, G^* :

- fiecărei fețe f a lui G îi corespunde un nod f^* al lui G^* ;
- fiecărei muchii e a lui G îi corespund o muchie e^* a lui G^* .
- două noduri f_1^* și f_2^* sunt unite printr-o muchie e^* dacă fețele f_1 și f_2 au muchia e în frontieră comună.

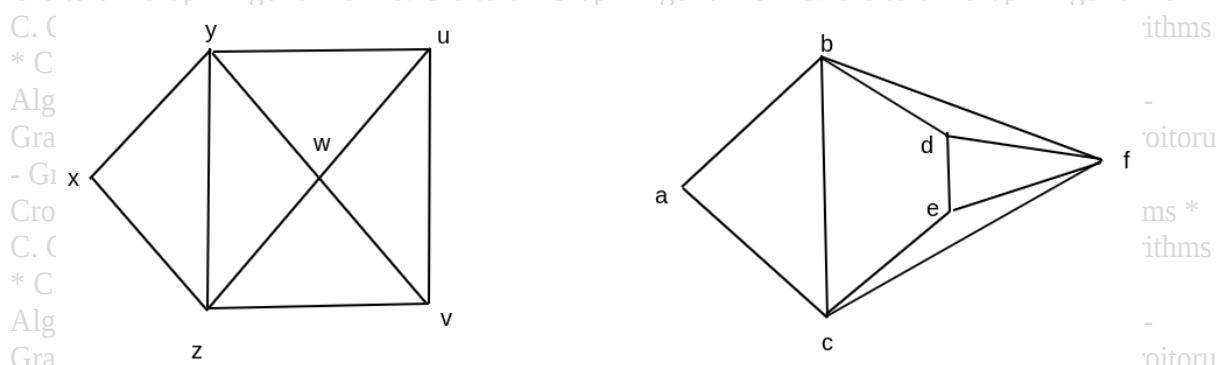
Desenați dualele următoarelor grafuri planare:



Exerciții pentru seminarul din următoarea săptămână

Exercițiu 7.

- Arătați că dualul unui graf plan este planar.
- Dacă G este un graf conex plan, atunci $G^{**} \cong G$.

Exercițiu 8. Arătați că următoarele două grafuri plane sunt izomorfe, dar dualele lor nu sunt.

Exercițiu 9. Fie G un graf conex plan și G^* dualul lui.

- (a) Fie T un arbore parțial al lui G , atunci muchiile lui G^* care nu corespund muchiilor din $E(T)$ sunt muchiile unui arbore parțial în G^* .
- (b) Numărul de arbori parțiali ai lui G este egal cu numărul de arbori parțiali din dualul său, G^* .

Exercițiu 10. Fie G un graf plan cu toate fețele triunghiulare; colorăm aleatoriu cu trei culori nodurile sale. Arătați că numărul de fețe care primesc toate cele trei culori este par.

Exercițiu 11*. Fie G un graf plan cu toate gradele pare. Arătați că îi putem colora fețele cu două culori astfel ca orice două fețe care au cel puțin o muchie în comun în frontiere au culori diferite.

Exercițiu 12*. Fie G un graf plan cu toate fețele triunghiulare ($|G| \geq 4$). Arătați că dualul său, G^* este 2-muchie conex și 3-regulat (în consecință G^* are cuplaj perfect).

13 CURS 13: Tree decompositions.

Algoritmica Grafurilor - Cursul 13

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

15 ianuarie 2021

Cuprins

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

1 Tree decomposition

- **Tree width**
- **Small tree decompositions**
- **Proprietăți ale tree descompunerilor**
- **Tree descompuneri cu rădăcină**
- **Aplicații**

O tree decomposition a unui graf $G = (V, E)$ este o pereche $\mathcal{T} = (T, \{V_t : t \in T\})$, unde T este un arbore iar $\{V_t : t \in V(T)\}$ este o familie de submulțimi de noduri ale lui G , $V_t \subseteq V$, $\forall t \in T$ astfel ca:

- (Acoperirea nodurilor) $V = \bigcup_{t \in V(T)} V_t$;
 - (Acoperirea muchiilor) Pentru orice muchie $e \in E$, ambele capete ale lui e sunt conținute într-o mulțime V_t , pentru un anumit $t \in V(T)$.
 - (Coerență) Fie t_1, t_2, t_3 trei noduri din T astfel ca t_2 se află pe drumul dintre t_1 și t_3 în T . Atunci, dacă $v \in V$ se află în V_{t_1} și V_{t_3} , v se află și în V_{t_2} .

Tree-width - Definiție

Remarci

Proprietatea de coerentă poate fi reformulată astfel:

- (Coerența') Fie $t_1, t_2, t_3 \in V(T)$ așa încât t_2 aparține drumului dintre t_1 și t_2 în T . Atunci $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$.
 - (Coerența") Pentru orice $x \in V$, subgraful lui T indus de $\{t \in V(T) : x \in V_t\}$ este (un subarbore al lui T) conex.

Multimedie

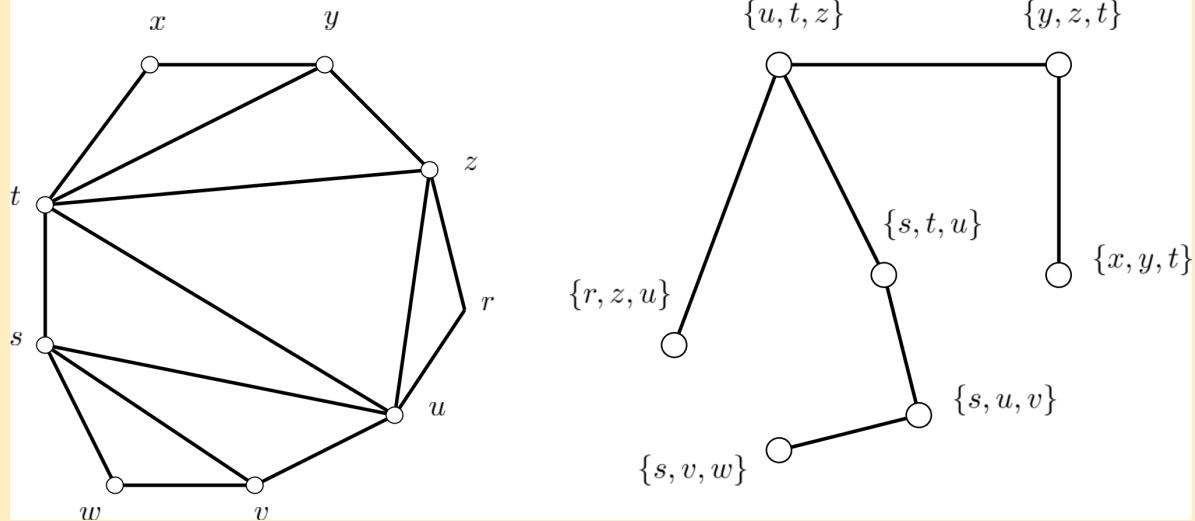
Fie $\mathcal{T} = (T, \{V_t : t \in T\})$ o tree-descompunere a lui G , lățimea (width) lui \mathcal{T} este

$$width(\mathcal{T}) = \max_{t \in V(\mathcal{T})} (|V_t| - 1).$$

Definiție

Tree-width a unui graf G , este cea mai mică lățime a unei tree-descompuneri a lui G :

$$tw(G) = \min \{ width(\mathcal{T}) : \mathcal{T} \text{ tree-descompunere a lui } G \}.$$



Tree-width

Remarcă

$tw(G) = 0$ dacă și numai dacă $E(G) = \emptyset$.

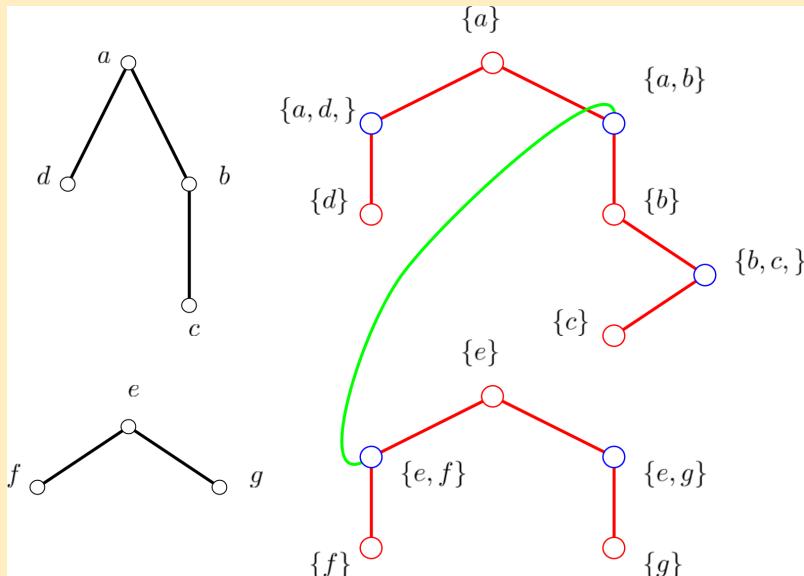
Propoziție

Dacă G este o pădure cu $E(G) \neq \emptyset$, atunci $tw(G) = 1$.

Demonstrație. Din observația de mai sus $tw(G) \geq 1$. Dacă G este un arbore, atunci

- fie T obținut din G prin redenumirea fiecărui nod $v \in V(G)$ prin t_v ,
- inserează pe fiecare muchie $t_u t_v$ ($uv \in E(G)$) un vârf nou t_{uv} ,
- setează $V_{t_u} = \{u\}$ pentru orice t_u asociat lui $u \in V(G)$ și $V_{t_{uv}} = \{u, v\}$ pentru orice $t_{uv} \in V(T)$ asociat lui $uv \in E(G)$.
- $(T, \{V_t : t \in V(T)\})$ este o tree-descompunere a lui G cu lățimea 1.

Demonstrație (continuare). O tree-descompunere a unei păduri cu p componente se poate obține adăugând arbitrar $p - 1$ muchii la tree-descompunerile componentelor (fără a crea circuite).



Small tree decompositions

Definiție

O tree-descompunere, $\mathcal{T} = (T, \{V_t : t \in V(T)\})$, este mică (**small**) dacă nu există noduri distincte $t_1, t_2 \in V(T)$ astfel ca $V_{t_1} \subseteq V_{t_2}$.

Propoziție

Dată o tree-descompunere a lui G , o tree-descompunere small a lui G de aceeași lățime poate fi construită în timp polinomial.

Demonstrație. Fie $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ o tree-descompunere a lui G cu $V_{t_1} \subseteq V_{t_2}$ pentru $t_1, t_2 \in V(T)$, $t_1 \neq t_2$. Putem presupune că $t_1 t_2 \in E(T)$ (altfel, putem determina două noduri adiacente cu această proprietate considerând drumul de la t_1 la t_2).

Contractând $t_1 t_2$ într-un nod nou t_{12} cu $V_{t_{12}} = V_{t_2}$, se obține o tree-descompunere a lui G mai mică (conține mai puține perechi de noduri (t'_1, t'_2) cu $V_{t'_1} \subseteq V_{t'_2}$).

Demonstrație (continuare). Repetăm această reducere până se obține o tree-descompunere small.



Propoziție

Dacă $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ este o tree-descompunere small a lui G , atunci $|T| \leq |G|$.

Demonstrație. Prin inducție după $n = |G|$. Dacă $n = 1$, atunci $|T| = 1$. În pasul inductiv, pentru $n \geq 2$, considerăm o frunză t_1 a lui T cu vecinul t_2 . $(T - t_1, \{V_t : t \in V(T - t_1)\})$ este o tree-descompunere small a lui $G' = G \setminus (V_{t_1} \setminus V_{t_2})$. Din ipoteza inductivă $|T - t_1| \leq |G'|$, astfel

$$|T| = |T - t_1| + 1 \leq |G'| + 1 \leq |G|.$$



Minori

Remarci

- dacă graful H se obține din G prin contractarea unei muchii uv în z , atunci $tw(H) \leq tw(G)$: încearcă să inserăm z în fiecare bag care conține pe u sau v și ștergem apoi pe u și v din orice bag pentru a obține o tree-descompunere a lui H .
- dacă H este un subgraf al lui G , atunci $tw(H) \leq tw(G)$.

Definiție

H este un **minor** al unui graf G dacă se poate obține din G prin ștergerea și contractarea succesivă a unor muchii din G .

Corolar

Dacă H este un minor al grafului G , atunci $tw(H) \leq tw(G)$.

Demonstrație. Folosind observațiile de mai sus.



Fie $TW_k = \{G : tw(G) \leq k\}$.

TW (Tree-Width - versiunea de decizie)

Instanță: G un graf și $k \in \mathbb{N}$.

Întrebare: $G \in TW_k$?

Teoremă

Problema Tree-Width (versiunea de decizie) este o problemă NP-completă.

Demonstrație. Omisă.

Tree-width este FPT (fixed-parameter tractable)

Lemă

Pentru orice $k \in \mathbb{N}^*$, TW_k este o familie închisă la minori.

Teoremă

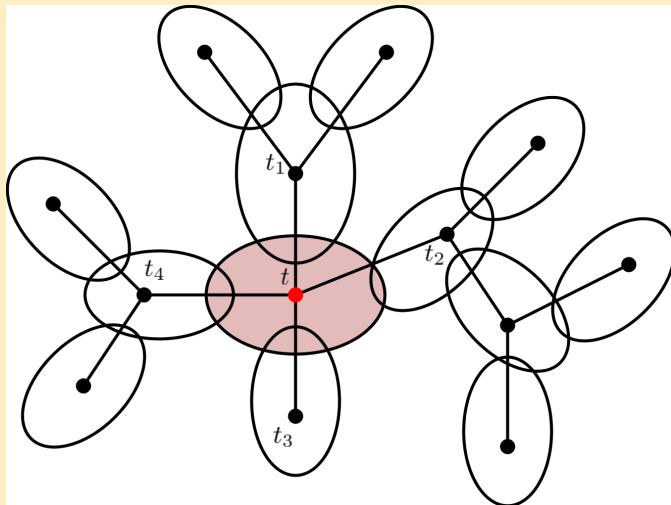
(Bodlaender) Pentru orice k fixat, problema determinării dacă G este în TW_k sau nu poate fi rezolvată în timpul $\mathcal{O}(f(k) \cdot n)$.

Proofs. Omisă. ($f(k)$ este o funcție exponențială în k .)

Notăție: Fie $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ o tree-descompunere of G . Dacă T' este un subgraf al lui T , $G_{T'}$ este subgraful lui G induș de multimea de noduri $\bigcup_{t \in V(T')} V_t$.

Teoremă

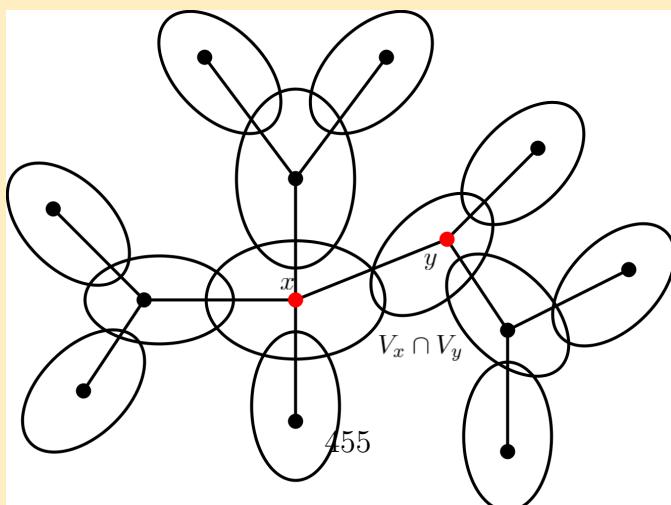
(Proprietatea separării nodurilor.) Presupunem că $T - t$ are componente conexe T_1, T_2, \dots, T_p . Atunci subgrafurile $G_{T_1} - V_t, G_{T_2} - V_t, \dots, G_{T_p} - V_t$ nu au noduri în comun și nici muchii între ele.



Proprietăți ale tree descompunerilor

Teoremă

(Proprietatea separării muchiilor.) Fie X și Y componente conexe ale lui T după ștergerea unei muchii $xy \in E(T)$. Atunci, ștergerea nodurilor din $V_x \cap V_y$ deconectează G în două subgrafuri $H_X = G_X - V_x \cap V_y$ și $H_Y = G_Y - V_x \cap V_y$. Cu alte cuvinte H_X și H_Y nu au noduri în comun și nici muchii între ele.



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

Alte proprietăți:

- Fie G un graf conex cu $tw(G) = k$, atunci sau $|G| = k + 1$ sau G are o mulțime separatoare de noduri de cardinal k .
 - dacă $tw(G) = 1$, atunci G este o pădure.
 - $tw(P_r \times P_s) = \min \{r, s\}$.
 - $tw(K_n) = n - 1$.

Tree descompuneri cu rădăcină

O tree-descompunere cu rădăcină a lui G este o tree-descompunere $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ a lui G , în care un anumit nod r al lui T este declarat rădăcină.

Notății: Fie t un nod dintr-o tree-descompunere cu rădăcină $\mathcal{T} = (T, \{V_t : t \in V(T)\})$.

- T_t este subarborele lui T cu rădăcina în t .
 - $G[t]$ este subgraful lui G indus de nodurile din $\bigcup_{x \in V(T_t)} V_x$ (i. e., $G[t] = G_{T_t}$).

- Reamintim: o p -colorare a nodurilor unui graf $G = (V, E)$ este o funcție $c : V \rightarrow \{1, 2, \dots, p\}$ astfel ca $c(u) \neq c(v)$ pentru orice $uv \in E$.
- Fie H' și H'' două subgrafuri ale lui G , cu p -colorările c' și c'' , respectiv. c'' este c' -compatibilă dacă pentru orice $v \in V(H') \cap V(H'')$ avem $c'(v) = c''(v)$.
- Fie $T = (T, \{V_t : t \in V(T)\})$ tree-descompunere cu rădăcină a lui G . Pentru orice $t \in T$ și orice p -colorare c a lui G_t , definim

$$\text{Prev}_t(c) = \begin{cases} 1, & \text{dacă } G[t] \text{ are o } p\text{-colorare } \bar{c}, c\text{-compatibilă} \\ 0, & \text{altfel.} \end{cases}$$

Propoziție

$\text{Prev}_u(c) = 1$ dacă și numai dacă pentru orice copil v al lui u , există o colorare \bar{c} a lui G_v , c -compatibilă cu $\text{Prev}_v(\bar{c}) = 1$

Aplicații - Colorarea nodurilor

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

Demonstrație. “ \implies ” dacă γ este colorare a lui $G[u]$, c -compatibilă, cum G_v este un subgraf al lui $G[u]$, atunci restricția lui γ la G_v ne oferă colorarea dorită, \bar{c} .

“ \impliedby ” Presupunem că u are exact doi copii v și w , și că avem două colorări \bar{c}' și \bar{c}'' , c -compatibile, respectiv (demonstrația este similară pentru mai mulți copii).

Deoarece $(T, \{V_t : t \in V(T)\})$ este o tree-descompunere, $V(G[v]) \cap V(G[w]) \subseteq V_u$, urmează că \bar{c}' este \bar{c}'' -compatibilă.

Combinând \bar{c}' și \bar{c}'' obținem $\bar{c} : V(G[u]) \rightarrow \{1, 2, \dots, p\}$. Cum $(T, \{V_t : t \in V(T)\})$ este o tree-descompunere, nu există muchii $xy \in E(G)$ cu $x \in V(G[v]) - V_u$ și $y \in V(G[w]) - V_u$, deci \bar{c} este o p -colorare a lui $G[u]$.



Teoremă

Dacă G , un graf de ordin n , are o tree-descompunere small ($T, \{V_t : t \in V(T)\}$) cu lățimea w , atunci putem decide dacă G este p -colorabil în timpul $\mathcal{O}(p^{w+1} \cdot n^{\mathcal{O}(1)})$.

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

Demonstrație. Transformăm $(T, \{V_t : t \in V(T)\})$ într-o a tree-descompunere cu rădăcină (r). Pentru orice $v \in V(T)$ și orice p -colorare c a lui G_v , determinăm $Prev_v(c)$: plecând din frunzele lui T , și apoi folosind proprietatea de mai sus pentru celelalte noduri, într-o ordine corespunzătoare.

$G = G[r]$ este p -colorabil dacă și numai dacă $Prev_v(c) = 1$ pentru o colorare c . Testarea dacă este G_v colorare și determinarea lui $Prev_v(c)$ pot fi făcute în timpul $\mathcal{O}(n^{\mathcal{O}(1)})$, astfel complexitatea timp totală este dată în principal de numărul de candidați pentru c , adică $p^{|V_v|}$.

Complexitate: $|V(T)| \cdot p^{w+1} \cdot n^{\mathcal{O}(1)} = \mathcal{O}(p^{w+1} \cdot n^{\mathcal{O}(1)})$.

Alte aplicații - Abordări similare (programare dinamică mai avansată)

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

Teoremă

Dacă G , un graf de ordin n , are o tree-descompunere small ($T, \{V_t : t \in V(T)\}$) cu lățimea w , cardinalul minim al unei acoperiri cu noduri a lui G poate fi determinată în timpul $\mathcal{O}(2^{w+1} \cdot n^{\mathcal{O}(1)})$.

Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

Dacă G , un graf de ordin n cu ponderi pe noduri, are o tree-descompunere small ($T, \{V_t : t \in V(T)\}$) cu lățimea w , o mulțime stabila de pondere maximă a lui G poate fi determinată în timpul $\mathcal{O}(4^{w+1} \cdot w \cdot n)$.

Graph Algorithms * C. Croitoru - Graph Algorithms

14 Diverse cursuri explicate poate mai bine

1. Grafuri neorientate

Cerințele programei pentru BAC:

- terminologie (nod/vârf, muchie, adiacență, incidentă, grad, lanț, ciclu, lungime, subgraf, graf parțial)
- proprietăți (regulat, complet, aciclic, conex, componentă conexă, hamiltonian, eulerian)
- metode de reprezentare în memorie (matrice de adiacență, liste de adiacență)

Definiție: Un **graf** este o pereche ordonată de mulțimi, notată $G=(X,U)$, unde

$X=\{x|x\in X\}$ este **mulțimea nodurilor** (vârfurile) iar $U=\{(x,y)| x,y\in X\}$, **mulțimea muchiilor**.

nod/vârf = element al mulțimii X ; poate fi reprezentat în plan printr-un punct (cerc etc.), eventual numerotat.

muchie = pereche neordonată de noduri; poate fi reprezentată în plan printr-un segment de dreaptă/arc

adiacență = proprietatea a două noduri de a fi unite prin muchie; dacă $[x,y]\in U$, spunem că nodurile x și y sunt adiacente

incidentă = proprietatea unei muchii de a uni două noduri; dacă $[x,y]\in U$, spunem că muchia este incidentă cu nodurile x și y

gradul nodului x = numărul de muchii incidente cu nodul x , notat cu $d(x)$

- *nod izolat* = nod cu gradul 0; $d(x)=0$
- *nod terminal* = nod cu gradul 1; $d(x)=1$

Propoziție: În orice graf neorientat cu n noduri și m muchii, are loc egalitatea

$$2*m = d(x_1) + d(x_2) + \dots + d(x_n)$$

(Suma gradelor varfurilor este dublul numarului de muchii)

Consecinta: În orice G există un număr PAR de varfuri de graf IMPAR

lanț = succesiune de noduri cu proprietatea că oricare două noduri consecutive din lanț sunt adiacente

- *lanț compus* = lanț în care muchiile se pot repeta
- *lanț simplu* = lanț în care fiecare muchie apare o singură dată dar nodurile se pot repeta
- *lanț elementar* = lanț în care nodurile sunt distincte

ciclu = lanț în care primul nod coincide cu ultimul

- *ciclu compus* = ciclu în care muchiile se pot repeta
- *ciclu simplu* = ciclu în care fiecare muchie apare o singură dată dar nodurile se pot repeta
- *ciclu elementar* = ciclu în care nodurile sunt distincte, cu excepția primului și ultimului nod

lungimea unui lanț/ciclu = numărul de muchii din care este format

graf parțial = graf care se obține din graful inițial prin eliminarea unor muchii, nu și a nodurilor

subgraf = graf care se obține din graful inițial prin eliminarea unor noduri și a tuturor muchiilor incidente cu acestea; nu pot fi eliminate alte muchii decât cele incidente cu nodurile eliminate

tipuri particulare de grafuri

graf regulat = graf în care toate nodurile au grade egale

graf complet = graf în care orice două noduri distincte sunt adiacente

- numărul de muchii într-un graf complet = $n * (n-1) / 2$
- Numărul grafurilor neorientate cu n vârfuri este $2^{n(n-1)/2}$
- Numărul grafurilor parțiale peste un graf cu M muchii 2^M

graf aciclic = graf în care nu există nici un ciclu

graf conex = oricare ar fi două noduri distincte, există lanț între ele

componentă conexă = un subgraf conex și maximal în raport cu această proprietate
(nu există lanț între un nod din subgraf și un nod care nu aparține subgrafului)

Obs: un nod izolat constituie o componentă conexă

ciclu hamiltonian = ciclu elementar care trece prin **toate** vâfurile grafului

graf hamiltonian = graf care conține cel puțin un ciclu hamiltonian

Condiție suficientă de existență a unui ciclu hamiltonian:

Un graf neorientat cu **n** vârfuri, în care gradul oricărui vârf este mai mare sau egal cu **n/2** este hamiltonian.

ciclu eulerian = ciclu care trece prin toate muchiile grafului

graf eulerian = graf care conține un ciclu eulerian

Condiție necesară și suficientă de existență a unui ciclu eulerian

Th. lui Dirac: Un graf fără vârfuri izolate, este **eulerian** dacă și numai dacă

- este **conex**
- **gradele** tuturor vâfurilor sale sunt **pare**

OBS: Un graf complet cu numar **impar** de varfuri este hamiltonian și eulerian

Un graf complet cu numar **par** de varfuri este hamiltonian și NU este eulerian (ar avea gradele toate impare) => elimin MINIM n/2 muchii și poate deveni eulerian

Graf hamiltonian și eulerian: un poligon

Graf hamiltonian și NU eulerian: un poligon cu o diagonala

Graf NU hamiltonian și eulerian: o fundita cu 5 noduri (unul e în mijloc)

Graf NU hamiltonian și NU eulerian: ciclu neelementar și grade impare

Metode de reprezentare a grafurilor neorientate în memorie

Matricea de adiacență

- matricea este simetrică față de diagonala principală
- gradul unui nod i (x) = numarul de valori 1 de pe linia/coloana x

2. Listele de adiacență

L1: 2, 4
 L2: 1, 3, 4
 L3: 2, 5
 L4: 1, 2
 L5: 3, 6
 L6: 5

0	1	0	1	0	0
1	0	1	1	0	0
0	1	0	0	1	0
1	1	0	0	0	0
0	0	1	0	0	1
0	0	0	0	1	0

$$L_i = \{j \in X / [i, j] \in U\}$$

3. Lista de muchii

$t \in M_{2xm}$, unde $m =$ numărul de muchii din graf

$t_{1,k}$ și $t_{2,k} =$ extremitățile muchiei k

2	4	4	3	5	6
1	1	2	2	3	5

2. Grafuri orientate

Cerințele programei pentru BAC:

- terminologie (nod/vârf, muchie, adiacență, incidență, grad intern si extern, drum, circuit, lungime, subgraf, graf parțial)
- proprietăți (tare conex, componentă tare conexă)
- metode de reprezentare în memorie (matrice de adiacență, liste de adiacență)

Definiție: Un **graf orientat** este o pereche ordonată de mulțimi, notată $G=(X,U)$, unde $X=\{x|x \in X\}$ este **mulțimea nodurilor** (vârfurilor) iar $U=\{(x,y)| x,y \in X\}$, **mulțimea arcelor**.

nod/vârf = element al mulțimii X ; poate fi reprezentat în plan printr-un punct (cerc etc.), eventual numerotat.

arc = pereche ordonată de noduri; poate fi reprezentată în plan printr-o sageata orientata

adiacență = proprietatea a două noduri de a fi unite prin arc; dacă $(x,y) \in U$, spunem că nodurile x și y sunt adiacente

incidență = proprietatea unei arce de a uni două noduri; dacă $(x,y) \in U$, spunem că arcul este incident cu nodul x .

gradul intern al nodului x = numărul de arce care intra în nodul x , notat cu $d^-(x)$ **gradul extern al**

nodului x = numărul de arce care ies din nodul x , notat cu $d^+(x)$

• **nod izolat** = nod cu gradul intern si extern 0; $d^-(x)=d^+(x)=0$

• **Propoziție:** În orice graf orientat cu n noduri și m arce, are loc egalitatea

Suma gradelor interioare = suma gradelor exterioare = numarul de arce

drum = succesiune de noduri cu proprietatea că oricare două noduri consecutive sunt adiacente (arcele pastreaza aceeasi orientare)

- *drum simplu* = drum în care fiecare arc apare o singură dată dar nodurile se pot repeta
- *drum elementar* = drum în care nodurile sunt distincte

circuit = drum în care primul nod coincide cu ultimul

- *circuit simplu* = circuit în care fiecare arc apare o singură dată dar nodurile se pot repeta
- *circuit elementar* = circuit în care nodurile sunt distincte, cu excepția primului și ultimului nod

lungimea unui drum/circuit = numărul de arce din care este format

graf parțial = graf care se obține din graful inițial prin eliminarea unor arce, nu și a nodurilor

subgraf = graf care se obține din graful inițial prin eliminarea unor noduri și a tuturor arcelor care au o extremitate în nodurile eliminate; nu pot fi eliminate alte arce decât cele cu legătura cu nodurile eliminate

Numărul grafurilor orientate cu n vârfuri este $2^{n(n-1)/2}$

tipuri particulare de grafuri

graf complet = graf în care orice două noduri distincte sunt adiacente (nu este unic, numărul de arce este cel mult $n^*(n-1)$)

graf plin = graf în care între orice două noduri distincte x și y există arc dus-întors (x, y) și (y, x)

Propoziție: numărul de arce într-un graf plin = $n^*(n-1)$

OBS: Numărul grafurilor orientate cu n vârfuri este $2^{n(n-1)} = 4^{n(n-1)/2}$

Numărul grafurilor orientate COMPLETE cu n vârfuri (există cel puțin un arc între oricare 2 noduri) este $3^{n(n-1)/2}$

graf tare conex = oricare ar fi două noduri distincte x și y, există drum DUS-INTORS de la x la y

componentă tare conexă = un subgraf tare conex și maximal în raport cu această proprietate (nu există drum între un nod din subgraf și un nod care nu aparține subgrafului)

Obs: un nod izolat constituie o componentă tare conexă

Metode de reprezentare a grafurilor orientate în memorie

1. Matricea de adiacență

- matricea nu este simetrică față de diagonala principala
- gardul exterior = numărul de valori 1 de pe linia x
- gradul interior = numărul de valori 1 de pe coloana x

2. Listele de adiacență

$L_i = \{j \in X / (i, j) \in U\}$

3. Lista de arce

$t \in M_{2xm}$, unde $m =$ numărul de arce din graf

$t_{1,k}$ și $t_{2,k} =$ extremitățile arcului k

$L_1: 2, 4 \quad L_4: 1, 2$

$L_2: 1, 3, 4 \quad L_5: 3, 6$

$L_3: 2, 5 \quad L_6: 5$

3. Arbori

Cerințele programei pentru BAC:

- terminologie (nod/vârf, muchie, radacina, descendant, descendant direct/fiu, ascendent, ascendent direct/parinte, frati, nod terminal, frunza)
- metode de reprezentare în memorie (matrice de adiacență, liste de descendenti, vectori de tati)

Definiție: Un **arbore** este un graf conex aciclic.

Teorema de caracterizare:

Urmatoarele afirmatii sunt echivalente:

- A este arbore cu n varfuri
- A este conex cu $n-1$ muchii
- A este aciclic cu $n-1$ muchii
- A este conex minimal (daca se elimina o muchie se distrug conexitatea)
- A este aciclic maximal (daca se adauga o muchie se formeaza un ciclu)

Proprietate: Oricare ar fi doua noduri distincte in arbore exista un lant elementar **unic** intre ele.

Definiție: Un **arbore cu radacina** este un arbore in care exista un nod special numit **radacina** iar toate celelalte noduri reprezinta descendenti directi sau indirecti ai radacinii.

descendent al nodului x = nod care se afla pe un lant elementar ce pleaca din x, altul decat cel care uneste radacina de x.

Fiu/descendent direct al nodului x = descendant al nodului x adjacent cu x (nod adjacent cu x care nu se afla pe lantul care uneste radacina de nodul x)

ascendent al nodului x = nod care se afla pe lantul elementar care uneste radacina de nodul x.

Parinte/tata/ascendent direct al nodului x = ascendent al nodului x adjacent cu x.

Frunza/terminal = nod care nu are descendenti (are gradul 1)

Adancime = lungimea lantului elementar maximal care uneste radacina cu o frunza

Arbore degenerat = arbore in care orice nod care nu este terminal are exact un descendant direct/fiu.

Arbore binar – arbore vid sau arbore în care orice nod are cel mult doi fii, intre care se face distinctie clara, fiu stang, fiu drept.

Inaltimea arborelui = numarul de muchii a lantului maxim de la radacina la o frunza

In orice arbore exista un lant elementar unic intre oricare doua varfuri

Metode de reprezentare a arborilor în memorie

Matricea de adiacență

1. Reprezentare ascendentă în arbori cu radacina - EFICIENTA

Tata(n), tata_i = Parintele nodului i daca i ≠ radacina; 0, altfel

ex: rad=2, tata = (2, 0, 2, 2, 3, 5)

2. Reprezentare descendenta în arbori cu radacina

fiu(n), L_i = lista fiilor nodului i, i ≠ frunza; 0, altfel

ex: rad=2,

Cuprins

1 Noțiuni preliminare și scurt istoric	3
1.1 Scurt istoric	3
1.2 Structura cursului	3
1.3 Notații generale. Notiuni recapitulative de combinatorică	4
2 Grafuri, digrafuri, multigrafuri și grafuri generale	7
2.1 Teorie	7
2.2 Problema săptămânii	10
2.3 Seminar	10
3 Subgrafuri și morfisme de grafuri	13
3.1 Teorie	13
3.1.1 Operații cu subgrafuri	14
3.1.2 Proprietăți	15
3.2 Problema săptămânii	16
3.3 Seminar	16
4 Grade și semigrade	17
4.1 Teorie	17
4.2 Problema săptămânii	20
4.3 Seminar	20
5 Drumuri, cicluri și circuite	21
5.1 Teorie	21
5.1.1 Conexitate	22
5.2 Problema săptămânii	22
5.3 Seminar	22
6 Conexitate	23
6.1 Teorie	23
6.2 Problema săptămânii	24
6.3 Seminar	24
7 Clase importante de grafuri	25
7.1 Teorie	26
7.2 Problema săptămânii	28
7.3 Seminar	28

8 Arbori. Arbori parțiali	29
8.1 Teorie	29
8.2 Problema săptămânii	30
8.3 Seminar	32
9 Arbori parțiali de cost minim. Algoritmul lui Kruskal/Prim	33
9.1 Teorie	33
9.2 Problema săptămânii	37
9.3 Seminar	37
10 Probleme de numărare a arborilor	39
10.1 Teorie	39
10.2 Problema săptămânii	40
10.3 Seminar	40
11 Algoritmi de căutare	41
11.1 Teorie	41
11.2 Problema săptămânii	41
11.3 Seminar	41
12 Probleme de drum minim (maxim)	43
12.1 Teorie	43
12.2 Problema săptămânii	46
12.3 Seminar	46
13 Algoritmii lui Dantzig și Ford, Dijkstra, Floyd și Warshall	47
13.1 Teorie	47
13.2 Problema săptămânii	49
13.3 Seminar	49
14 Metoda drumului critic	51
14.1 Teorie	51
14.2 Problema săptămânii	53
14.3 Seminar	53
15 Diagrame	55

Cursul 1

Noțiuni preliminare și scurt istoric

1.1	Scurt istoric	3
1.2	Structura cursului	3
1.3	Notății generale. Noțiuni recapitulative de combinatorică	4

1.1 Scurt istoric

Claude Berge (1926-2002) a fost un matematician francez, recunoscut drept unul din fondatorii combinatoricii și teoriei grafurilor. Acesta a jucat un rol major în renașterea combinatoricii, iar în teoria grafurilor a rămas cunoscut pentru conjectura grafurilor perfecte, rezolvată după câteva luni de la moartea sa.

1.2 Structura cursului

Scopul cursului constă în parcurgerea noțiunilor de bază legate de grafuri și prezentarea câtorva rezultate fundamentale ce au fost de mare folos în rezolvarea problemelor teoretice sau practice. Pe scurt, vom parcurge următoarele aspecte:

◊ Noțiuni fundamentale

- ◆₁ Grafuri, digrafuri, multigrafuri și grafuri generale
- ◆₂ Metode de reprezentare a grafurilor și digrafurilor
- ◆₃ Subgrafuri și morfisme de grafuri
- ◆₄ Grade și semigrade
- ◆₅ Drumuri, cicluri și circuite
- ◆₆ Conexitate
- ◆₇ Clase importante de grafuri: grafuri complete, grafuri planare, grafuri bipartite, grafuri regulate.

◊ Arbori

- ◆₁ Arbori. Arbori parțiali
- ◆₂ Arbori parțiali de cost minim. Algoritmii lui Kruskal/ Prim
- ◆₃ Probleme de numărare a arborilor

◊ Probleme de drum în grafuri și digrafuri

- ◆₁ Algoritmi de căutare
- ◆₂ Probleme de drum minim (maxim)
- ◆₃ Algoritmii lui Dantzig și Ford, Dijkstra, Floyd și Warshall
- ◆₄ Metoda drumului critic.

1.3 Notații generale. Notiuni recapitulative de combinatorică

Vom nota prin \mathbb{N} mulțimea numerelor naturale

$$\mathbb{N} := \{0, 1, 2, \dots, n, \dots\},$$

iar pentru $i \in \mathbb{N}$, notăm $\mathbb{N}_{\geq i} := \{i, i+1, \dots\}$. Vom obișnui să mai notăm

$$[n] := \{1, 2, \dots, n\} \quad \text{și} \quad [n]_0 := \{0, 1, 2, \dots, n\},$$

iar pentru $n = 0$, facem convenția $[0] := \emptyset$.

Oricare ar fi $x, y \in \mathbb{R}$ notăm

$$x \wedge y := \min \{x, y\} \quad \text{și} \quad x \vee y := \max \{x, y\}.$$

Oricare ar fi $x \in \mathbb{R}$ și $k \in \mathbb{N}$ definim **funcția factorială descrescătoră**

$$[x]_k := x(x-1)(x-2) \cdot \dots \cdot (x-k+1) =: x^{\bar{k}},$$

și **funcția factorială crescătoare**

$$[x]^k := x(x+1)(x+2) \cdot \dots \cdot (x+k-1) =: x^{\bar{k}},$$

iar pentru $k = 0$ considerăm prin convenție $[x]_0 = [x]^0 = 1$.

Vom nota prin

$$\sum_i x_1 \cdot \dots \cdot x_i$$

suma tuturor produselor de câte i numere din n numere x_1, x_2, \dots, x_n , unde $n \in \mathbb{N}$.

Pe tot parcursului cursului vom lucra doar cu mulțimi finite. Datează o mulțime finită A , vom nota cu $|A|$ cardinalul acesteia. Pentru două mulțimi finite A, B vom nota cu $\mathcal{F}(A, B)$ mulțimea tuturor funcțiilor definite pe A cu valori în B . Pe mulțimea $\mathcal{F}(A, B)$ definim **relația de echipotentă** prin

$$A \sim B : \Leftrightarrow \exists f \in \mathcal{F}(A, B), f \text{ funcție bijectivă.}$$

Putem defini acum notiunea de mulțime finită prin

Definiția 1 O mulțime A se numește finită dacă este vidă sau dacă este echipotentă cu mulțimea $\{1, 2, \dots, n\}$. O mulțimea este infinită dacă nu este finită.

Teorema 2 Dacă o mulțime este finită, atunci aceasta nu este echipotentă cu nici o submulțime proprie a sa.

Legat de mulțimile A și B vom folosi deseori

incluziunea	$\rightsquigarrow A \subseteq B : \Leftrightarrow (x \in A \Rightarrow x \in B)$
reuniunea	$\rightsquigarrow A \cup B := \{x \mid x \in A \text{ sau } x \in B\}$
intersectia	$\rightsquigarrow A \cap B := \{x \mid x \in A \text{ și } x \in B\}$
diferența	$\rightsquigarrow A - B := \{x \mid x \in A \text{ și } x \notin B\}$
reuniunea disjunctă	$\rightsquigarrow A \sqcup B := (A \cup B) - (A \cap B)$
diferența simetrică	$\rightsquigarrow A \Delta B := (A - B) \cup (B - A).$

Oricărei funcții $f : A \rightarrow B$, unde $A = \{a_1, a_2, \dots, a_m\}$ și $B = \{b_1, b_2, \dots, b_n\}$, $m, n \in \mathbb{N}^*$, ii putem **asocia** o aranjarea a obiectelor din A , în căsuțele din B astfel încât

$$\boxed{\text{căsuța } b_i} = \{a \in A \mid f(a) = b_i\}.$$

Această corespondență este o bijecție. Mai mult, putem stabili o corespondență bijectivă între mulțimea $\mathcal{F}(A, B)$ și mulțimea m -uplelor formate cu elemente din B sau a cuvintelor de m litere formate cu litere din mulțimea B :

$$\mathcal{F}(A, B) \ni f \longmapsto f(a_1) f(a_2) \dots f(a_m),$$

unde ordinea literelor este vitală. Deci, avem că

Propoziția 3 Cardinalul mulțimii $\mathcal{F}(A, B)$ este n^m .

Se numește **permutare** a mulțimii $A = \{a_1, a_2, \dots, a_m\}$ orice aplicație bijectivă $\sigma : A \rightarrow A$. Vom nota mulțimea tuturor permutărilor mulțimii A cu

$$S_m = \{\sigma \in \mathcal{F}(A, A) \mid \sigma \text{ funcție bijectivă}\}.$$

O partiție a mulțimii A este o descompunere de forma

$$A = A_1 \sqcup A_2 \sqcup \dots \sqcup A_k.$$

Dat un număr $k \in \mathbb{N}, 1 \leq m \leq n$, unde $m = |A|$, se numește **aranjament de n elemente luate câte m** orice **submulțime ordonată** alcătuită din m elemente ale lui A . Cu alte cuvinte, un aranjament de n elemente luate câte m este un cuvânt de lungime m format cu litere diferite din mulțimea B , iar numărul lor este

$$[n]_m := A_n^m = \frac{n!}{(n-m)!}.$$

Orice aranjament poate fi privit ca o funcție injectivă, iar un aranjament de m elemente luate câte m se numește permutare. Astfel, orice permutare poate fi privit ca o funcție bijectivă.

Observația 4 Numărul de posibilități de a introduce m bile, numerotate de la 1 la m , în n urne, numerotate de la 1 la n , este egal cu $[n]_m$. Numărul **funcțiilor injective** $f : A \rightarrow B$ este egal cu $[n]_m$. De vreme ce, în cazul finit, o funcție injectivă este tot una cu o funcție bijectivă deducem că numărul funcțiilor $[m]_m = m!$.

Dacă mulțimea B este o mulțimea ordonată astfel încât $b_1 < \dots < b_n$, un cuvânt de lungime n format cu litere din B , $b_{i_1} b_{i_2} \dots b_{i_k}$ se numește crescător dacă

$$b_{i_1} \leq b_{i_2} \leq \dots \leq b_{i_k},$$

și strict crescător dacă inegalitățile de mai sus sunt stricte. Cuvintele strict crescătoare de lungime m formate cu n simboluri se mai numesc **combinări de n luate câte m** , iar cele crescătoare se numesc **combinări cu repetiții de n luate câte m** .

Numărul de combinări de n luate câte m , respectiv, numărul de combinări cu repetiții de n luate câte m este

$$\binom{n}{m} := \frac{[n]_m}{m!} = \frac{n!}{m!(n-m)!}, \text{ respectiv, } {}^r\binom{n}{m} := \frac{[n]^m}{m!}.$$

Numărul $\binom{n}{m}$ se numește număr binomial și este egal și cu numărul submulțimilor cu m elemente ale unei mulțimi cu n elemente. Au loc formulele de calcul

$$\binom{n}{m} = \binom{n-1}{m} + \binom{n-1}{m-1}, \quad \forall a, b \in \mathbb{C} : (a+b)^n = \sum_{k=0}^n \binom{n}{k} a^k b^{n-k}.$$

Oricare ar fi p mulțimile finite A_1, A_2, \dots, A_p are loc **principiul includerii și excluderii**

$$|A_1 \cup A_2 \cup \dots \cup A_p| = \sum |A_i| - \sum |A_i \cap A_j| + \sum |A_i \cap A_j \cap A_k| - \dots + \sum (-1)^p |A_1 \cap \dots \cap A_p|.$$

Cu ajutorul acestui principiu se poate stabili că **numărul funcțiilor surjective** $f : A \rightarrow B, |A| = m, |B| = n, m, n \in \mathbb{N}^*, m \geq n$ este egal cu

$$n^m - \binom{n}{1} (n-1)^m + \binom{n}{2} (n-2)^m - \binom{n}{3} (n-3)^m + \dots + (-1)^{n-1} \binom{n}{n-1}.$$

Numim funcție generatoare asociată unui sir de numere (a_n) ca fiind suma seriei

$$\sum_{n=0}^{\infty} a_n x^n,$$

în ipoteza în care seria este convergentă pe \mathbb{R} sau pe o submulțime a sa.

Numărul lui Catalan reprezintă numărul de moduri în care se pot pune parantezele într-un produs neasociativ de n factori, scriși în ordinea a_1, \dots, a_n . Acest număr se poate calcula cu ajutorul formulei

$$\forall n \in \mathbb{N}^* : T(n) := \frac{1}{n} \binom{2n-2}{n-1}.$$

Cursul 2

Grafuri, digrafuri, multigrafuri și grafuri generale

2.1 Teorie	7
2.2 Problema săptămânii	10
2.3 Seminar	10

2.1 Teorie

Fie V o mulțime nevidă și finită, i.e. $V = \{v_1, v_2, \dots, v_n\}, n \in \mathbb{N}^*$. Mulțimea părților mulțimii V o vom nota cu $\mathcal{P}(V)$, iar mulțimea părților mulțimii V alcătuite din 2 elemente, respectiv, mulțimea perechilor ordonate formate cu elemente din X o vom nota cu

$$\mathcal{P}_2(V) := \{Y \subseteq V \mid |Y| = 2\}, \text{ respectiv, } \overrightarrow{\mathcal{P}_2(V)} := \{(x, y) \mid x, y \in V, x \neq y\}.$$

Tinând cont de formulele de calcul din Cursul 0, putem stabili următoarea

Propoziția 5 *Dacă $V \neq \emptyset$ și $|V| = n$, atunci*

$$|\mathcal{P}(V)| = 2^n, \quad |\mathcal{P}_2(V)| = 2^{C_n^2} = \sqrt{2^{n(n-1)}}, \quad |\overrightarrow{\mathcal{P}_2(V)}| = 2^{n(n-1)}. \quad (2.1)$$

Definiția 6 *Se numește **graf neorientat** o perechea ordonată de mulțimi $G = (V, E)$, unde V este o mulțimi nevidă, finită și $E \subseteq \mathcal{P}_2(V)$.*

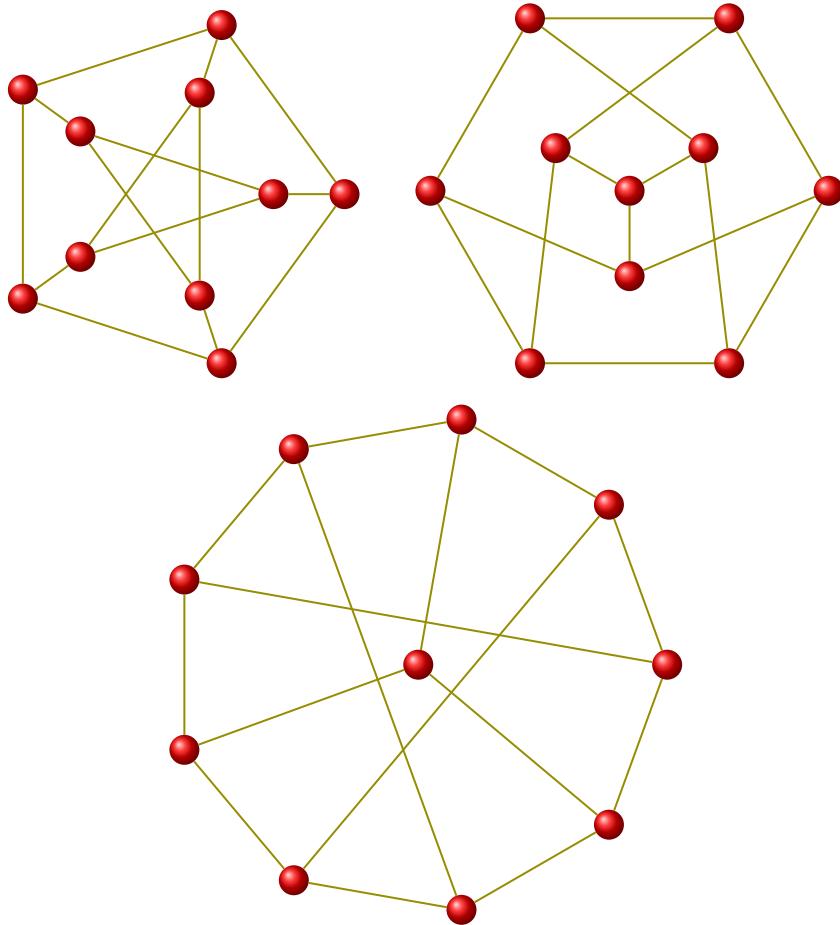
Elementele mulțimii V se numesc **vârfuri** (vertices) sau noduri ale grafului G , iar elementele lui E se numesc **muchii** (edges). Ordinul grafului $G = (V, E)$, notat cu $o(G)$, reprezintă cardinalul mulțimii V , iar dimensiunea grafului G , notată cu $\dim G$, reprezintă cardinalul mulțimii E . Dată o muchie $e = \{u, v\} \in E$, u, v se numesc **extremitățile** muchiei și o vom nota cu $[u, v]$. Mai mult, dacă $[u, v] \in E$, spunem că vârfurile u, v sunt **adiacente** în graful G și **incidente** în muchia $[u, v]$. Dacă $u = v$, muchia e se numește **bucătă**. Mulțimea

$$N(v) := \{u \in V \mid u, v \text{ sunt incidente în muchia } [u, v]\}$$

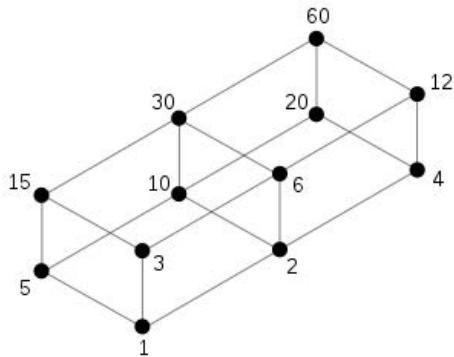
se numește vecinătatea vârfului v .

Dacă $E = \emptyset$, atunci graful (V, \emptyset) se numește graf **nul** pe care îl vom nota cu O_n , iar dacă $E = \mathcal{P}_2(V)$, atunci graful $(V, \mathcal{P}_2(V))$ se numește graf **complet** pe care le vom nota cu K_n .

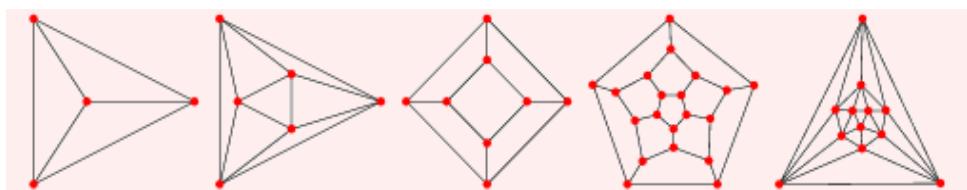
Exemplu 7 Vom reprezenta grafurile cu ajutorul unor desene alcătuite din puncte și linii continue ce vor lega anumite perechi de puncte. Grafuri Peterson



Exemplu 8 Orice mulțime ordonată (alfabetul limbii latine, mulțimea \mathbb{R} , laticele, clasa părților unei mulțimi etc.) este o latice.



Exemplu 9 Oricărui poliedru îi putem asocia un graf neorientat. În particular, pentru poliedrele Platon (tetraedru, cub, octaedru, dodecaedru, icosaedru) avem grafurile corespunzătoare



Exemplu 10 Graficul oricărei funcții numerice reale este un graf.

Exemplu 11 (din lumea reală) Arborele genealogic al unui individ dintr-o anumită populație (oameni sau animale), rețeaua de drumuri/căi ferate a unei țări, paginile Web etc.

Definiția 12 (construcții de grafuri) Fie $G_1 = (V_1, E_1)$ și $G_2 = (V_2, E_2)$ grafuri neorientate.

- (i) Numim complementarul grafului G_1 perechea ordonată de mulțimi $\overline{G}_1 := (V, \mathcal{P}_2(V) - E)$, i.e. este graful alcătuit din aceleași vârfuri și două vârfuri sunt adiacente în \overline{G} dacă și numai dacă nu sunt adiacente în G .
- (ii) Numim produsul cartezian al grafurilor G_1 și G_2 perechea ordonată $G_1 \times G_2 := (V_1 \times V_2, E)$, unde vârfurile $\{(v_1, v_2), (v'_1, v'_2)\}$ sunt adiacente în graful $G_1 \times G_2$ dacă și numai dacă

$$v_1 = v'_1 \text{ și } \{v_2, v'_2\} \in E_2 \quad \text{sau} \quad v_2 = v'_2 \text{ și } \{v_1, v'_1\} \in E_1.$$

Definiția 13 Se numește **graf orientat** sau **digraf** o perechea ordonată de mulțimi $G = (V, E)$, unde V este o mulțime nevidă, finită și $E \subseteq \overrightarrow{\mathcal{P}_2(V)}$.

Elementele mulțimii V se numesc **vârfuri** (vertices) sau noduri ale grafului G , iar elementele lui E se numesc **arc**. Ordinul grafului orientat $G = (V, E)$, notat cu $o(G)$ sau n_G sau $n(G)$, reprezintă cardinalul mulțimii V , iar dimensiunea grafului G , notată cu $\dim G$ sau m_G sau $m(G)$, reprezintă cardinalul mulțimii E . Dată o muchie $e = \{u, v\} \in E$, u, v se numesc **extremitățile** (inițiale/finale) muchiei și o vom nota cu $[u, v]$. Dacă $u = v$, muchia e se numește **bucă**.

Observația 14 Din punct de vedere al reprezentării grafice, digrafurile au în plus un sens de parcurs al arcelor.

Observația 15 În baza propoziției de la începutul cursului putem afirma că există $\sqrt{2^{n(n-1)}}$ grafuri cu n vârfuri, respectiv, $2^{n(n-1)}$ grafuri orientate cu n vârfuri.

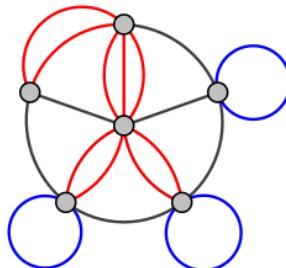
Definiția 16 (generalizări) (i) Se numește **graf multiplu** sau **multigraf** o pereche ordonată de mulțimi $G = (V, E)$, unde V este o mulțime nevidă, finită și E este o familie de elemente din $\mathcal{P}_2(V)$, i.e.

$$e \in E, \quad e = (\{u_i, v_i\})_{i \in \mathcal{I}}, \forall i \in \mathcal{I}: u_i, v_i \in V, u_i \neq v_i$$

(ii) Se numește **graf general** sau **pseudograf** o pereche ordonată de mulțimi $G = (V, E)$, unde V este o mulțime nevidă, finită și E este o familie de elemente din $\mathcal{P}_1(V) \cup \mathcal{P}_2(V)$.

(iii) Se numește **graf mixt** o pereche ordonată de mulțimi $G = (V, E \cup E')$, unde V este o mulțime nevidă, finită, $E \subseteq \mathcal{P}_2(V)$ și $E' \subseteq \overrightarrow{\mathcal{P}_2(V)}$.

Exemplu 17 Intuitiv un graf multiplu este un graf în care între două vârfuri pot există mai multe muchii/arce. Observăm că grafurile sunt multigrafuri muchii multiple sau pseudografuli fără muchii multiple și fără bucle.



2.2 Problema săptămânii

În secțiunea de față ne propunem să descriem o problemă interesantă care se poate soluționa cu ajutorul grafurilor.

Problema 18 (problema prieteniei a lui Paul Erdős) *Descrieți printre-un graf un grup de $2n+1$ persoane astfel încât oricare două persoane au exact un prieten comun în interiorul grupului.*

2.3 Seminar

Exercițiul 19 Arătați că într-un grup de mai mult de două persoane există întotdeauna două persoane cu același număr de prieteni din interiorul grupului.

Soluție 20 Fie $V = \{1, 2, \dots, n\}$, $n \in \mathbb{N}$, $n \geq 2$ fixat, un grup oarecare de persoane și $E \subseteq \mathcal{P}_2(V)$. O muchie $\{i, j\} \in E$, $i, j \in V$ reprezintă faptul că i și j sunt prieteni. Formal ne propunem să arătăm că

$$\exists i, j \in V : i \neq j \Rightarrow |N(i)| = |N(j)|,$$

unde $N(i) = \{j \in V \mid \{i, j\} \in E\}$. Dacă presupunem pentru reducere la absurd că

$$\forall i, j \in V : i \neq j \Rightarrow |N(i)| \neq |N(j)|$$

cea ce este echivalent cu injectivitatea funcției

$$N : \{1, 2, \dots, n\} \rightarrow \{0, 1, \dots, n-1\}.$$

Injectivitatea, surjectivitatea și bijectivitatea coincid în cazul în care funcția în discuție este definită între mulțimi finite cu același număr de elemente. Prin urmare, N este surjectivă, i.e.

$$\exists i, j \in V : |N(i)| = 0 \quad \text{și} \quad |N(j)| = n-1$$

Exercițiul 21 Construiți graful complementar asociat grafului

$$G = (\{1, 2, 3, 4, 5, 6\}, \{\{1, 2\}, \{1, 4\}, \{2, 5\}, \{3, 5\}, \{3, 6\}, \{5, 6\}\})$$

și produsul cartezian al grafurilor $G_1 = (\{1, 2, 3, 4\}, \{\{1, 2\}, \{2, 3\}, \{3, 4\}, \{4, 1\}\})$ și $G_2 = (\{z, y, x\}, \{\{z, y\}, \{y, x\}, \{x, z\}\})$.

Soluție 22 ...

Exercițiul 23 Să se deseneze graful corespunzător clasei părților unei mulțimi finite A . Să se calculeze mărimea acestui graf?

Soluție 24 Graful corespunzător clasei părților mulțimii A , $\mathcal{P}(A)$, este $G = (V, E)$ unde $V = \mathcal{P}(A)$ și $\{A_1, A_2\} \in E$ dacă și numai dacă

$$A_1 \subseteq A_2 \quad \text{și} \quad \nexists A_3 \in V : A_1 \subset A_3 \subset A_2$$

sau

$$A_2 \subseteq A_1 \quad \text{și} \quad \nexists A_3 \in V : A_2 \subset A_3 \subset A_1.$$

Urmărind cu atenție desenul de mai sus putem deduce următoarea formulă de calcul pentru mărimea grafului G

$$|E| = m(G) = \sum_{k=0}^n \binom{n}{k} k = n2^{n-1}.$$

Exercițiul 25 Să se deseneze graful corespunzător divizorilor unui număr natural $n \geq 2$. Să se calculeze mărimea acestui graf?

Soluție 26 Trebuie să realizăm desenul grafului $G = (V, E)$, unde $V = \{d \mid d \text{ divide } n\}$ și $\{d_1, d_2\} \in E$ dacă și numai dacă

$$d_1 \mid d_2 \text{ și } \nexists d_3 \in V - \{d_1, d_2\} : d_1 \mid d_3 \mid d_2$$

sau

$$d_2 \mid d_1 \text{ și } \nexists d_3 \in V - \{d_1, d_2\} : d_2 \mid d_3 \mid d_1.$$

Dat un $n \in \mathbb{N}, n \geq 2$, avem $n = p_1^{\alpha_1} p_2^{\alpha_2} \cdot \dots \cdot p_k^{\alpha_k}$ și

$$|V| = \prod_{i=1}^k (\alpha_i + 1) \quad \text{și} \quad |E| = \sum_{d|n} f(d),$$

unde $f(d)$ reprezintă numărul de factori primi din descompunerea lui $d = p_1^{\beta_1} p_2^{\beta_2} \cdot \dots \cdot p_k^{\beta_k}, 0 \leq \beta_j \leq \alpha_j, j \in \overline{1, k}$. De exemplu, avem $f(1) = 0, \sum \alpha_j$ divizori de forma

$$d = p_j^{\beta_j}, \quad 1 \leq \beta_j \leq \alpha_j, j \in \overline{1, k},$$

$\sum \alpha_i \alpha_j$ divizori de forma

$$d = p_j^{\beta_j} p_i^{\beta_i}, \quad 1 \leq \beta_i \leq \alpha_i, 1 \leq \beta_j \leq \alpha_j, i, j \in \overline{1, k},$$

și inductiv $\alpha_1 \alpha_2 \dots \alpha_k$ divizori de forma

$$d = p_1^{\beta_1} p_2^{\beta_2} \cdot \dots \cdot p_k^{\beta_k}, \quad 1 \leq \beta_j \leq \alpha_j, j \in \overline{1, k}.$$

Prin urmare...

Exercițiu 27 Să se deseneze graful corespunzător comutativității unui grup. Să se calculeze mărimea acestui graf? Să se arate că în cazul grupurilor abeliene acest graf coincide cu K_n .

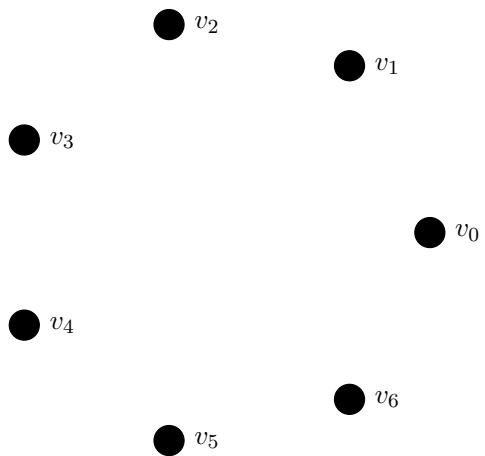
Soluție 28 Dat un grup (H, \cdot) graful comutativității acestuia este $G = (V, E)$, unde $V = H$ și, oricare ar fi $x, y \in H, \{x, y\} \in E$ dacă și numai dacă $xy = yx$.

Exercițiu 29 Într-o țară sunt 2000 de orașe și niciun drum între acestea. Să se arate că se poate construi o rețea de drumuri astfel încât din 2 orașe să pornească câte un drum, din alte 2 orașe să pornească câte 2 drumuri și tot așa până când la ultimele 2 din care vor porni 1000 de drumuri.

Soluție 30 Există cel puțin două abordări ale acestei probleme dacă la o olimpiadă desfășurată în orașul Sankt-Petersburg, în anul 2001. Prima din ele constă în demonstrarea prin inducție a unei afirmații generale din care vom regăsi problema noastră drept un caz particular. Cea de-a doua utilizează noțiunea de graf bipartit.

Demonstrăm prin inducție după numărul de vârfuri că există un graf cu $4n$ vârfuri astfel încât gradul a două vârfuri să fie 1, gradul altor două să fie 2, ..., gradul ultimelor două vârfuri să fie 2. Pentru $n = 1$, luăm 4 vârfuri v_1, v_2, v_3, v_4 și construim muchiile $\{v_1, v_2\}, \{v_2, v_3\}$ și $\{v_3, v_4\}$. Astfel v_1 și v_4 au gradul 1, iar v_2 și v_3 au gradul 2.

Presupunem acum că afirmația este adeverată pentru un graf cu $4n$ vârfuri și o vom demonstra pentru un graf cu $4n + 4$ vârfuri. Fie G un graf cu $4n$ vârfuri care satisface proprietatea enunțată.



Exercițiul 31 Vlad a observat că printre cei 25 de colegi de clasă ai lui nu există 2 care să aibă același număr de prieteni, în schimb fiecare coleg are cel puțin un prieten. Câți prieteni are Vlad?

Cursul 3

Subgrafuri și morfisme de grafuri

3.1 Teorie	13
3.1.1 Operații cu subgrafuri . .	14
3.1.2 Proprietăți	15
3.2 Problema săptămânii	16
3.3 Seminar	16

3.1 Teorie

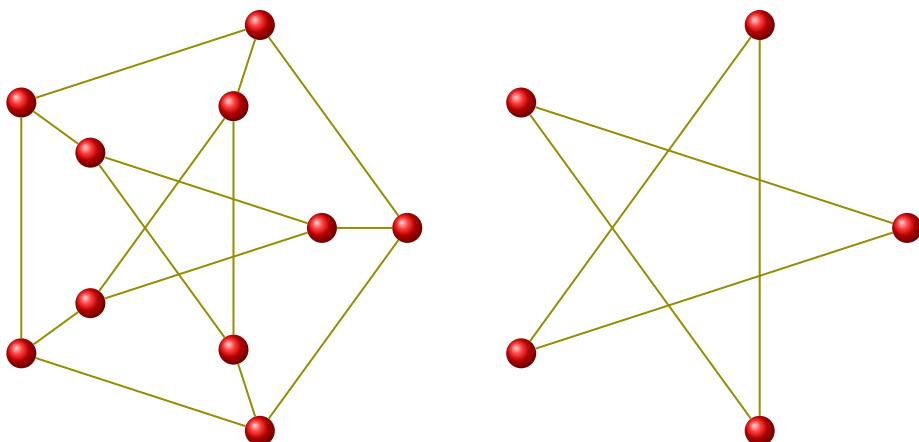
Definiția 32 Fie $G = (V, E)$ un graf neorientat.

- Se numește **subgraf** al lui G un graf $G' = (V', E')$, unde $V' \subset V$ și $V' \subset E$.
- Se numește **subgraf** al lui G **generat** de $V' \subset V$, **graful** $G(V') = (V', E \cap \mathcal{P}_2(V'))$.
- Se numește **subgraf** al lui G **generat** de $E' \subset E$, **graful** $G(E') = (U, E')$, unde U este mulțimea vârfurilor din V unite prin muchii din E' .

Observația 33 Pe mulțimea subgrafurilor lui $G = (V, E)$ avem relația de ordine

$$G' = (V', E') \leq G'' = (V'', E'') \Leftrightarrow V' \subset V'' \quad \text{și} \quad E' \subset E''.$$

Observația 34 Dacă $V' = V$, atunci $G' = (V, E')$, $E' \subset E$, se numește **graf parțial**. Dacă $E' = E$, atunci $G' = (V', E)$, $V' \subset V$, se numește **subgraf total/plin** al lui G .



3.1.1 Operații cu subgrafuri

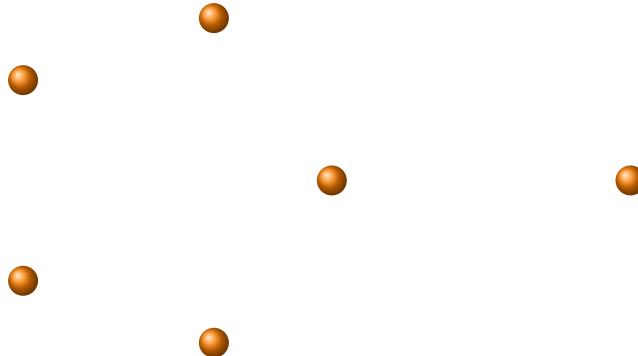
Fie $G = (V, E)$, $G_1 = (V_1, E_1)$ grafuri neorientate, $V' \subset V$ și $E' \subset E$.

- $G - V' \rightsquigarrow$ graful care se obține eliminând din G toate vârfurile din V' și toate muchiile ce au măcar o extremitate în V' . În particular, pentru $V' = \{v\}$, avem $G - B' := G - v$.
- $G - E' \rightsquigarrow$ graful care se obține eliminând din G toate muchiile din E' , dar păstrând toate vârfurile. În particular, pentru $E' = \{e\}$, avem $G - E' = G - e$.
- $G + U \rightsquigarrow$ graful obținut prin adăugarea elementelor din U drept vârfuri izolate, unde U este o mulțime astfel încât $U \cap V = \emptyset$
- $G + G_1 \rightsquigarrow$ graful $(V \cup V_1, E \cup E_1)$, impunând condiția $E \cap E_1 = \emptyset$.

Definiția 35 Fie $G = (V, E)$ un graf și $U \subset V$.

- Dacă $G(U) = (U, \emptyset)$, i.e. $G(U)$ este graful nul, atunci U se numește **mulțime stabilă** a lui G . Numărul maxim de elemente ale unei mulțimi stable se numește **număr de stabilitate**.
- Dacă $G(U) = (U, \mathcal{P}_2(U))$, i.e. $G(U)$ este graf complet, atunci U se numește **clică** a lui G . Numărul maxim de elemente ale unei clici se numește **densitatea** lui G , notat cu $\rho(G)$.

Exemplu 36 Considerând graful din figura de mai jos,



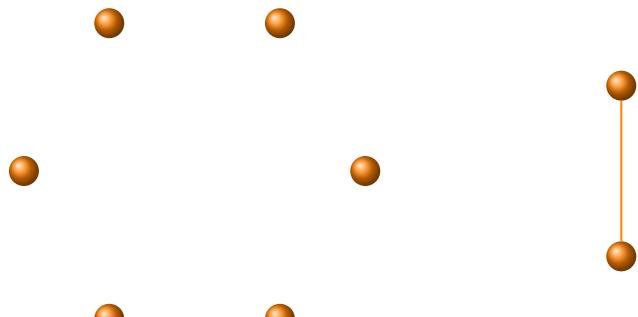
constatăm că mulțimile $\{3, 4\}$, $\{1, 6\}$, $\{1, 4, 6\}$ etc. sunt stable, iar numărul de stabilitate este 3, mulțimile $\{1, 2\}$, $\{1, 2, 3\}$ etc. sunt clici, iar densitatea lui G este 3.

Definiția 37 Fie $G = (V, E)$ și $G' = (V', E')$ două grafuri. Se numește morfism de grafuri de la G la G' o funcție $f : V \rightarrow V'$ astfel încât

$$\forall e = \{u, v\} \in E : f(e) = \{f(u), f(v)\} \in E'.$$

Exemplu 38 Date grafurile din figurile de mai jos, putem defini morfismul $f : \{1, 2, 3, 4, 5\} \rightarrow \{1, 2\}$ prin

$$f(1) = f(2) = f(3) = f(4) = 1 \quad \text{și} \quad f(5) = f(6) = 2.$$



3.1.2 Proprietăți

1. Orice morfism de grafuri invariază relațiile de incidentă și de adiacență.
2. Componerea a două morfisme de grafuri este un morfism de grafuri.

Utilizând morfismele de grafuri, încercăm în cele ce urmează să dăm o teoremă de caracterizare a grafurilor colorabile în sensul definiției următoare.

Definiția 39 Fie $G = (V, E)$ un graf și $r \in \mathbb{N}^*$.

- Se numește **colorare proprie** a lui G o atribuire de culori vârfurilor lui G astfel încât toate vârfurile adiacente să fie colorate diferit.
- Se numește **număr cromatic** al lui G numărul minim de culori necesare pentru o colorare proprie. Acest număr îl vom nota cu $\chi(G)$.
- G se numește r – **colorabil** dacă admite o colorare cu r culori.

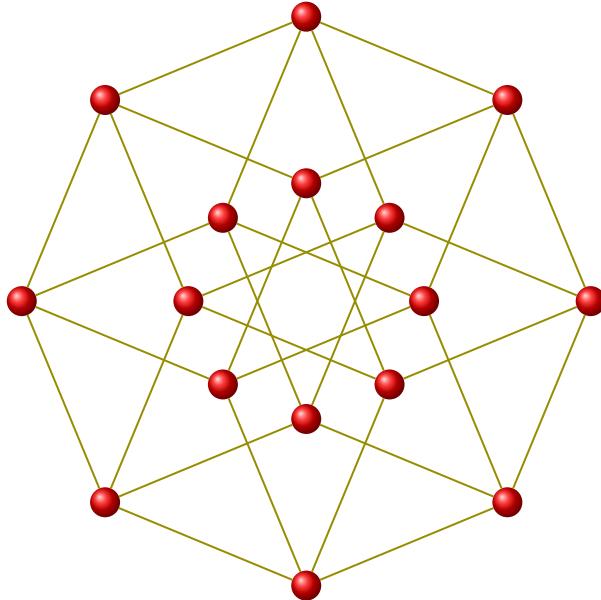
Teorema 40 Un graf $G = (V, E)$ este r – colorabil dacă și numai dacă există un morfism de la G la K_r .

Demonstrație.

... ■

Exemplu 41 Grafurile 1 – colorabile sunt cele formate din vârfuri izolate. Cubul, în general hipercubul, este un graf 2 – colorabil, morfismul fiind $f : G \rightarrow K_2$ definit prin

$$f(1) = f(3) = f(6) = f(8) = 1 \quad \text{și} \quad f(2) = f(4) = f(5) = f(7) = 2.$$



Graful lui Peterson este un graf 3 – colorabil, iar morfismul este $f : G \rightarrow K_3$, definit prin

$$f(1) = f(4) = f(6) = 1, \quad f(2) = f(5) = f(8) = f(9) = 2, \quad f(3) = f(7) = 3.$$

Una din probleme de colecție ale matematicii ce are legătură cu această teoremă este **problema celor 4 culori** (pentru mai multe detalii vezi secțiunea următoare).

Definiția 42 Fie $G = (V, E)$ un graf și $G' = (V', E')$. Se numește **izomorfism** de la G la G' o funcție bijectivă $f : V \rightarrow V'$ care este morfism. Dacă $G = G'$, atunci f se va numi **automorfism** a lui G , iar cu $\text{Aut}(G)$ vom nota mulțimea automorfismelor lui G .

Observația 43 $\text{Aut}(G), \circ)$ este grup.

Exemplu 44 Orice permutare σ din S_n este automorfism al grafului complet cu n vârfuri, K_n . Deci, $\text{Aut}(K_n) \cong S_n$.

Exemplu 45 Grupul automorfismelor grafului ciclic cu n vârfuri, C_n , este izomorf cu grupul diedral D_{2n} . De exemplu, pentru $n = 4$, avem

$$\text{Aut}(C_4) = \left\{ \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \rho = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 1 \end{pmatrix}, \varepsilon = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}, \rho^2, \rho^3, \rho\varepsilon, \rho^2\varepsilon, \rho^3\varepsilon \right\}$$

Exemplu 46 Graful A_n definit prin

$$\left\{ \begin{array}{l} V = \{a_i, b_i, c_i \mid 0 \leq i < n\} \\ E = \{(a_i, a_{i+1}), (a_i, b_i), (a_i, c_i), (b_i, c_i), (c_i, a_{i+1}) \mid 0 \leq i < n\} \end{array} \right.,$$

i.e. A_n este un poligon cu n muchii, iar pe fiecare muchie este desenat un dreptunghi împreună cu o diagonală a acestuia. Grupul automorfismelor lui A_n este izomorf cu \mathbb{Z}_n .

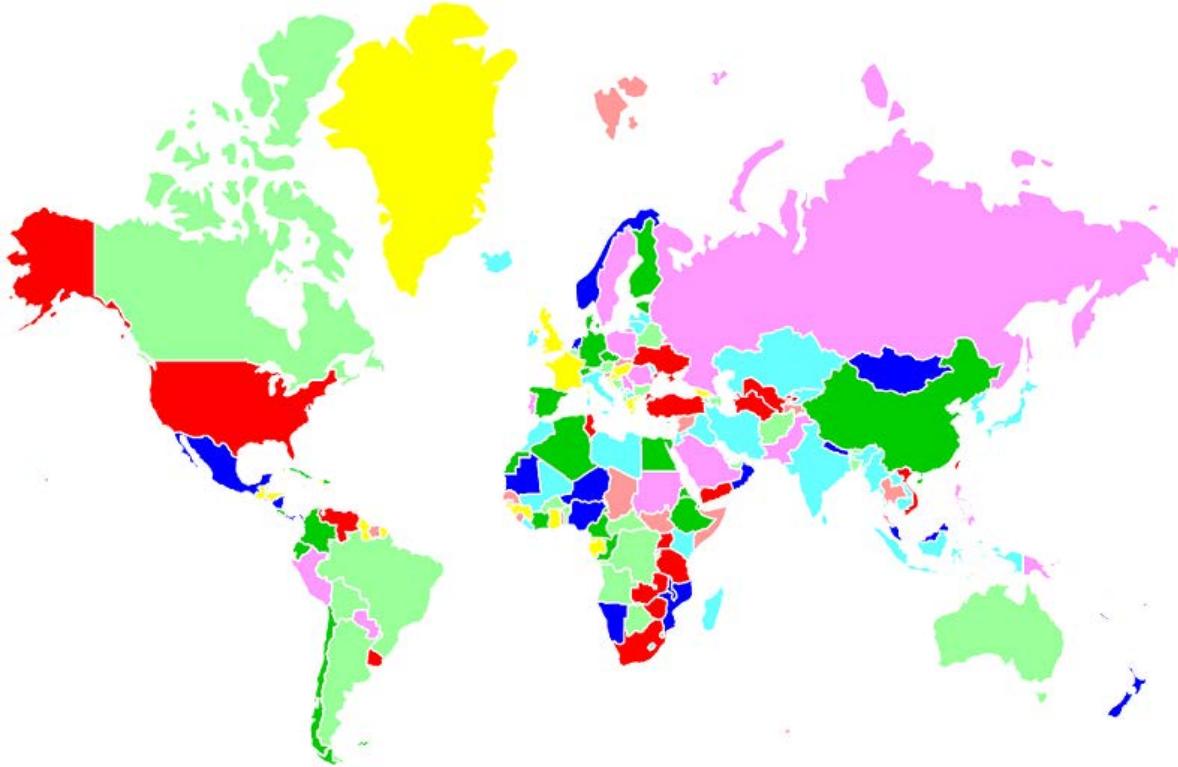
Exemplu 47 Automorfismele grafului G , definit prin $V = \{1, 2, 3, 4\}$, $E = \{\{1, 2\}\}$, este format din permutările

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 3 & 4 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 3 & 4 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 1 & 2 & 4 & 3 \end{pmatrix}, \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}.$$

Prin urmare, $\text{Aut}(G)$ este izomorf cu grupul lui Klein.

3.2 Problema săptămânii

Problema 48 (De Morgan, 1852/soluție computațională 1976) Dată o hartă, se pot colora țările acesteia cu cel mult 4 culori astfel încât oricare două țări vecine să fie colorate diferit.



3.3 Seminar

Cursul 4

Grade și semigrade

4.1 Teorie	17
4.2 Problema săptămânii	20
4.3 Seminar	20

4.1 Teorie

Definiția 49 Fie $G = (V, E)$ un graf și $v \in V$. Definim **gradul** (valența) vârfului v ca fiind numărul natural

$$d_G(v) := \text{card} \{e \in E \mid v \in e\}. \quad (4.1)$$

Observația 50 Gradul unui vârf oarecare v se poate la fel de bine defini prin

$$d_G(v) := \text{card} \{u \in V \mid \{u, v\} \in E\}.$$

Dacă $d_G(v) = 0$, atunci v este vârf izolat, iar dacă $d_G(v) = 1$, atunci v se numește vârf pendant.

Observația 51 Definiția de mai sus poate fi extinsă la pseudograuri și multigrauri.

Calculând cardinalul mulțimii $M = \{(v, e) \in V \times E \mid e \in E\}$ în două moduri, i.e.

$$\text{card } M = \sum_{v \in V} \text{card} \{e \in E \mid v \in e\} = \sum_{v \in V} d_G(v)$$

și

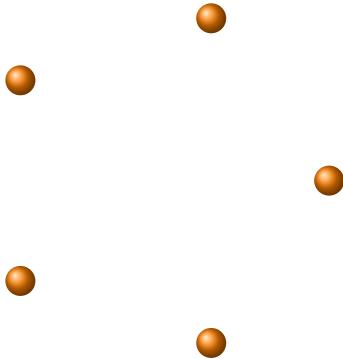
$$\text{card } M = \sum_{e \in E} \text{card} \{v \in V \mid v \in e\} = \sum_{e \in E} 2 = 2 \text{ card } E,$$

obținem un rezultat prin rezultat remarcabil.

Teorema 52 (Euler) Dacă $G = (V, E)$ este graf, atunci are loc egalitatea

$$\sum_{v \in V} d_G(v) = 2 \text{ card } E. \quad (4.2)$$

Corolar 53 În orice graf numărul vârfurilor de grad impar este par, după cum se poate vedea în exemplul următor



Definiția 54 Fie $G = (V, E)$ un graf.

- Numărul $\delta_G := \min \{d_G(v) \mid v \in V\}$ se numește gradul minim al lui G .
- Numărul $\Delta_G := \max \{d_G(v) \mid v \in V\}$ se numește gradul maxim al lui G .
- Numărul

$$d_G := \frac{1}{|V|} \sum_{v \in V} d_G(v), \quad |V| := \text{card } V,$$

se numește gradul mediu al lui G .

Se poate observa cu ușurință că

$$\forall v \in V : \delta_G \leq d_G(v) \leq \Delta_G \quad \text{și} \quad d_G = \frac{2|E|}{|V|} = \frac{2m_G}{2n_G},$$

unde m_G reprezintă numărul de muchii din graful F , iar n_G numărul de vârfuri/noduri din graful G .

Definiția 55 Fie $G = (V, E)$ un digraf și $v \in V$.

- Numim **semigrad exterior** al vârfului v ca fiind numărul natural

$$d_G^+(v) := \text{card} \{e \in E \mid e = \{v, u\}, u \in V\}. \quad (4.3)$$

- Numim **semigrad interior** al vârfului v ca fiind numărul natural

$$d_G^-(v) := \text{card} \{e \in E \mid e = \{u, v\}, u \in V\}. \quad (4.4)$$

Cu alte cuvinte, dacă $d_G^+(v) = 1$, atunci vârful v este extremitate inițială a unei singure muchii, iar dacă $d_G^-(v) = 1$, atunci v este extremitate finală a unei singure muchii. Teorema lui Euler în cazul digrafilor are forma

$$\sum_{v \in V} d_G^+(v) = \sum_{v \in V} d_G^-(v) = |E|. \quad (4.5)$$

Definiția 56 Fie $r \in \mathbb{N}$. Un graf $G = (V, E)$ se numește r – regulat dacă toate vârfurile au gradul r , i.e.

$$\forall v \in V : d_G(v) = r.$$

Exemplul standard de graf $(n - 1)$ – regulat este graful complet cu n vârfuri, K_n .

Teorema 57 Fie $r \in \mathbb{N}$. Dacă $G = (V, E)$ este graf r – regulat de ordin n , atunci

$$r \in \{0, 1, \dots, n-1\} \quad \text{și} \quad n \cdot r \in 2\mathbb{N}.$$

Reciproc, oricare ar fi $n \in \mathbb{N}^*$ și $r \in \overline{0, n-1}$ cu proprietatea că $n \cdot r \in 2\mathbb{N}$, există un graf r – regulat de ordin n .

Demonstrație. ■

Observația 58 Teorema de mai sus ne oferă odată cu demonstrația sa și un algoritm de construcție a unui graf r – regulat. De exemplu, dorim să construim un graf r – regulat de ordinul n . Fie mulțimea $M = \{\pm\hat{1}, \pm\hat{2}, \dots, \pm\hat{q}\}$, dacă $r = 2q$, sau $M = \{\pm\hat{1}, \pm\hat{2}, \dots, \pm\hat{q}, m\}$, dacă $r = 2q + 1$ și $n = 2m$. Atunci construind graful cu vârfurile $V = \mathbb{Z}_6$ și muchiile obținute prin

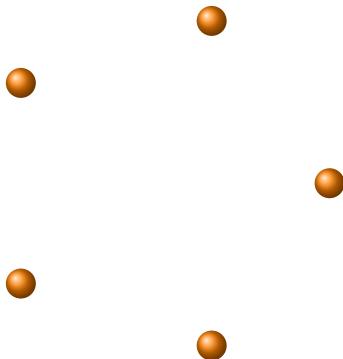
$$\forall x, y \in \mathbb{Z}_n : \{x, y\} \in E \Leftrightarrow x - y \in M, \quad (4.6)$$

vom obține un graf care r – regulat de ordinul n .

Definiția 59 Fie $n \in \mathbb{N}^*$ și $\{d_i \in \mathbb{N} \mid i \in \overline{1, n}\}$ astfel încât $d_1 \geq d_2 \geq \dots \geq d_n$. Spunem că d_1, d_2, \dots, d_n este un **șir de grade** / șir grafic dacă există un graf $G = (V, E)$ cu $V = \{v_i \mid i \in \overline{1, n}\}$ astfel încât

$$\forall i \in \overline{1, n} : d_G(v_i) = d_i.$$

De exemplu, sirul $4, 3, 3, 3, 1$ este un șir grafic, iar graful corespunzător este



Criteriu 60 Pentru rezolvarea problemei șirului grafic trebuie să verificăm unul din următoarele două cazuri

1. Fie $n \geq 2$ și $d_1, d_2, \dots, d_n \in \mathbb{N}$ cu $d_1 \geq \dots \geq d_n$ și $d_1 \neq 0$. Atunci $\{d_i \mid i \in \overline{1, n}\}$ este șir grafic dacă și numai dacă

$$d_2 - 1, \quad d_3 - 1, \dots, \quad d_{d_1+1} - 1, \quad d_{d_1+2} - 1, \dots, d_n$$

este șir grafic.

2. Fie $n \geq 2$ și $d_1, d_2, \dots, d_n \in \mathbb{N}$ cu $d_1 \geq \dots \geq d_n$. Atunci $\{d_i \mid i \in \overline{1, n}\}$ este șir grafic dacă și numai dacă

$$\sum_{i=1}^n d_i \in 2\mathbb{N} \quad \text{și} \quad \forall k \in \overline{1, n-1} : \sum_{i=1}^n d_i \leq k(k-1) + \sum_{i=k+1}^n \min\{k, d_i\}.$$

4.2 Problema săptămânii

Exercițiu 61 Studiați existența unui graf pentru care gradele vârfurilor sunt

Problema 62 (i) 4, 3, 3, 3, 1.v (zmeu+o muchie exterioară)

(ii) 4, 3, 2, 2, 1.v

(iii) 4, 3, 1, 1, 1.x

(iv) 4, 3, 2, 1, 1.x

(v) 5, 3, 2, 1, 1.x

Problema 63 În general, date numerele naturale $d_1 \geq \dots \geq d_n$, să se decidă dacă există un graf $G = (V, E)$ cu $V = \{d_i \mid i \in \overline{1, n}\}$ și

$$\forall i \in \overline{1, n} : d_G(v_i) = d_i.$$

4.3 Seminar

Cursul 5

Drumuri, cicluri și circuite

5.1 Teorie	21
5.1.1 Conexitate	22
5.2 Problema săptămânii	22
5.3 Seminar	22

5.1 Teorie

Definiția 64 Fie $G = (V, E)$ un graf.

A

- Se numește **rută** în G o secvență de tipul

$$\mu = (v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1}), \quad \begin{cases} v_i \in V & , \quad i \in \overline{1, k+1} \\ e_j \in E & , \quad j \in \overline{1, k} \end{cases} .$$

Vom nota cu $l(\mu) = k$ lungimea rutei.

- Se numește **rută simplă** în G o rută pentru care toate muchiile sunt distincte.
- Se numește **drum** în G o rută pentru care toate vârfurile sunt distincte (eventual cu excepția extremităților).
- Se numește **rută ciclică** în G o rută $\mu = (v_1, e_1, v_2, e_2, \dots, v_k, e_k, v_{k+1})$ în care $v_1 = v_{k+1}$.

B

- Se numește **ciclu** în G o rută simplă ciclică.
- Se numește **ciclu elementar** în G un ciclu pentru care și muchiile și vârfurile sunt distincte două câte două (cu excepția extremităților).
- Se numește **ciclu eulerian** în G un ciclu ce conține toate muchiile lui G .
- Se numește **ciclu hamiltonian** în G un ciclu ce conține toate vârfurile lui G .

Exemplu 65 Abundență... Trapez și un vârf izolat- ciclu eulerian, ciclul

$$\mu = (1, 2, 3, 4, 8, 7, 6, 5, 1)$$

este un ciclu hamiltonian în cub.

Teorema 66 Fie $G = (V, E)$ un graf. Dacă, pentru toate vârfurile din V , $d_G(v) \geq 2m$, atunci G conține cicluri.

Teorema 67 Fie $G = (V, E)$ un graf astfel încât $\text{card } V \geq 3$. Dacă $\text{card } E \geq \text{card } V$, atunci G conține cicluri.

Teorema 68 Fie $G = (V, E)$ un graf. Dacă există $u, v \in V$, distincte, astfel încât între u și v există două drumuri diferite, atunci G admite cicluri.

Observația 69 Toate conceptele definite mai sus se pot generaliza pentru digrafuri, multigrafuri și pseudografuri. Bunăoară, în cazul digrafilor un ciclu poartă denumirea de **circuit**. Se poate arăta că un digraf $G = (V, E)$ admite circuite dacă

$$\forall v \in V : d_G^+(v) \geq 1 \quad \text{sau} \quad d_G^-(v) \geq 1.$$

Teorema 70 (criteriu grafuri bipartite) Un graf este bipartit dacă și numai dacă nu conține cicluri de lungime impară.

Pentru o aplicație interesantă a acestui rezultat merită să parcugem problema din secțiunea următoare.

5.1.1 Conexitate

Definiția 71 Un graf $G = (V, E)$ se numește conex dacă între oricare două vârfuri distincte există cel puțin un drum.

Observația 72 Dacă G este un graf oarecare, atunci putem defini pe V relația \sim prin

$$\forall u, v \in V : u \sim v \Leftrightarrow \exists \mu - \text{deum de la } u \text{ la } v.$$

Se arată imediat că \sim este o relație de echivalență, iar clasele de echivalență le vom numi componente conexe ale lui G (evidenț sunt grafuri conexe).

Să presupunem că G este conex. Fie $v_0 \in V$ și definim mulțimile

$$V_1 := \{v \in V \mid \exists \mu \text{ drum de lungime impară de la } v_0 \text{ la } v\}$$

și

$$V_2 := \{v \in V \mid \exists \mu \text{ drum de lungime pară de la } v_0 \text{ la } v\}.$$

Atunci se poate arăta că (V_1, V_2) este o bipartiție a lui V .

5.2 Problema săptămânii

Problema 73 (Hamilton) Fie $n \in \mathbb{N}$. Fie o tablă de șah cu n^2 pătrătele. Se poate ca un cal să rindă în L să plece de pe o anumită poziție, să treacă prin toate pătratele tablei și în final să se întoarcă în poziția inițială?

Soluție 74 Dacă $n \in 2\mathbb{N} + 1$, nu deoarece fiind un graf bipartit nu există cicluri de lungime impară (n^2 este impar). În rest, dacă $n \geq 6$ și $n \in 2\mathbb{N}$, atunci răspunsul este afirmativ.

.....

5.3 Seminar

Cursul 6

Conexitate

6.1 Teorie	23
6.2 Problema săptămânii	24
6.3 Seminar	24

6.1 Teorie

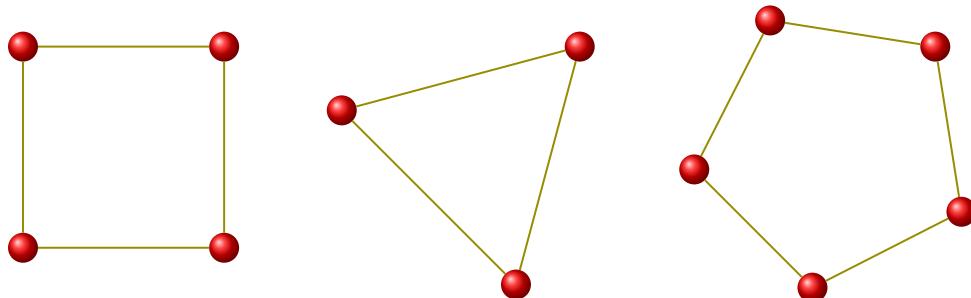
Definiția 75 Un graf $G = (V, E)$ se numește conex dacă între oricare două vârfuri distincte există cel puțin un drum.

Observația 76 Dacă G este un graf oarecare, atunci putem defini pe V relația \sim prin

$$\forall u, v \in V : u \sim v \Leftrightarrow \exists \mu - deum de la u la v.$$

Se arată imediat că \sim este o relație de echivalență, iar clasele de echivalență le vom numi componente conexe ale lui G (evident sunt grafuri conexe). Vom nota numărul total al componentelor conexe sau cardinalul mulțimii factor prin $p(G)$.

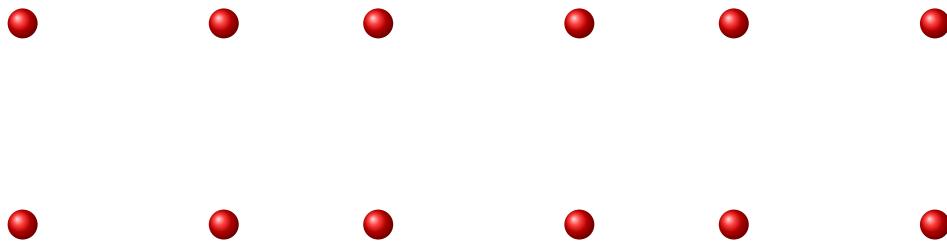
Graful din figura de mai jos conține 3 componente conexe.



Definiția 77 Un digraf $G = (V, E)$ se numește

- **tare conex** dacă $\forall u, v \in V, u \neq v$, există cel puțin un drum de la u la v .
- **unilateral conex** dacă $\forall u, v \in V, u \neq v$, există cel puțin un drum de la u la v sau de la v la u .
- **conex** dacă graful suport al lui G , $G' = (V, E')$, $E' = \{\{u, v\} \mid (u, v) \in E\}$, este conex.

Exemplu 78 În figura de mai jos regăsim 3 exemple, câte unul pentru fiecare noțiune introdusă în definiția anterioară.



Teorema 79 (legătura dintre $|E|, |V|$ și $p(G)$) Fie $G = (V, E)$ un graf, $n = |V|, m = |E|$ și $p = p(G)$. Atunci avem estimarea

$$n - p \leq m \leq C_{n-p+1}^2. \quad (6.1)$$

Demonstrație. Inducție după n ... ■

Teorema 80 Un graf $G = (V, E)$, cu $n = |V|, m = |E|$ și $p = p(G)$, nu admite cicluri dacă și numai dacă $m - n + p = 0$.

Observația 81 Notăm că numărul $m - n + p$ se numește **număr ciclomatic** al lui G .

Lemă 82 Fie $G = (V, E)$ un graf cu proprietatea că, pentru toți $v \in V$, $2 \leq d_G(v) \in 2\mathbb{N}$. Atunci orice muchie a lui G este conținută într-un ciclu.

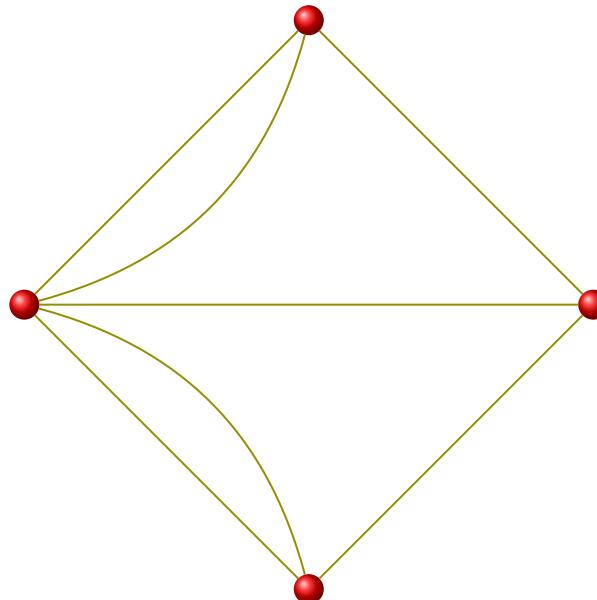
Teorema 83 (Euler) Fie $G = (V, E)$ un graf conex cu proprietatea că $|V| \geq 2$. Atunci G admite un ciclu eulerian dacă și numai dacă

$$\forall v \in V : d_G(v) \in 2\mathbb{N}.$$

Observația 84 Teorema de mai sus are loc și pentru multigrafuri.

6.2 Problema săptămânii

Problema 85 Problema celor 7 poduri de la Königsberg (azi cunoscut sub denumirea de Kaliningrad). Se pune problema existenței unui ciclu eulerian în multigraful de mai jos



6.3 Seminar

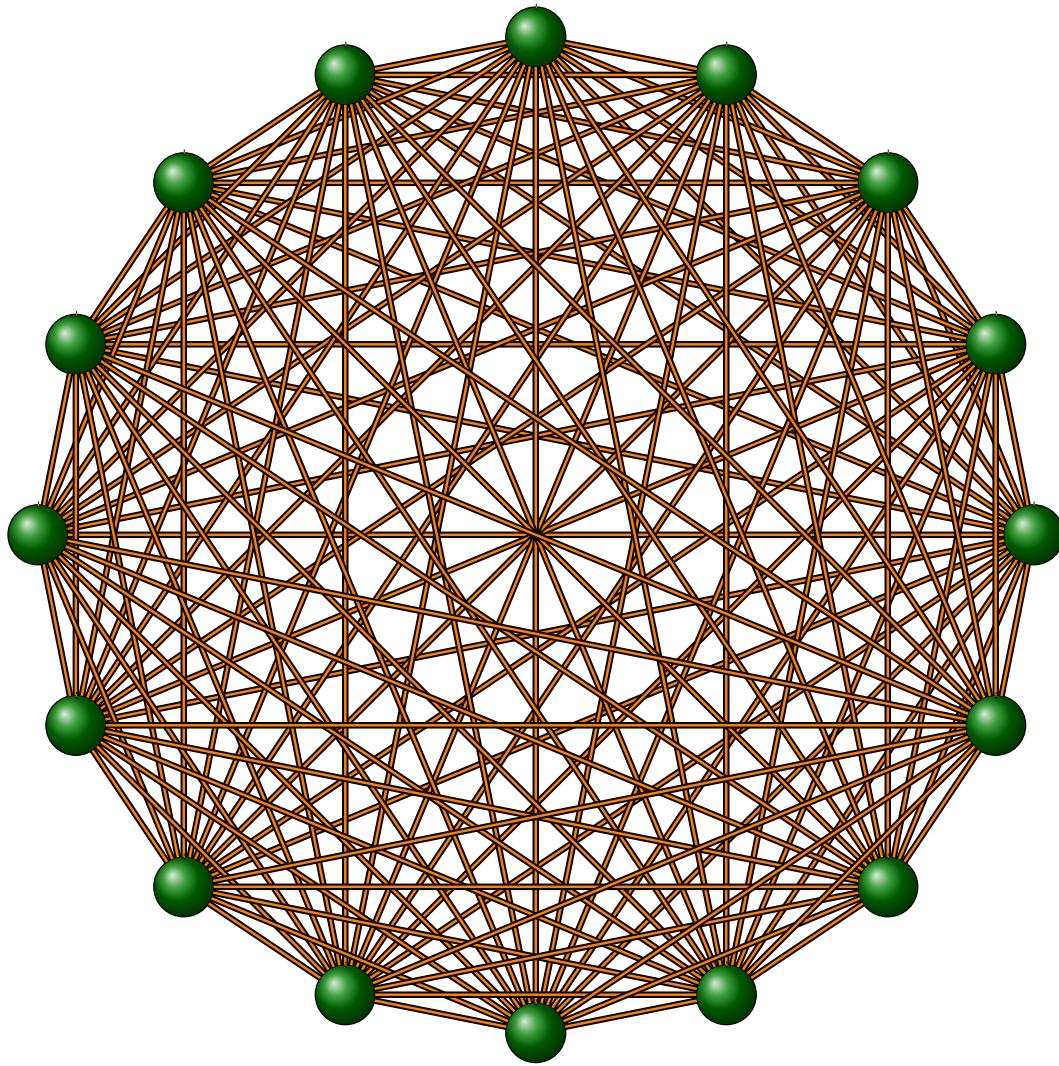
Cursul

7

Clase importante de grafuri

7.1 Teorie	26
7.2 Problema săptămânii	28
7.3 Seminar	28

7.1 Teorie



Definiția 86 Se numește graf **planar** un graf care admite o reprezentare în plan astfel încât muchiile să se intersecteze cel mult în vârfuri.

Observația 87 Orice graf planar împarte planul în suprafețe conexe numite fețe. Numărul muchiilor care delimitizează o față se va numi **gradul** acelei fețe. Vom nota un graf planar G prin

$$G = (V, E, F),$$

unde F este multimea fețelor.

Teorema 88 (Euler) Dacă $G = (V, E, F)$ este un graf planar, atunci

$$\sum_{f \in F} d_G(f) = 2 \operatorname{card} E. \quad (7.1)$$

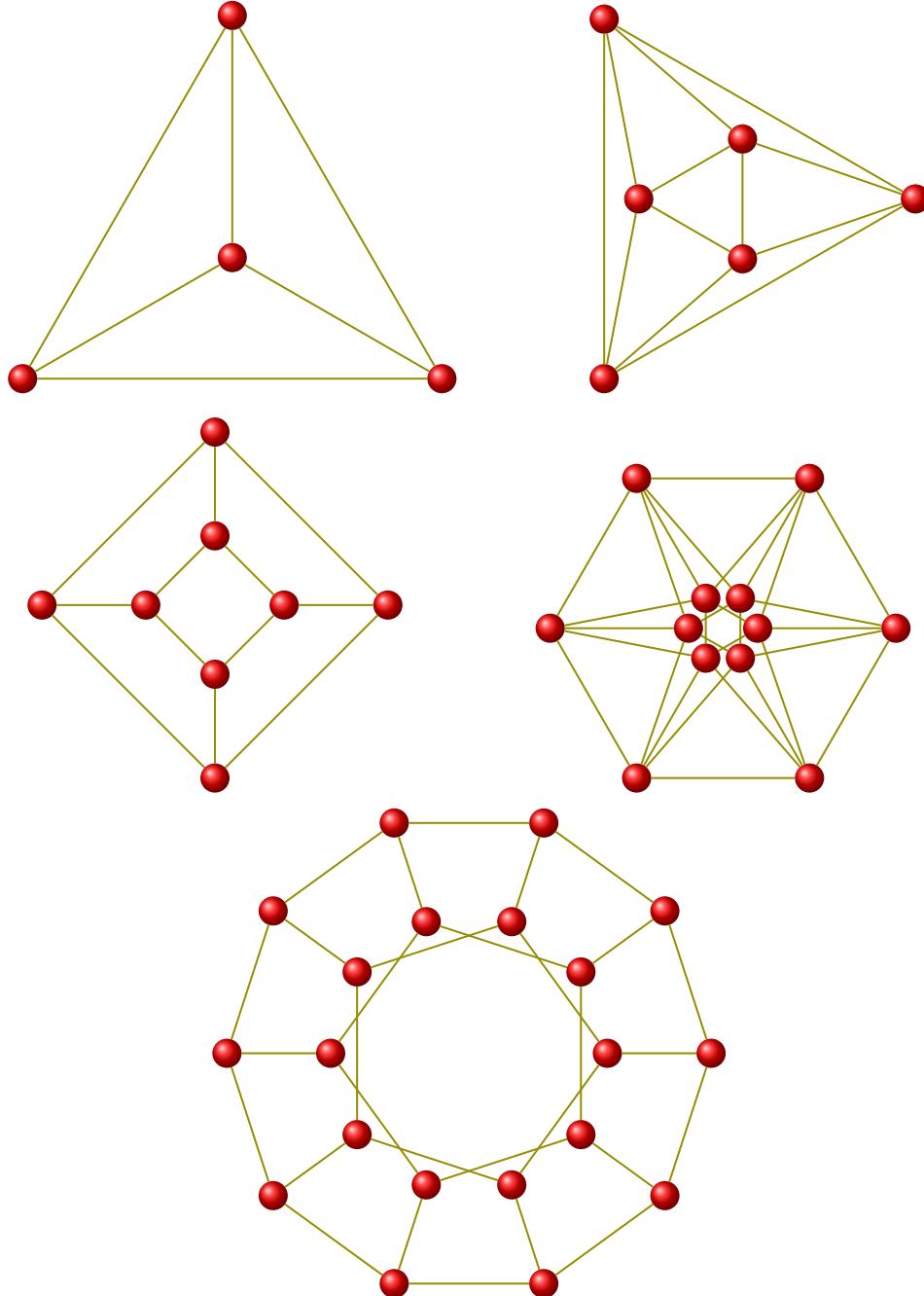
Teorema 89 (formula poliedrală a lui Euler) Dacă $G = (V, E, F)$ este un graf planar conex, atunci

$$|V| - |E| + |F| = 2.$$

Exemplu 90 Grafurile platonice sunt grafuri obținute din muchiile și vârfurile celor cinci poliedre regulate ale lui Platon. Aceste sunt

- *tetraedrul*
- *octoedrul*
- *cubul*
- *icosaedrul*
- *dodecaedrul.*

În figurile de mai jos sunt reprezentate cele 5 grafuri.



Corolar 91 Fie $G = (V, E, F)$ un graf planar conex.

(i) Dacă cu $|V| \geq 3$, atunci

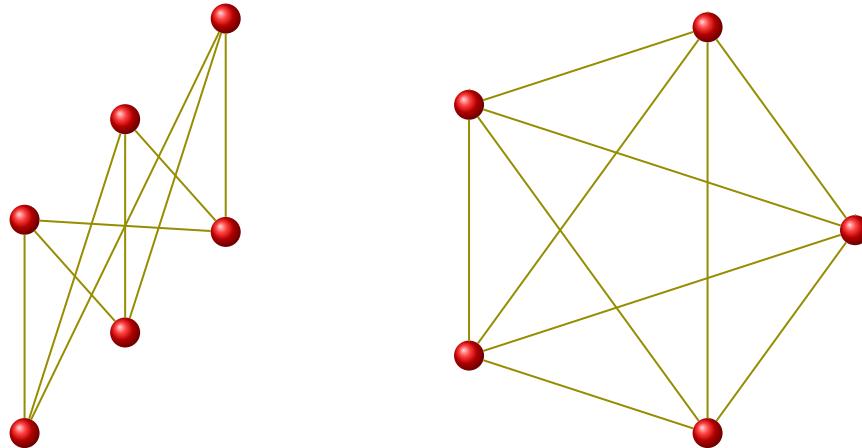
$$|E| \leq 3|V| - 6.$$

(ii) Există un vârf $v \in V$ astfel încât $d_G(v) \leq 5$.

Exemplu 92 Grafurile K_5 și $K_{3,3}$ nu sunt planare.

Definiția 93 Spunem că realizăm o diviziune a unui graf G dacă inserăm vârfuri de grad 2 pe muchiile sale. Graful nou obținut se numește subdiviziune a lui G .

Teorema 94 (Kuratowski) Un graf G este planar dacă și numai dacă nu conține subdiviziuni ale grafurilor K_5 și $K_{3,3}$.



Demonstrație. ... ■

Observația 95 Folosind teorema de mai sus se poate arăta că graful lui Peterson nu este planar.

7.2 Problema săptămânii

7.3 Seminar

Cursul

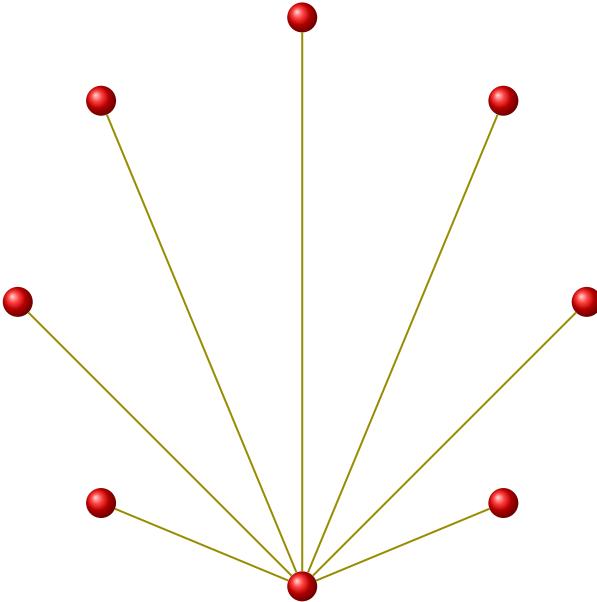
8

Arbore. Arbori parțiali

8.1 Teorie	29
8.2 Problema săptămânii	30
8.3 Seminar	32

8.1 Teorie

Definiția 96 Se numește **arbore** un graf conex aciclic.



Propoziția 97 Fie $G = (V, E)$ un arbore. Atunci G are următoarele proprietăți

- (i) $\forall u, v \in V, u \neq v$, există un unic drum de la u la v .
- (ii) $|E| = |V| - 1$.
- (iii) Există cel puțin 2 vârfuri de grad 1.

Teorema 98 (de caracterizare a arborilor) Fie $G = (V, E)$ un graf. Următoarele afirmații sunt echivalente.

- (i) G este arbore.

-
- (ii) G este conex și $|E| = |V| - 1$.
 (iii) G este aciclic și $|E| = |V| - 1$.
 (iv) G este aciclic și, oricare ar fi $e \in \mathcal{P}_2(V) \setminus E$, graful $G + e$ are exact un ciclu.
 (v) G este conex și, oricare ar fi $e \in E$, graful $G - e$ nu este conex.

Teorema 99 Fie numerele $d_1 \geq d_2 \geq \dots \geq d_n \geq 1$. Atunci există un arbore cu sirul gradelor d_1, d_2, \dots, d_n dacă și numai dacă

$$\sum_{i=1}^n d_i = 2n - 2.$$

Demonstrație. ... ■

Definiția 100 Fie $G = (V, E)$ un graf și $G' = (V, E')$ un graf parțial al lui G . Spunem că G' se numește **arbore parțial** al lui G , dacă G' este arbore.

Teorema 101 Un graf admite arbori parțiali dacă și numai dacă este graf conex.

8.2 Problema săptămânii

Problema 102 Dat un graf conex $G = (V, E)$ construiți un arbore parțial al său.

Pentru a rezolva această problemă, intuitiv, trebuie să parcurgem următori pași

- plecăm de la graful nul $G_0 = (V, \emptyset)$;
- adăugăm succesiv muchii astfel încât ele să nu formeze cicluri cu muchiile deja adăugate;
- ne oprim când numărul de muchii este $|V| - 1$.

Concret, soluția problemei va rezulta aplicând cu multă grijă următorul algoritm format din 5 pași:

1. Dat graful $G = (V, E)$, construim $G_0 = (V, \emptyset)$ și fie $i = 0$;
2. Presupunem că am construit graful $G_i = (V, E_i)$.

Dacă $i = |V| - 1$

atunci G_i este un arbore parțial al lui G ;

STOP;

3. Căutăm o muchie $e \in E \setminus E_i$ astfel încât ea are extremitățile în componente conexe diferite ale lui G_i .

Dacă există o astfel de muchie e

atunci $G_{i+1} := G_i + e$;

altfel trecem la pasul 5;

4. $i := i + 1$

Trecem la pasul 1;

5. Graful G nu este conex, deci nu admite arbori parțiali.

STOP.

1. Dat graful $G = (V, E)$, construim $G_0 = (V, \emptyset)$ și fie $i = 0$;

2. Presupunem că am construit graful $G_i = (V, E_i)$.

Dacă $i = |V| - 1$

atunci G_i este un arbore parțial al lui G ;

STOP;

3. Căutăm o muchie $e \in E \setminus E_i$ astfel încât ea are extremitățile în componente conexe diferite ale lui G_i .

Dacă există o astfel de muchie e

atunci $G_{i+1} := G_i + e$;

altfel trecem la pasul 5;

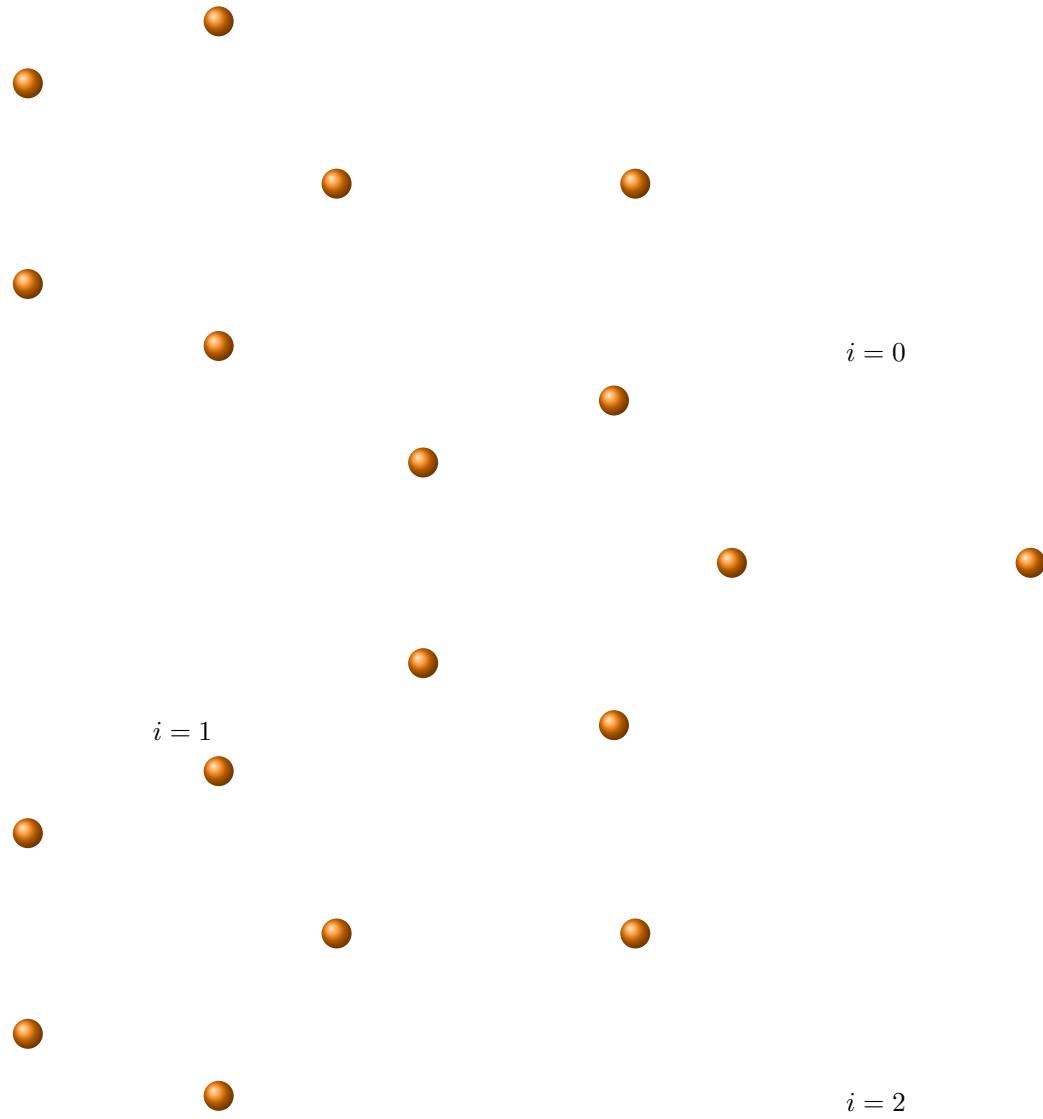
4. $i := i + 1$

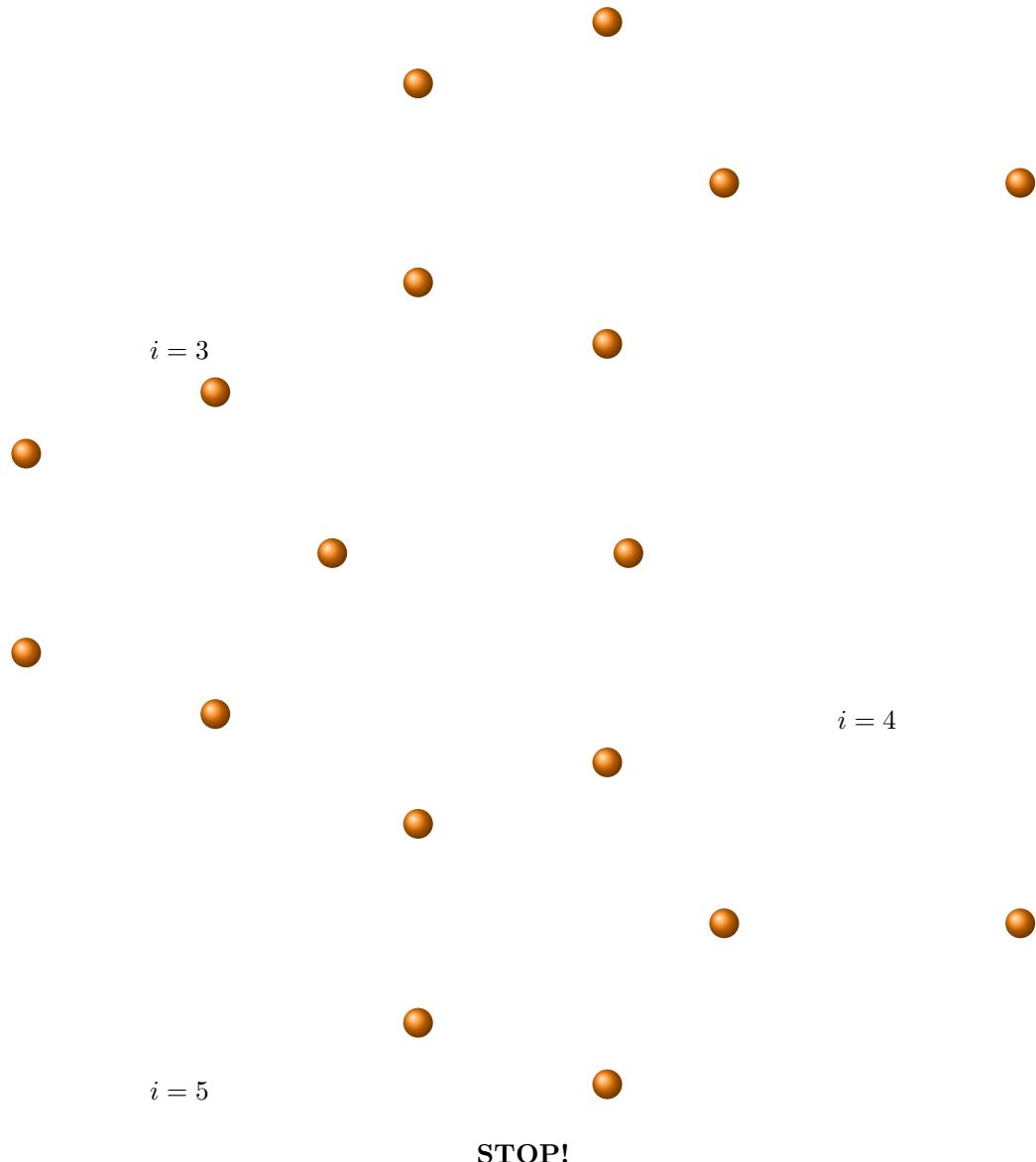
Trecem la pasul 1;

5. Graful G nu este conex, deci nu admite arbori parțiali.

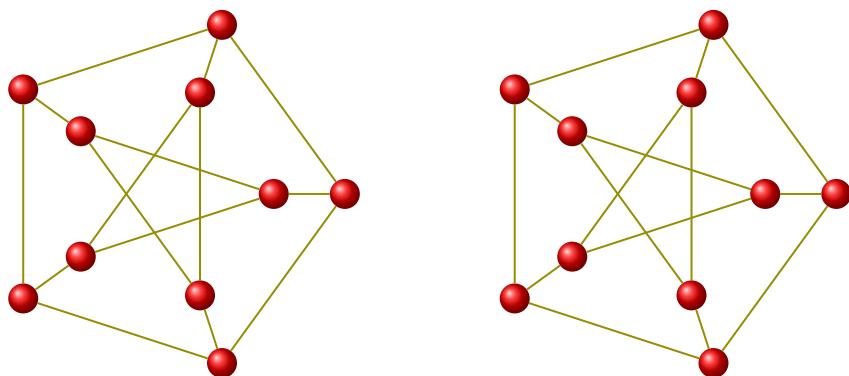
STOP.

Exemplu 103 Găsiți arborele parțial al lui G dat în figura de mai jos





Exemplu 104 După 10 iterații ale algoritmului de mai sus, se arată că graful din dreapta este arborele parțial obținut din graful lui Peterson.



8.3 Seminar

Cursul 9

Arbore parțiali de cost minim. Algoritmul lui Kruskal/Prim

9.1 Teorie	33
9.2 Problema săptămânii	37
9.3 Seminar	37

9.1 Teorie

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

Definiția 105 O funcție $c : E \rightarrow \mathbb{R}$ care asociază fiecărei muchii $e \in E$, numărul real $c(e)$ se numește funcție **cost** asociată grafului G .

Dacă luăm $G' = (V', E')$ un subgraf al lui G , atunci

$$c(G') = \sum_{e \in E'} c(e).$$

Problema 106 Dat un graf $G = (V, E)$ și o funcție cost $c : E \rightarrow \mathbb{R}$, să se determine arborele parțial de cost minim al lui G .

Pentru început, dat un graf trebuie să obținem arborele parțial corespunzător. Acest lucru este posibil utilizând algoritmul următor (dat și în cursul anterior):

1. Dat graful $G = (V, E)$, construim $G_0 = (V, \emptyset)$ și fie $i = 0$;
2. Presupunem că am construit graful $G_i = (V, E_i)$.

Dacă $i = |V| - 1$

atunci STOP – G_i este un arbore parțial al lui G ;

STOP;

3. Căutăm o muchie $e^* \in E \setminus E_i$ astfel încât ea are extremitățile în componente conexe diferite ale lui G_i .

Dacă există o astfel de muchie e^*

atunci $G_{i+1} := G_i + e^*$;

altfel trecem la pasul 5;

4. $i := i + 1$;

Trecem la pasul 1;

5. Graful G nu este conex, deci nu admite arbori parțiali.

STOP.

Teorema 107 Fie $G = (V, E)$ un graf, $c : E \rightarrow \mathbb{R}$ funcția cost asociată, $G' = (V', E')$ un graf parțial aciclic al lui G și V_1 o componentă conexă a lui G' . În aceste condiții, dacă G' este conținut într-un arbore parțial de cost minim al lui G și e^* este o muchie de cost minim ce are exact o extremitate în V_1 , atunci $G' + e^*$ este de asemenea conținut într-un arbore parțial de cost minim în G .

În continuare prezentăm doi algoritmi pentru construirea unui arbore parțial de cost minim.

Algoritmul 108 (Kruskal)

1. Dat graful $G = (V, E)$, $c : E \rightarrow \mathbb{R}$, luăm $G_0 = (V, \emptyset)$ și fie $i = 0$;

2. Presupunem că am construit graful $G_i = (V, E_i)$.

Dacă $i = |V| - 1$

atunci STOP – G_i este un arbore parțial al lui G de cost minim;

3. Căutăm o muchie de cost minim $e^* \in E \setminus E_i$ astfel încât ea are extremitățile în componente conexe diferite ale lui G_i .

Dacă există o astfel de muchie e^*

atunci $G_{i+1} := G_i + e^*$;

altfel trecem la pasul 5;

4. $i := i + 1$;

Trecem la pasul 2;

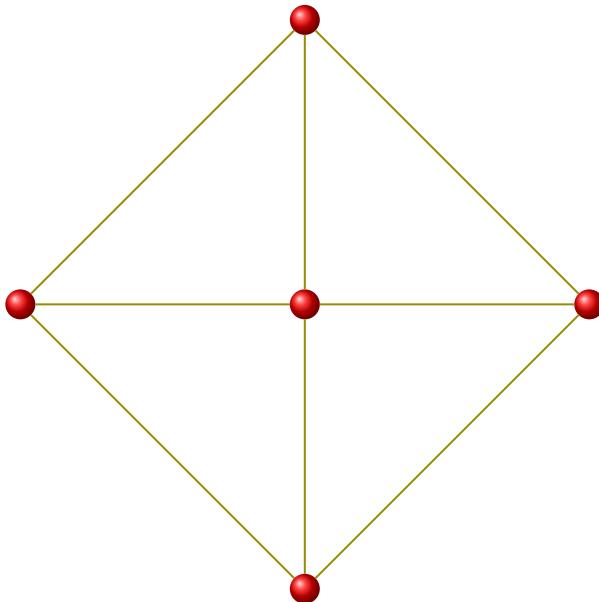
5. Graful G nu este conex, deci nu admite arbori parțiali de cost minim.

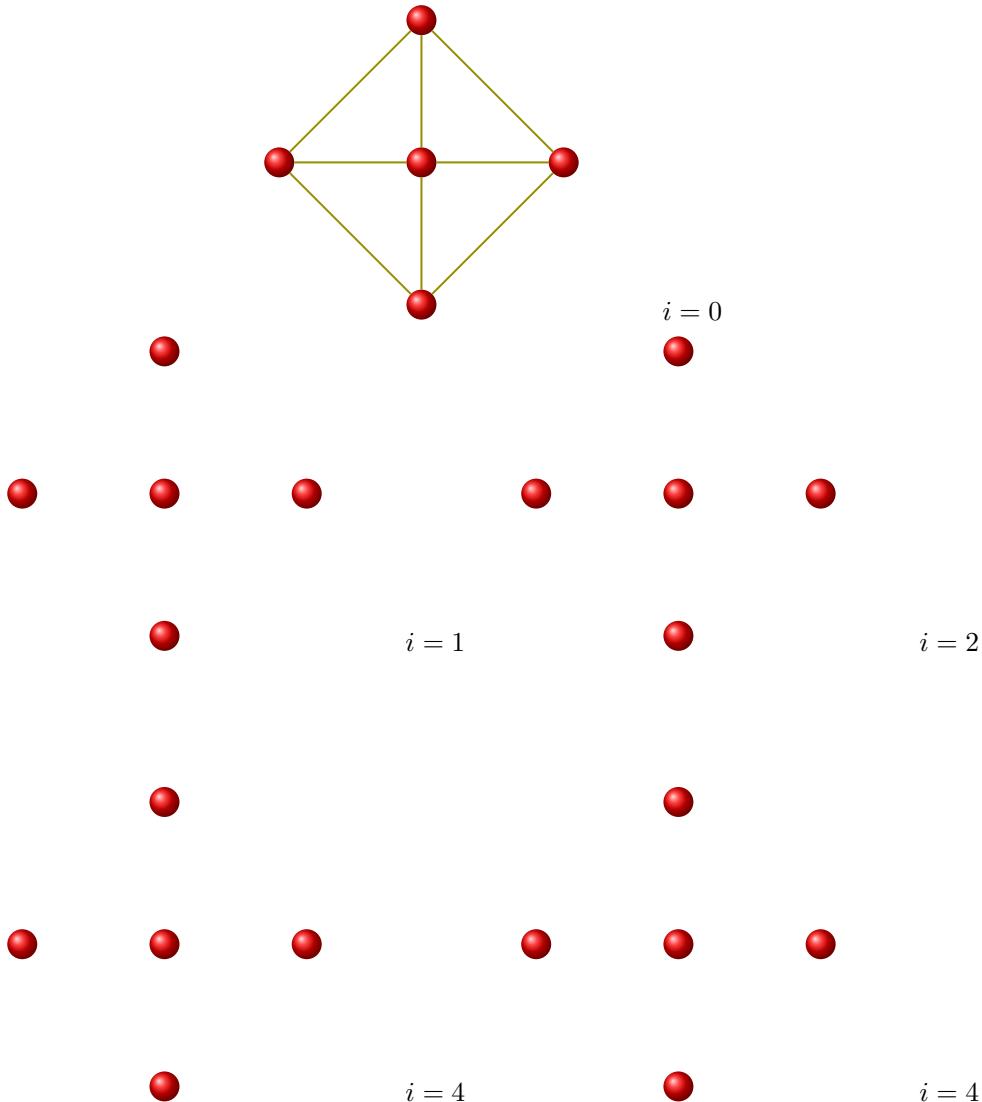
STOP.

Exemplu 109 Aplicăm algoritmul lui Kruskal pentru graful ce are vârfurile $V = \{1, 2, 3, 4, 5\}$ și

$$E = \{\{1, 2\}_2, \{1, 4\}_1, \{1, 5\}_4, \{2, 3\}_1, \{2, 5\}_1, \{3, 5\}_2, \{4, 3\}_3, \{4, 5\}_3\},$$

unde $\{u, v\}_x$ reprezintă o muchie oarecare, iar $x = c(\{u, v\}) \in \mathbb{R}$. Graful dat este reprezentat în figura de mai jos



**Algoritmul 110 (Prim)**

1. Dat graful $G = (V, E)$, $c : E \rightarrow \mathbb{R}$, luăm $v_0 \in V$, $G_0 = (V, \emptyset)$ și $i = 0$;
2. Presupunem că am construit graful $G_i = (V_i, E_i)$.
Dacă $i = |V| - 1$
atunci STOP – G_i este un arbore parțial al lui G de cost minim;
3. Căutăm o muchie de cost minim $e^* \in E \setminus E_i$ astfel încât ea are o extremitate în V_i și cealaltă extremitate în $V \setminus V_i$.
Dacă există o astfel de muchie $e^* = \{u, v\}$, $u \in V_i, v \in V \setminus V_i$
atunci $V_{i+1} := V_i \cup \{v\}$;
 $E_{i+1} = E_i \cup \{e^*\}$;
altfel trecem la pasul 5;
4. $i := i + 1$;
Trecem la pasul 2;
5. Graful G nu este conex, deci nu admite arbori parțiali de cost minim.
STOP.

Exemplu 111 Aplicăm algoritmul lui Prim pentru graful ce are vârfurile $V = \{1, 2, 3, 4, 5\}$ și

$$E = \{\{1, 2\}_2, \{1, 4\}_1, \{1, 5\}_4, \{2, 3\}_1, \{2, 5\}_1, \{3, 5\}_2, \{4, 3\}_3, \{4, 5\}_3\},$$

unde $\{u, v\}_x$ reprezintă o muchie oarecare, iar $x = c(\{u, v\}) \in \mathbb{R}$. Graful dat este reprezentat în figura de mai sus.

Teorema 112 (Tomescu) Într-un graf cu n vârfuri care nu conține un subgraf complet de ordin k , $k \geq 2$, atunci acesta conține cel puțin $m = \{n/(k-1)\}$ vârfuri de grad

$$p \leq \left\lceil \frac{(k-2)n}{k-1} \right\rceil,$$

unde $\{x\}$ este cel mai mic număr întreg $\geq x$.

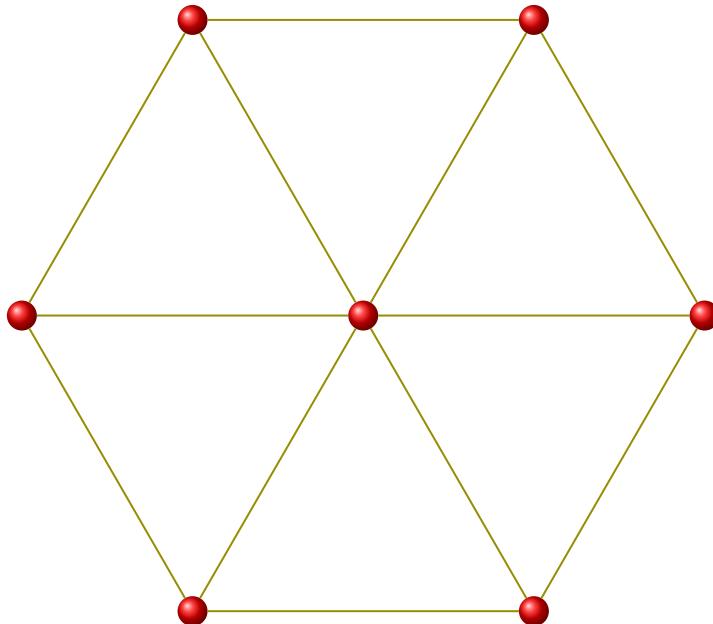
Demonstrație. ...

...

... ■

Exercițiul 113 Într-o mulțime M ce conține 1001 persoane, fiecare submulțime de 11 persoane conține cel puțin două care se cunosc reciproc. Să se arate că există cel puțin 101 persoane astfel încât fiecare din ele cunoaște cel puțin 100 persoane din M .

Exemplu 114 Să se aplique cei doi algoritmi pentru graful de tip roată



9.2 Problema săptămânii

9.3 Seminar

Cursul 10

Probleme de numărare a arborilor

10.1 Teorie	39
10.2 Problema săptămânii	40
10.3 Seminar	40

10.1 Teorie

Teorema 115 (Cayley) Numărul arborilor cu $n \in \mathbb{N}$ vârfuri, v_1, v_2, \dots, v_n este egal cu n^{n-2} .

Demonstrație. Pentru $n = 0$ și $n = 1$ se verifică imediat.

Pentru $n = 3$, Notăm cu $V = \{1, 2, 3, \dots, n\}$ și cu $\mathcal{A}(n)$ multimea arborilor cu vârfuri în V . Definim funcția $f : \mathcal{A}(n) \rightarrow V^{n-2}$, unde

$$V^{n-2} = \{(u_1, u_2, \dots, u_{n-2}) \mid u_i \in V, i \in \overline{1, n}\},$$

definită prin

$$G = (V, E) \mapsto f(G) = (u_1, u_2, \dots, u_{n-2})$$

definit astfel

$$\begin{cases} v_1 = \min \{v \in V \mid d_G(v) = 1\} \\ u_1 \in N(v_1) \text{ (unic)} \end{cases} \Rightarrow G_1 = G - v_1,$$

$$\begin{cases} v_2 = \min \{v \in V \setminus \{v_1\} \mid d_{G_1}(v) = 1\} \\ u_2 \in N(v_2) \text{ (unic)} \end{cases} \Rightarrow G_2 = G_1 - v_2,$$

...

$$\begin{cases} v_k = \min \{v \in V \setminus \{v_1, \dots, v_{k-1}\} \mid d_{G_{k-1}}(v) = 1\} \\ u_k \in N(v_k) \text{ (unic)} \end{cases} \Rightarrow G_k = G_{k-1} - v_k.$$

Obținem astfel un sir $(u_1, u_2, \dots, u_{n-2})$, numit cod Prüfer al lui G . Arătând că funcția f este bijectivă se obține concluzia dorită. ■

Observația 116 Graful $G_k = (V_k, E_k)$ este alcătuit din

$$V_k = V \setminus \{v_1, \dots, v_k\}$$

și

$$E_k = \{\{v_i, u_i\} \mid i \in \overline{k+1, n-1}\}.$$

Observația 117 Vârfurile de grad 1 din G_k sunt $\{v_{k+1}, v_{k+2}, \dots, v_{n-1}, n\} \setminus \{u_{k+1}, \dots, u_{n-2}\}$. Prin urmare putem defini

$$v_i = \min \{v \in V \mid v \neq v_k, \forall k \geq i \text{ și } v \notin \{v_1, \dots, v_{i-1}\}\}.$$

Exercițiul 118 Să se găsească graful G al cărui cod Prüfer este

- (i) $f(G) = (2, 1, 1)$
- (ii) $f(G) = (1, 1, 1, 1, 6, 5)$.

Exercițiul 119 Să se determine o metodă de a determina codul Prüfer, dacă avem dat graful G .

10.2 Problema săptămânii

10.3 Seminar

Cursul 11

Algoritmi de căutare

11.1 Teorie	41
11.2 Problema săptămânii	41
11.3 Seminar	41

11.1 Teorie

11.2 Problema săptămânii

11.3 Seminar

Cursul 12

Probleme de drum minim (maxim)

12.1 Teorie	43
12.2 Problema săptămânii	46
12.3 Seminar	46

12.1 Teorie

Fie $G = (V, E)$ un digraf, $c : E \rightarrow \mathbb{R}$ funcția cost asociată lui G . Vom face următoarele notații:

- $\forall e = (u, v) \in E$, $c(e) = c(u, v)$ se numește ponderea arcului (u, v) .
- $\forall \mu = (u_1, \dots, u_k)$, $c(\mu) = c(u_1, u_2) + \dots + c(u_{k-1}, u_k)$ se numește ponderea drumului μ .
- $\mu = (u)$, atunci $c(\mu) = 0$.
- $\forall u, v \in V$, $(u, v) \notin E \implies c(u, v) = \infty$.
- $\forall u, v \in V$, $\mathcal{D}(u, v) = \{\mu \mid \mu \text{ drum de la } u \text{ la } v\}$.
- $\forall u, v \in V$, $d(u, v) = c(\mu^*)$, $\mu^* = \min \{c(\mu) \mid \mu \in \mathcal{D}(u, v)\}$. Prin convenție

$$d(u, u) = 0 \quad \text{și} \quad d(u, v) = \infty, \text{ dacă } u \neq v \text{ și } \mathcal{D}(u, v) = \emptyset.$$

Vom studia în cele ce urmează următoarea problemă

Problema 120 (cost min/max) Fie $G = (V, E)$ un graf și $c : E \rightarrow \mathbb{R}$ funcția cost asociată.

Oricare ar fi $u, v \in V$, să se determine $\mu^* \in \mathcal{D}(u, v)$ astfel încât

$$c(\mu^*) = \min \{c(\mu) \mid \mu \in \mathcal{D}(u, v)\}$$

sau

$$c(\mu^*) = \max \{c(\mu) \mid \mu \in \mathcal{D}(u, v)\}.$$

Date două drumuri $\mu_1 = (x_1, \dots, x_k)$ și $\mu_2 = (y_1, \dots, y_h)$, atunci definim

$$\mu_1 \circ \mu_2 = (x_1, \dots, x_{k-1}, x_k = y_1, y_2, \dots, y_h)$$

și notăm prin $\mu(x_i, x_k) = (x_i, \dots, x_k)$, $\forall i \in \overline{1, k-1}$.

Observația 121 Problema de cost min / max se rezolvă ușor atunci când digraful G nu conține circuite de pondere negativă.

Teorema 122 Fie $G = (V, E)$ un digraf, $c : E \rightarrow \mathbb{R}$ funcția cost și presupunem că G nu conține circuite de pondere negativă. Atunci

$$\forall u, v, w \in V : d(u, v) \leq d(u, w) + d(w, v). \quad (12.1)$$

Teorema 123 Fie $G = (V, E)$ un digraf, $c : E \rightarrow \mathbb{R}$ funcția cost și presupunem că G nu conține circuite de pondere negativă. Atunci oricare ar fi $u, v \in V$ și oricare ar fi μ drum de pondere minimă de la u la v , oricare ar fi w vârf al lui μ , $\mu = (u, w)$ și $\mu(w, v)$ sunt drumuri de pondere minimă de la u la w , respectiv de la w la v .

Teorema 124 Fie $G = (V, E)$ un digraf, $c : E \rightarrow \mathbb{R}$ funcția cost și presupunem că G nu conține circuite de pondere negativă. Atunci

$$\forall v, u \in V : u \neq v \Rightarrow d(u, v) = \min \{d(u, w) + c(w, v) \mid w \in V \setminus \{v\}\}.$$

Vom studia trei cazuri pentru problema de cost minim:

- G nu are circuite;
- $c : R \rightarrow \mathbb{R}_+$;
- G nu conține circuite de pondere negativă.

CAZUL II

Definiția 125 Fie $G = (V, E)$ un digraf cu $|V| = n$. Numim **numerotare aciclică** a vârfurilor lui G o funcție bijectivă

$$\varphi : V \rightarrow \{1, 2, \dots, n\} : \forall (u, v) \in E, \quad \varphi(u) < \varphi(v).$$

Teorema 126 Un digraf G admite numerotare aciclică dacă și numai dacă G nu are circuite.

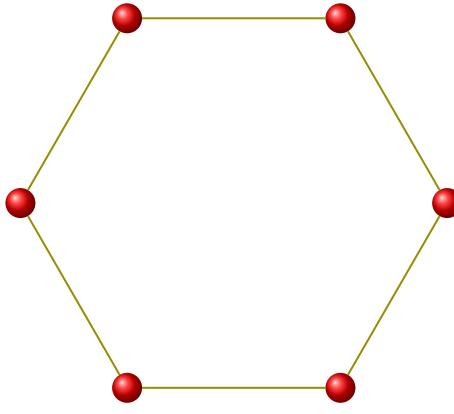
Observația 127 Pentru determinarea unei numerotări aciclice a unui digraf fără circuite se procedează după cum urmează. Fie

$$V_1 = \{v \in V \mid d_G^-(v) = 0 \text{ sau } v \text{ nu are antecedenți}\}.$$

Numerotăm cu $1, 2, \dots, |V_1|$ elementele mulțimii V_1 . Apoi alegem $G_1 = G - V_1$ și definim mulțimea

$$V_2 = \{v \in V \setminus V_1 \mid d_{G_1}^-(v) = 0\}.$$

Numerotăm cu $|V_1| + 1, \dots, |V_1| + |V_2|$ elementele mulțimii V_2 . Continuăm acest procedeu până epuizăm elementele mulțimii V . În figura de mai jos este un digraf numerotat conform cu cele stabilite mai sus.



Presupunem că G nu are circuite, $c : E \rightarrow \mathbb{R}$, $V = \{1, 2, \dots, n\}$ și

$$\forall (i, j) \in E : i < j.$$

Fixăm $s \in V$. Oricare ar fi $i \in V$, $d(s, i) := m(i)$ și notăm cu $p(i)$ vârful ce precede vârful i pe un drum de pondere minimă de la s la i . Avem că

$$m(s) = 0, \quad \forall i < s : m(i) = \infty.$$

Oricare ar fi $i \leq h$, presupunem cunoscute valorile $m(i)$. Astfel avem

$$\begin{aligned} m(h+1) &= \min \{m(i) + c(i, h+1) \mid s \leq i \leq h\} \\ &= m(i^*) + c(i^*, h+1). \end{aligned}$$

Deducem că $p(h+1) = i^*$.

CAZUL III

Fie $G = (V, E)$ un digraf, $c : E \rightarrow \mathbb{R}_+$.

Fixăm $s \in V, S \subset V$ astfel încât $s \in S$.

Definiția 128 Se numește (s, S) – marcare a vârfurilor lui G o funcție

$$m : V \rightarrow \mathbb{R}, \quad m(v) := \begin{cases} d(s, v) & , \quad v \in S \\ \min \{m(u) + c(u, v) \mid u \in S\} & , \quad v \in V \setminus S \end{cases}.$$

În algoritmul lui Dijkstra se pornește de la $S = \{s\}$ și se construiește un sir de (s, S) – marcări ale vârfurilor lui G . În final, vom lua $S = V$, situație în care

$$\forall v \in V : m(v) = d(s, v).$$

Teorema 129 În ipotezele anterioare, dacă $v_0 \in V \setminus S$ astfel încât

$$m(v_0) = \min \{m(v) \in V \setminus S\},$$

atunci $m' : V \rightarrow \mathbb{R}$ definită prin

$$m'(v) := \begin{cases} m(v) & , \quad v \in S \cup \{v_0\} \\ \min \{m(v), m(v_0) + c(v_0, v)\} & , \quad v \in V \setminus (S \cup \{v_0\}) \end{cases}$$

este o $(s, S \cup \{v_0\})$ – marcare a vârfurilor lui G .

CAZUL IIII

Fie $G = (V, E)$ un digraf și $c : E \rightarrow \mathbb{R}$ funcția cost asociată. Presupunem că G nu conține circuite de pondere negativă.

Fie $V = \{1, 2, \dots, n\}$. Oricare ar fi $i, j \in V$ definim

$$\mathcal{D}(i, j) := \{\mu \mid \mu \text{ drum de la } i \text{ la } j\}$$

și

$$d(i, j) := \min \{c(\mu) \mid \mu \in \mathcal{D}(i, j)\}.$$

Pentru $\mu \in \mathcal{D}(i, j)$, notăm cu $V(\mu)$ mulțimea vârfurilor lui μ , oricare ar fi $k \in \overline{1, n}$ definim

$$\mathcal{D}_k(i, j) := \{\mu \in \mathcal{D}(i, j) \mid V(\mu) \setminus \{i, j\} \subset \{1, 2, 3, \dots, k\}\},$$

iar $\mathcal{D}_n(i, j) = \mathcal{D}(i, j)$, și

$$d_k(i, j) := \min \{c(\mu) \mid \mu \in \mathcal{D}_k(i, j)\},$$

iar $d_n(i, j) = d(i, j)$.

Algoritmul lui Floyd-Warshall de determinare de drumului de cost minim, se bazează pe găsirea unei relații de recurență cu ajutorul căreia putem calcula

$$d_k(i, j), \quad k \in \overline{1, n}.$$

Teorema 130 În ipoteze anterioare, oricare ar fi $i, j \in \overline{1, n}$, avem

$$\begin{cases} d_0(i, j) = c(i, j) \\ \forall k \geq 0 : d_{k+1}(i, j) = \min \{d_k(i, j), d_k(i, k+1) + d_k(k+1, j)\} \end{cases} . \quad (12.2)$$

12.2 Problema săptămânii

12.3 Seminar

Cursul 13

Algoritmii lui Dantzig și Ford, Dijkstra, Floyd și Warshall

13.1 Teorie	47
13.2 Problema săptămânii	49
13.3 Seminar	49

13.1 Teorie

CAZUL II

Fixăm $s \in V$. Oricare ar fi $i \in V$, $d(s, i) := m(i)$ și notăm cu $p(i)$ vârful ce precede vârful i pe un drum de pondere minimă de la s la i . Avem că

$$m(s) = 0, \quad \forall i < s : m(i) = \infty.$$

Algoritm 131 (Dantzig și Ford) *Acet algoritm returnează drumul de cost minim într-un graf ce nu conține circuite.*

1. $m(s) := 0, \quad \forall i < s : m(i) = \infty; \quad \forall i \leq s : p(i) := s; \quad h = s;$

2. Presupunem cunoscute $\forall i \leq h, m(i)$ și $p(i)$.

Definim $m(h+1) = \min \{m(i) + c(i, h+1) \mid s \leq i \leq h\} = m(i^*) + c(i^*, h+1);$

$p(h+1) = i^*;$

3. $h := h + 1;$

4. Dacă $h < n$

atunci trecem la Pasul 2;

altfel trecem la Pasul 5;

5. Fie $t \in \{1, 2, 3, \dots, n\}$.

Dacă $m(t) = \infty$

atunci STOP (nu există drum de la s la t);

altfel $m(t)$ este ponderea unui drum de pondere minimă de la s la t și un astfel de drum este

$$\mu = (i_r, i_{r-1}, \dots, i_1, t), \quad \text{unde } \begin{cases} i_1 = p(t) \\ i_2 = p(i_1) \\ \dots \\ i_r = p(i_{r-1}) = s \end{cases}.$$

CAZUL III

Oricare ar fi $v \in V$, notăm cu $m(v)$ ponderea minimă de la s la v și cu $p(v)$ vârful care îl precede pe v pe un drum de pondere minimă de la s la v .

Algoritm 132 (Dijkstra) *Dăm acum un algoritm pentru determinarea drumului de cost minim atunci când $c : E \rightarrow \mathbb{R}_+$.*

1. $m(s) := 0, \quad \forall v \in V \setminus \{s\} : m(v) = c(s, v);$

$\forall v \in V : p(v) := s;$

$S = \{s\};$

2. Se determină $v_0 \in V \setminus S$ astfel încât

$$m(v_0) = \min \{m(v) \mid v \in V \setminus S\}.$$

3. $S := S \cup \{v_0\};$

Dacă $S = V$

atunci trecem la Pasul 5;

altfel trecem la Pasul 4;

4. Oricare ar fi $v \in V \setminus S$, execută

Dacă $m(v) > m(v_0) + c(v_0, v)$

atunci $m(v) := m(v_0) + c(v_0, v);$

$p(v) := v_0;$

5. Oricare ar fi $v \in V$, $m(v)$ reprezintă ponderea minimă a unui drum de la s la v , iar $p(v)$ reprezintă vârful ce precede v pe un astfel de drum.

CAZUL III

Algoritmul 133 (Floyd și Warshall) *Dăm acum un algoritm pentru determinarea drumului de cost minim atunci când G nu conține circuite de pondere negativă.*

1. *Oricare ar fi $i, j \in \overline{1, n}$*

$$d_0(i, j) := c(i, j);$$

$$p_0(i, j) := i;$$

$$k = 0;$$

2. *Oricare ar fi $i, j \in \overline{1, n}$, execută*

$$\text{Dacă } d_k(i, j) \geq d_k(i, k+1) + d_k(k+1, j)$$

$$\text{atunci } d_{k+1}(i, j) = d_k(i, k+1) + d_k(k+1, j);$$

$$p_{k+1}(i, j) = p_k(k+1, j);$$

3. $k := k + 1$;

Dacă $k < n$

atunci trecem la Pasul 2;

altfel trecem la Pasul 4;

4. *Oricare ar fi $i, j \in \overline{1, n}$, $d_n(i, j)$ reprezintă ponderea minimă a unui drum de la i la j , iar un astfel de drum este*

$$\mu = (i, i_r, i_{r-1}, \dots, i_2, i_1, j), \quad \text{unde} \quad \begin{cases} p_n(i, j) = i_1 \\ p_n(i, i_1) = i_2 \\ \dots \\ p_n(i, i_r) = i \end{cases}.$$

13.2 Problema săptămânii

13.3 Seminar

Cursul 14

Metoda drumului critic

14.1 Teorie	51
14.2 Problema săptămânii	53
14.3 Seminar	53

14.1 Teorie

Fie \mathbb{P} un proiect, \mathbb{X} mulțimea operațiilor \mathbb{P} . Oricare ar fi $x \in \mathbb{X}$, notăm cu $\mathbb{P}(x)$ mulțimea operațiilor ce trebuie efectuate anterior lui x , numită și listă de precedență. Fie $x \in \mathbb{X}$, avem că

- Dacă $\mathbb{P}(x) = \emptyset$, atunci x se numește **operație inițială**.
- Dacă $\exists y \in \mathbb{X}, x \in \mathbb{P}(y)$, atunci x se numește **operație finală**.

Definiția 134 Un digraf $G = (V, E)$ este o reprezentare a listelor de precedență $\mathbb{P}(x), x \in \mathbb{X}$, dacă există o funcție injectivă $f : X \rightarrow E$ astfel încât, $\forall x, y \in \mathbb{X}, x \in \mathbb{P}(y)$ dacă și numai dacă există în G un drum de la extremitatea finală a lui $f(x)$ la extremitatea inițială a lui $f(y)$.

Exemplu 135 De exemplu, dacă $\mathbb{X} = \{a, b, c, d\}$, atunci avem

$$\begin{array}{c|ccccc} x & a & b & c & d \\ \hline \mathbb{P}(x) & \emptyset & \emptyset & a & ab \end{array}.$$

Observația 136 Presupunem că $x \in \mathbb{P}(y)$ și există $z \in \mathbb{P}(y)$ astfel încât $x \in \mathbb{P}(z)$. Atunci x se poate elimina din $\mathbb{P}(y)$, căci efectuarea operației z va duce implicit și la efectuarea operației x . Făcând toate liminările posibile se obțin liste de precedență ireductibile.

De exemplu, dacă luăm $\mathbb{X} = \{a, b, c, d, e\}$ cu $b \in \mathbb{P}(d)$ și $\exists c \in \mathbb{P}(d)$ astfel ca $b \in \mathbb{P}(c)$. Atunci putem scrie

$$\begin{array}{c|ccccc} x & a & b & c & d & e \\ \hline \mathbb{P}(x) & \emptyset & a & b & bc & dc \\ \mathbb{P}'(x) & \emptyset & a & b & c & c \end{array},$$

unde ultima linie reprezintă o listă ireductibilă.

Observația 137 De acum înainte presupunem că lucrăm doar cu liste de precedență ireductibile.

Oricare ar fi $x \in \mathbb{X}$ definim

$$Q(x) := \begin{cases} \bigcap_{x \in \mathbb{P}(y)} \mathbb{P}(x) & , \quad x \text{ nu este operație finală} \\ \mathbb{F} & , \quad x \text{ este operație finală} \end{cases},$$

unde \mathbb{F} este mulțimea tuturor operațiilor finale. Construim digraful $G = (V, E)$ asociat proiectului \mathbb{P} după cum urmează:

- $\forall x, y \in \mathbb{X}, x \neq y : \mathbb{P}(x) \neq \mathbb{P}(y)$ sau $Q(x) \neq Q(y)$.

Atunci construim $G = (V, E)$ luând

$$V = \{\mathbb{P}(x) \mid x \in \mathbb{X}\} \cup \{Q(x) \mid x \in \mathbb{X}\}$$

și

$$E = \{(\mathbb{P}(x), Q(x)) \mid x \in \mathbb{X}\} \cup \{(Q(x), \mathbb{P}(y)) \mid x \in \mathbb{P}(y) \neq Q(x)\}.$$

Prin urmare, G este o reprezentare a listelor de precedență $\mathbb{P}(x), x \in \mathbb{X}$. Aplicația $f : \mathbb{P} \rightarrow E, f(x) = (\mathbb{P}(x), Q(x))$ este injectivă.

Condiția $x \in \mathbb{P}(y)$ este echivalentă cu

$$Q(x) = \mathbb{P}(y) \Rightarrow \exists \mu = (Q(x))$$

sau

$$Q(x) \neq \mathbb{P}(y) \Rightarrow \exists \mu = (Q(x), \mathbb{P}(y)).$$

- $\exists x \in \mathbb{X}$ astfel încât

$$\underbrace{\text{card} \{y \in \mathbb{X} \mid \mathbb{P}(x) = \mathbb{P}(y) \text{ și } Q(x) = Q(y)\}}_{A_x} \geq 2.$$

Presupunem că $A_x = \{x, y_1, y_2, \dots, y_{p_x}\}, p_x \geq 1$. Atunci construim $G = (V, E)$ luând

$$V = \{\mathbb{P}(x) \mid x \in \mathbb{X}\} \cup \{Q(x) \mid x \in \mathbb{X}\} \cup \{Q_1, Q_2, \dots, Q_{p_x} \mid x \in \mathbb{X}\}$$

și

$$\begin{aligned} E &= \{(\mathbb{P}(x), Q(x)) \mid x \in \mathbb{X}\} \cup \{(\mathbb{P}(x), Q_i) \mid x \in \mathbb{X}, i \in \overline{1, p_x}\} \\ &\quad \cup \{(Q(x), \mathbb{P}(y)) \mid x \in \mathbb{P}(y) \neq Q(x)\} \cup \{(Q_i, Q(x)) \mid x \in \mathbb{X}, i \in \overline{1, p_x}\} \\ &= E_1 \cup E_2, \end{aligned}$$

unde E_1 conține primele două mulțimi, iar E_2 ultimele două. Prin urmare, G este o reprezentare a listelor de precedență $\mathbb{P}(x), x \in \mathbb{X}$. Aplicația $f : \mathbb{P} \rightarrow E, f(x) = (\mathbb{P}(x), Q_i), i \in \overline{1, p_x}$, este injectivă.

Condiția $x \in \mathbb{P}(y)$ este echivalentă cu

$$Q(x) = \mathbb{P}(y) \Rightarrow \exists \mu = (Q(x))$$

sau

$$Q(x) \neq \mathbb{P}(y) \Rightarrow \exists \mu = (Q(x), \mathbb{P}(y)).$$

Digraful $G = (V, E)$ astfel construit se numește digraful asociat proiectului \mathbb{P} . Oricare ar fi $x \in \mathbb{X}$, presupunem cunoscut timpul $t(x)$ necesar efectuării operației x și definim funcția cost $c : E \rightarrow \mathbb{R}$ prin

$$c(u, v) := \begin{cases} t(x) & , \quad (u, v) \text{ corespunde operației } x \\ 0 & , \quad (u, v) \text{ nu corespunde operației } x \end{cases}.$$

Notez cu \mathbb{T} timpul minim necesar efectuării operațiilor proiectului.

Teorema 138 În condițiile date mai sus, avem că

$$\mathbb{T} = \max_{\mu} c(\mu),$$

unde μ parcurge mulțimea drumurilor de la vârful inițial \oslash la vârfurile finale din \mathbb{F} .

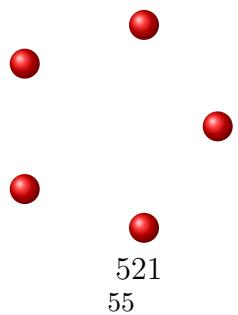
14.2 Problema săptămânii

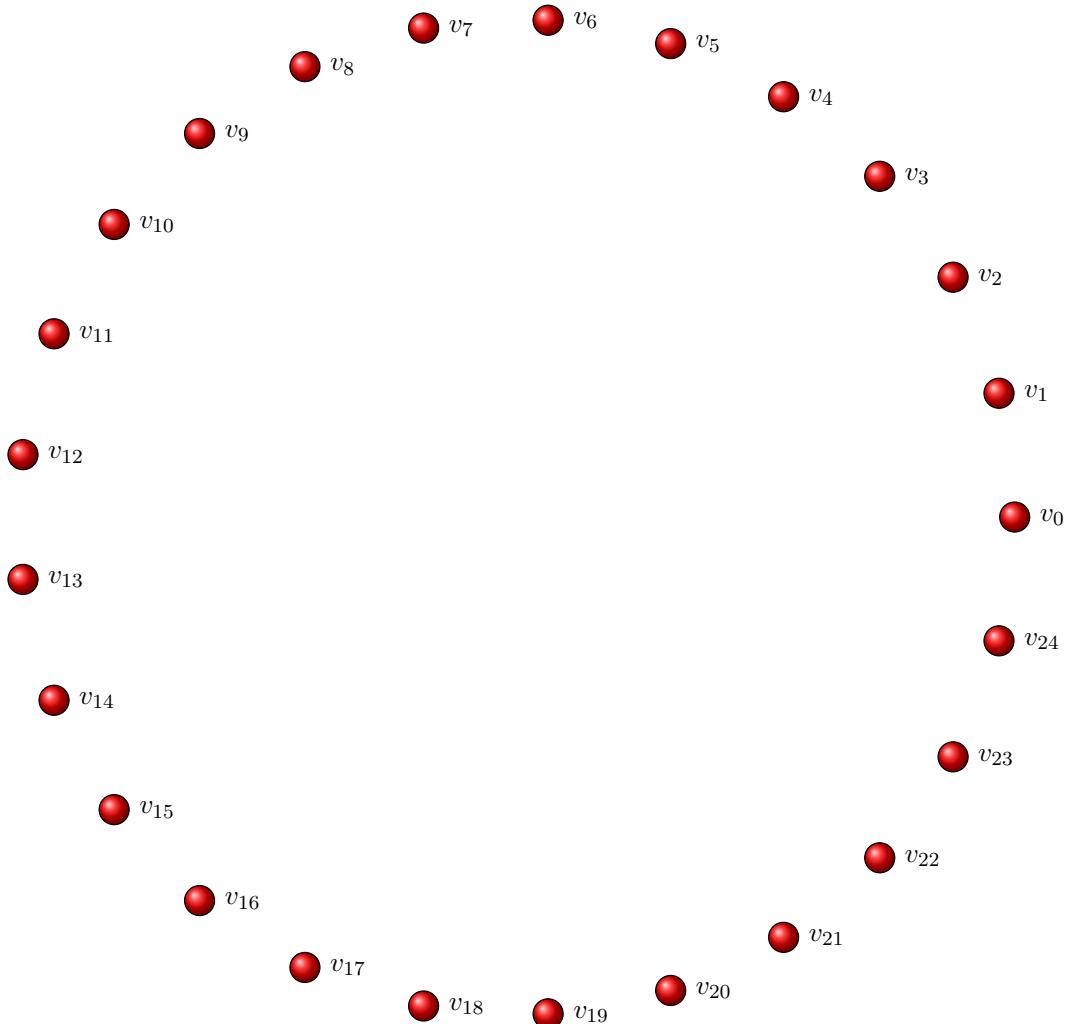
14.3 Seminar

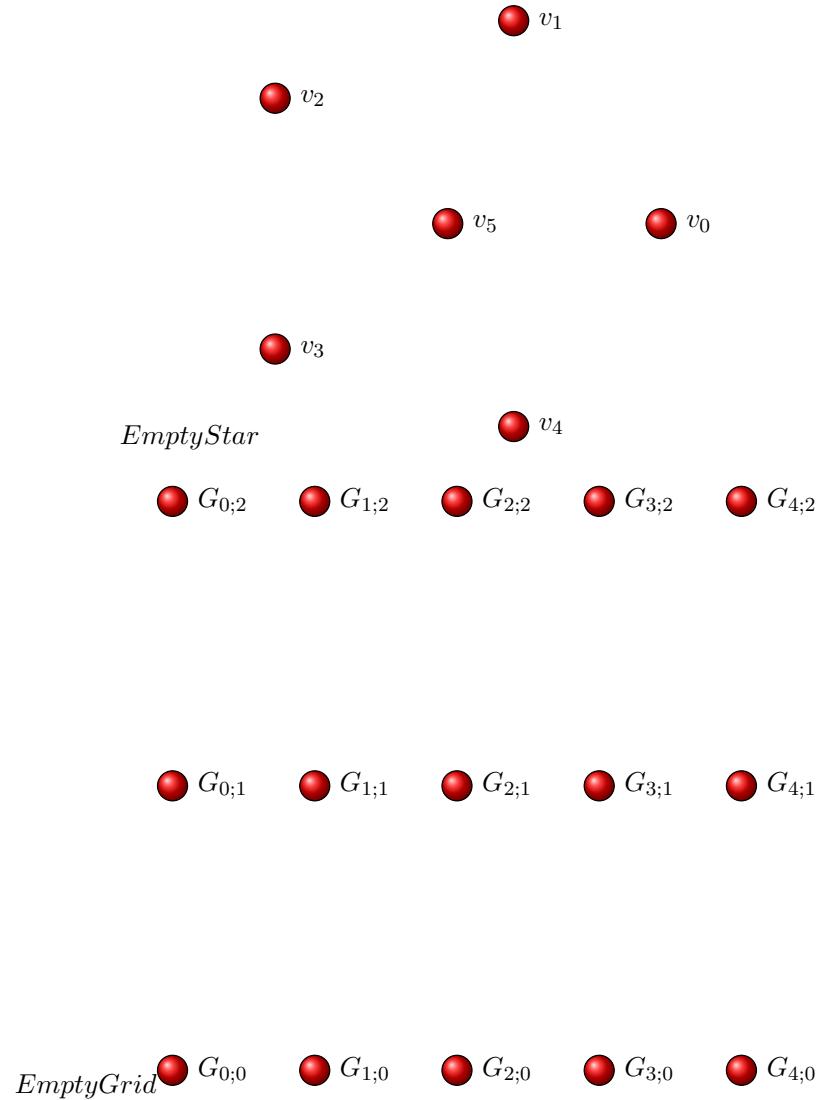
Cursul 15

Diagrame

Diagrame.....







 b_0

 b_1

 b_2

 b_3

 b_4

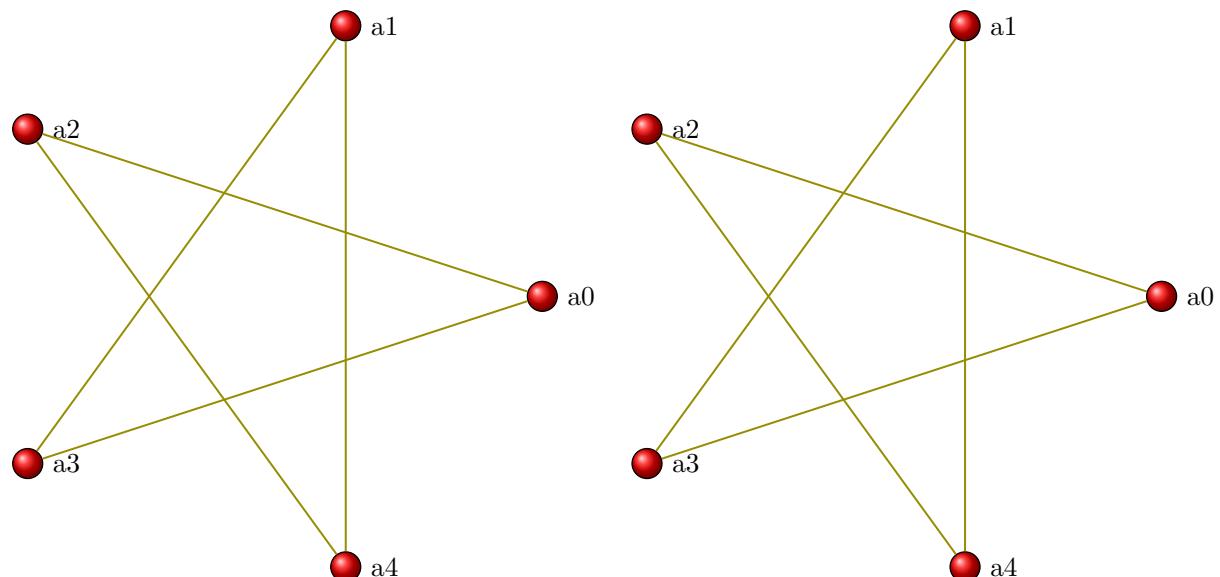
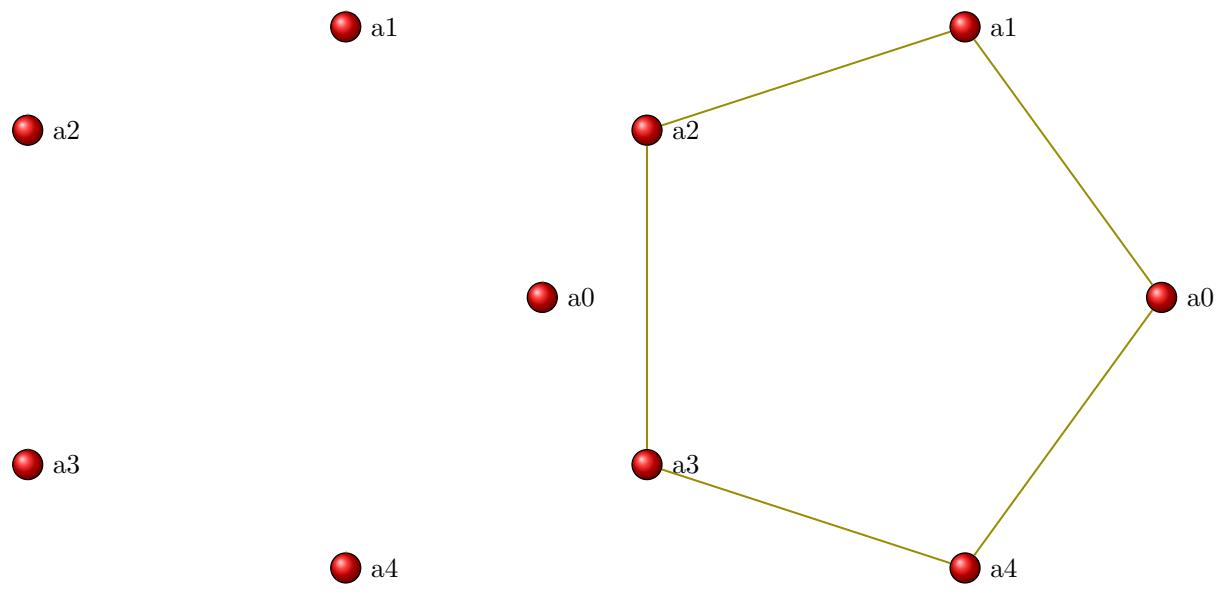
EmptyLader  a_0

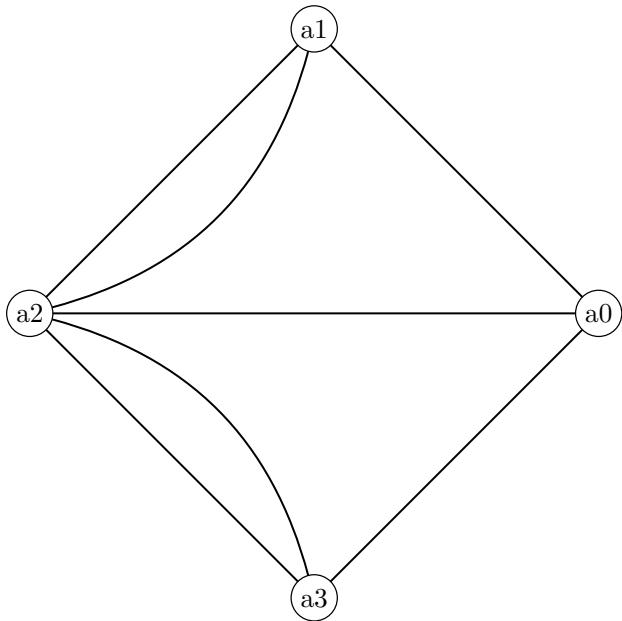
 a_1

 a_2

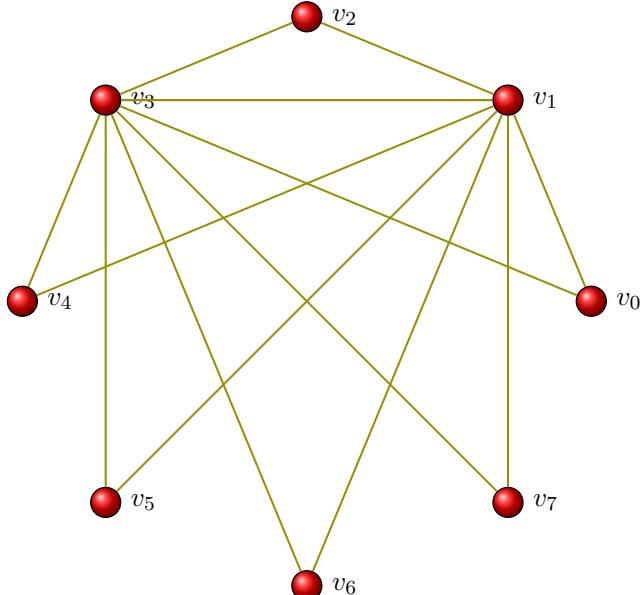
 a_3

 a_4

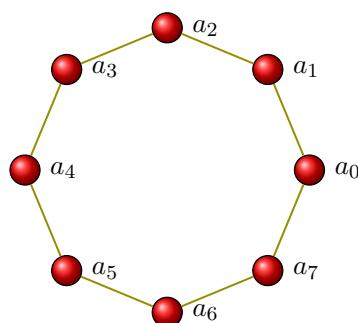




Graf cu muchii plecând dintr-un anumit vârf

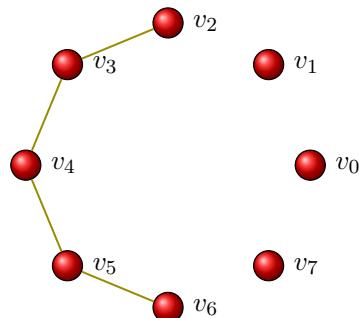


Ciclu



```
\GraphInit[vstyle=Classic]
\SetGraphArtColor{red}{olive}
```

- Desenează un graf între vârful menționat și ultimul vârf



*Tabăra de pregătire a lotului de juniori
Măgurele
18-22 mai 2016*

Teoria Grafurilor

Curs Introductiv

Prof Dana Lica
Centrul Județean de Excelență Prahova - Ploiești

Teoria grafurilor

I. Noțiuni introductive

1.1 Terminologie

Graf \Leftrightarrow orice mulțime finită V , prevăzută cu o relație binară internă E . Notăm graful cu $G=(V,E)$.

Graf neorientat \Leftrightarrow un graf $G=(V,E)$, în care relația binară este simetrică: dacă $(v,w) \in E$, atunci $(w,v) \in E$.

Graf orientat \Leftrightarrow un graf $G=(V,E)$, în care relația binară nu este simetrică.

Nod \Leftrightarrow element al mulțimii V , unde $G=(V,E)$ este un graf neorientat.

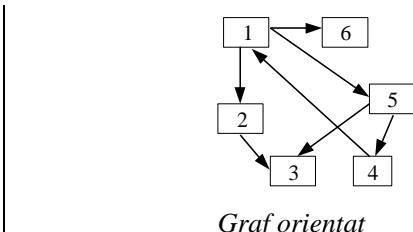
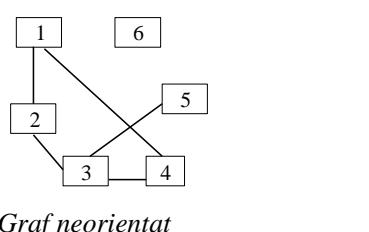
Vârf \Leftrightarrow element al mulțimii V , unde $G=(V,E)$ este un graf orientat sau neorientat.

Muchie \Leftrightarrow element al mulțimii E ce descrie o relație existentă între două vârfuri din V , unde $G=(V,E)$ este un graf neorientat;

Arc \Leftrightarrow element al mulțimii E ce descrie o relație existentă între două vârfuri din V , unde $G=(V,E)$ este un graf orientat;

Arcele sunt parcurse în direcția dată de relația vârf \rightarrow succesor_direct. Muchile unui graf neorientat sunt considerate ca neavând direcție, deci pot fi parcurse în ambele sensuri.

Adiacență \Leftrightarrow Vârful w este adiacent cu v dacă perechea $(v,w) \in E$. Într-un graf neorientat, existența muchiei (v,w) presupune că w este adiacent cu v și v adiacent cu w .



În exemplele din figura de mai sus, vârful 1 este adiacent cu 4, dar 1 și 3 nu reprezintă o pereche de vârfuri adiacente.

Incidență \Leftrightarrow o muchie este incidentă cu un nod dacă îl are pe acesta ca extremitate. Muchia (v,w) este incidentă în nodul v , respectiv w .

Incidență spre interior \Leftrightarrow Un arc este incident spre interior cu un vârf, dacă îl are pe acesta ca vârf terminal (arcul converge spre vârf). Arcul (v,w) este incident spre interior cu vârful w .

Incidență spre exterior \Leftrightarrow Un arc este incident spre exterior cu un vârf dacă îl are pe acesta ca vârf inițial (arcul pleacă din vârf). Arcul (v,w) este incident spre exterior cu vârful v .

Grad \Leftrightarrow Gradul unui nod v , dintr-un graf neorientat, este un număr natural ce reprezintă numărul de noduri adiacente cu acesta.

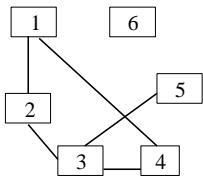
Grad interior \Leftrightarrow În cazul unui graf orientat, fiecare nod v are asociat un număr numit grad interior și care este egal cu numărul de arce care îl au pe v ca vârf terminal (numărul de arce incidente spre interior).

Grad exterior \Leftrightarrow În cazul unui graf orientat, fiecare nod v are asociat un număr numit grad exterior și

care este egal cu numărul de arce care îl au pe v ca vârf inițial (numărul de arce incidente spre exterior).

Vârf izolat \Leftrightarrow Un vârf cu gradul 0.

Vârf terminal \Leftrightarrow Un vârf cu gradul 1.



Vârful 5 este terminal (gradul 1).

Vârful 6 este izolat (gradul 0).

Vârfurile 1, 2, 4 au gradele egale cu 2.

Lanț \Leftrightarrow este o secvență de noduri ale unui graf neorientat $G=(V,E)$, cu proprietatea că oricare două noduri consecutive sunt adiacente:

$w_1, w_2, w_3, \dots, w_p$ cu proprietatea că $(w_i, w_{i+1}) \in E$ pentru $1 \leq i < p$.

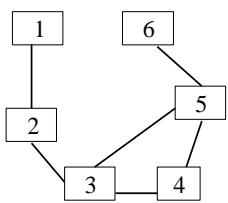
Lungimea unui lanț \Leftrightarrow numărul de muchii din care este format.

Lanț simplu \Leftrightarrow lanțul care conține numai muchii distincte.

Lanț compus \Leftrightarrow lanțul care nu este format numai din muchii distincte.

Lanț elementar \Leftrightarrow lanțul care conține numai noduri distincte.

Ciclu \Leftrightarrow Un lanț în care primul nod coincide cu ultimul. Ciclul este elementar dacă este format doar din noduri distincte, exceptie făcând primul și ultimul. Lungimea minimă a unui ciclu este 3.



Succesiunea de vârfuri 2, 3, 5, 6 reprezintă un lanț simplu și elementar de lungime 3.

Lanțul 5 3 4 5 6 este simplu dar nu este elementar.

Lanțul 5 3 4 5 3 2 este compus și nu este elementar.

Lanțul 3 4 5 3 reprezintă un ciclu elementar.

Drum \Leftrightarrow este o secvență de vârfuri ale unui graf orientat $G=(V,E)$, cu proprietatea că oricare două vârfuri consecutive sunt adiacente:

$(w_1, w_2, w_3, \dots, w_p)$, cu proprietatea că $(w_i, w_{i+1}) \in E$, pentru $1 \leq i < p$.

Lungimea unui drum \Leftrightarrow numărul de arce din care este format.

Drum simplu \Leftrightarrow drumul care conține numai arce distincte.

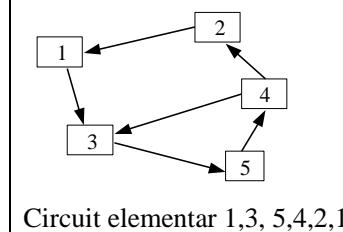
Drum compus \Leftrightarrow drumul care nu este format numai din arce distincte.

Drum elementar \Leftrightarrow drumul care conține numai vârfuri distincte.

Circuit \Leftrightarrow Un drum în care primul vârf coincide cu ultimul. Circuitul este elementar dacă este format doar din vârfuri distincte, exceptie făcând primul și ultimul.

Buclă \Leftrightarrow Circuit format dintr-un singur arc.

Ciclu elementar 3,6,4,5,1,3



Circuit elementar 1,3, 5,4,2,1

Graf parțial \Leftrightarrow Un graf $G'=(V,E')$ reprezintă graf parțial al grafului $G=(V,E)$ dacă $E' \subseteq E$. Cu alte cuvinte G' este graf parțial al lui G , dacă este identic, sau se obține prin suprimarea unor muchii (respectiv arce) din G .

Subgraf \Leftrightarrow Un subgraf al lui $G=(V,E)$ este un graf $G'=(V',E')$ în care $V' \subseteq V$, iar V' conține toate muchiile/arcele din E ce au ambele extremități în V' . Cu alte cuvinte G' este subgraf al lui G , dacă este identic, sau se obține prin suprimarea unor noduri împreună cu muchiile/arcele incidente cu acestea.

Graf regulat \Leftrightarrow graf neorientat în care toate nodurile au același grad.

Graf complet \Leftrightarrow graf neorientat $G=(V,E)$ în care există muchie între oricare două noduri. Numărul de muchii ale unui graf complet este $|V|*(|V|-1)/2$.

Graf conex \Leftrightarrow graf neorientat $G=(V,E)$ în care, pentru orice pereche de noduri (v,w) , există un lanț care le unește.

Graf tare conex \Leftrightarrow graf orientat $G=(V,E)$ în care, pentru orice pereche de vârfuri (v,w) , există drum de la v la w și un drum de la w la v .

Componentă conexă \Leftrightarrow subgraf al grafului de referință, maximal în raport cu proprietatea de conexitate (între oricare două vârfuri există lanț);

Lanț hamiltonian \Leftrightarrow un lanț elementar care conține toate nodurile unui graf.

Ciclu hamiltonian \Leftrightarrow un ciclu elementar care conține toate nodurile grafului.

Graf hamiltonian \Leftrightarrow un graf G care conține un ciclu hamiltonian.

Condiție de suficiență: Dacă G este un graf cu $n \geq 3$ vârfuri, astfel încât gradul oricărui vârf verifică inegalitatea: $\text{gr}(x) \geq n/2$, rezultă că G este graf hamiltonian.

Lanț eulerian \Leftrightarrow un lanț simplu care conține toate muchiile unui graf.

Ciclu eulerian \Leftrightarrow un ciclu simplu care conține toate muchiile grafului.

Graf eulerian \Leftrightarrow un graf care conține un ciclu eulerian.

Condiție necesară și suficientă: Un graf este eulerian dacă și numai dacă oricare vârf al său are gradul par.

1.2 Moduri de reprezentare la nivelul memoriei

Există mai multe modalități standard de reprezentare a unui graf $G=(V, E)$:

1. matricea de adiacență;
2. liste de adiacență;
3. matricea ponderilor (costurilor);
4. lista muchilor.

- *Matricea de adiacență* este o matrice binară (cu elemente 0 sau 1) care codifică existența sau nu a muchiei/arcului între oricare pereche de vârfuri din graf. Este indicată ca mod de memorare a grafurilor, în special în cazul grafurilor dense, adică cu număr mare de muchii ($|V|^2 \approx E$).

```

1 Creare_MA_neorientat(G=(V, E))      /*complexitate: O(|V|^2 + |E|) */
2 [pentru i ← 1, |V| execută
3   [pentru j ← 1 to |V| execută G[i][j] ← 0;
4
5   ]
6 [pentru e = 1, |E| execută
7   citeste i, j;
8   G[i][j] ← 1;
9   G[j][i] ← 1;      //instructiunea lipseste in cazul digrafului
10 ]

```

Implementarea în limbaj a subprogramului care creează matricea de adiacență a unui graf neorientat ia în considerare următoarele declarații:

```

const MAX_N=101;
      MAX_M=1001;
var n,m:longint;
G:array[0..max_n,0..max_n] of byte;

```

```

#include <stdio.h>
#define MAX_N 101
#define MAX_M 1001
int N, M, ; char G[MAX_N][MAX_N];

```

Subprogramul *citeste_graf()* preia informațiile din fișierul text *graf.in*, în felul următor: de pe prima linie numărul *n* de vârfuri și *m* de muchii, iar de pe următoarele *m* linii, perechi de numere reprezentând muchiile grafului. Matricea *G* de adiacență se consideră inițializată cu valoarea 0.

```

1 void citeste_graf(void)
2 {
3     int i, j;
4     freopen("graf.in", "r", stdin);
5     scanf("%d %d", &N, &M);
6     for (; M > 0; M--)
7     {
8         scanf("%d %d", &i, &j);
9         G[i][j] = G[j][i] = 1;
10    }
11 }
12

```

```

void citeste_graf(void)
{
    int i, j;
    freopen("graf.in", "r", stdin);
    scanf("%d %d", &N, &M);
    for (; M > 0; M--)
    {
        scanf("%d %d", &i, &j);
        G[i][j] = 1;
    }
}

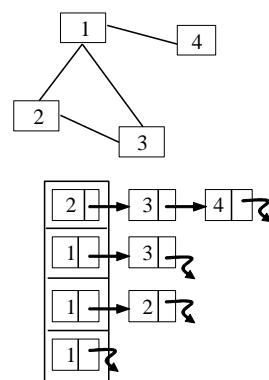
```

- *Listele de adiacență* rețin, pentru fiecare nod din graf, toate vâfurile adiacente cu acesta (vecine cu el). Ca modalitate de reprezentare se poate folosi un vector *LA* ale căruia elemente sunt adresele de început ale celor $|V|$ liste simplu înlăncuite memorate, una pentru fiecare vârf din V . Lista simplu înlăncuită, a cărei adresă de început este reținută de $LA[u]$ memorează toate vâfurile din G , adiacente cu u și stocate într-o ordine oarecare.

Pentru graful alăturat exemplificăm reprezentarea atât în liste de adiacență, cât și în matrice de adiacență.

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix}$$

Matricea de adiacență (MA)



Liste de adiacență (LA)

```
# include <vector>
using namespace std;
vector <int> G[MAXN];
# include <stdio.h>
#define MAXN 1001
struct lista
{ int nod;
  lista *urm;
} *G[MAXN];
```

Subprogramul *citeste_graf()* preia informațiile din fișierul text *graf.in*, în felul următor: de pe prima linie, numărul n de vârfuri și m de muchii, iar de pe următoarele m linii, perechi de numere reprezentând muchiile grafului. La apelul *adauga(i,j)* se realizează inserarea unui element de informație j în lista lui vecinilor nodului i . Tabloul G , reprezentând liste de adiacență, se consideră inițializat cu constanta *NULL* (C++).

<pre> 1 void load(){ 2 scanf("%d %d\n", &N, &M); 3 for (i=1; i<= M; i++){ 4 scanf("%d %d\n", &x, &y); 5 G[x].push_back(y); 6 G[y].push_back(x); 7 } 8 return; 9 } 10 11 12 13 14 15 16 17 </pre>	<pre> void adauga(int i, int j) { lista *p; p = new lista; p->nod = j; p->urm = G[i]; G[i] = p; } void citeste_graf(void) { int i, j; freopen("graf.in","r", stdin); scanf("%d %d", &N, &M); for (; M > 0; M--){ scanf("%d %d", &i, &j); adauga(i, j); adauga(j, i); } } </pre>
---	---

Un potențial dezavantaj al reprezentării sub forma listelor de adiacență este modul oarecum anevoieios de a determina dacă o muchie (u,v) este prezentă sau nu în graf. Eficiența reprezentării unui graf este dată de densitatea acestuia, deci matricea de adiacență este viabilă dacă numărul muchiilor este aproximativ egal cu $|V|^2$.

- *Matricea costurilor* este o matrice cu $|V|$ linii și $|V|$ coloane, care codifică existența sau nu a muchiei/arcului, între oricare pereche de vârfuri din graf, prin costul acesteia. Astfel:

$G[i][j] = \infty$, dacă $(i, j) \notin E$
 $G[i][j] = cost < \infty$, dacă $(i, j) \in E$
 $G[i][j] = 0$, dacă $i = j$

<pre> 1 Creare_MC_neorientat(G=(V,E)) /*complexitate: O(V^2 + E) */ 2 pentru i=1, V execută 3 pentru j ← 1 to V execută 4 dacă i = j atunci G[i][j] ← 0; 5 altfel G[i][j] ← ∞ 6 7 8 9 pentru e = 1, E execută 10 citeste i, j, c; 11 G[i][j] ← c; 12 G[j][i] ← c; //instructiunea lipseste in cazul digrafului 13 </pre>
--

Implementarea în limbaj a subprogramului care creează matricea costurilor unui graf neorientat ia în considerare următoarele declarații:

```
#include <algorithm>
#include <stdio.h>
#include <vector>
#define pb push_back
#define f first
#define s second
#define mp make_pair
#define MAXN 10001

using namespace std;

int N, M;
vector<pair<int, int>> L[MAXN];
```

```
#include <stdio.h>
#include <string.h>

#define MAXN 10001
#define INF 0x3f3f

int N, M, G[MAXN][MAXN];
```

Subprogramul *citeste_graf()* preia informațiile din fișierul text *graf.in* în felul următor: de pe prima linie, numerele n de vârfuri și m de muchii, iar de pe următoarele m linii, câte trei numere i, j, c având semnificația muchiei între i și j de cost c .

```
1 void load(){
2     scanf("%d %d\n", &N, &M);
3     for (i=1; i<= M; i++) {
4         scanf("%d %d %d\n", &x, &y, &c);
5         G[x].pb(mp(y,c));
6         G[y].pb(mp(x,c));
7     }
8     return;
9 }
10}
11}
12}
13}
14}
```

```
void citeste_graf(void) {
    int i, j, c;
    freopen("graf.in","r",stdin);

    scanf("%d %d", &N, &M);
    for (i = 1; i <= N; i++)
        for (j = 1; j <= N; j++)
            G[i][j] = (i != j) * INF;
    for (; M > 0; M--) {
        scanf("%d %d %d", &i, &j, &c);
        G[i][j] = c;
        G[j][i] = c;
    }
}
```

- *Lista muchiilor* este utilizată în cadrul unor algoritmi, ea memorând, pentru fiecare muchie, cele două vârfuri incidente și eventual, costul ei.

Implementarea în limbaj a subprogramului care creează lista muchiilor unui graf neorientat ponderat, ia în considerare următoarele declarații:

```
#include <algorithm>
#include <stdio.h>
#include <vector>
#define pb push_back
#define f first
#define s second
#define mp make_pair

using namespace std;

int N, M;
vector<pair<int, pair<int, int>> L;
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX_N 101
#define MAX_M 10001

int N, M, E[MAX_M][3];
```

Tabloul E reține lista muchiilor. Subprogramul *citeste_graf()* preia informațiile din fișierul text *graf.in* în felul următor: de pe prima linie, numărul n de vârfuri și m de muchii, iar de pe următoarele m linii, câte trei numere i, j, c având semnificația muchiei între i și j de cost c .

```
1 void load(){
2     int x, y, c;
3     scanf("%d %d", &N, &M);
4     for (int i = 0; i < M; i++) {
5         scanf("%d %d %d", &x, &y, &c);
6         L.pb (mp (c, mp(x, y) ) );
7     }
8 }
```

```
void citeste_graf(void) {
    int i, j, k, c;
    freopen("graf.in","r",stdin);
    scanf("%d %d", &N, &M);
    for (k = 0; k < M; k++) {
        scanf("%d %d %d", &i, &j, &c);
        E[k][0] = i; E[k][1] = j;
        E[k][2] = c;
    }
}
```

2. Algoritmi pe grafuri

1.2 Parcurgeri pe grafuri

1. Parcugerea grafurilor în adâncime DF - Depth First

Fie graful $G=(V,E)$, unde V este mulțimea nodurilor și E mulțimea muchiilor. Realizați un subprogram care să permită afișarea nodurilor în cazul parcurgerii în adâncime-DF.

Soluție:

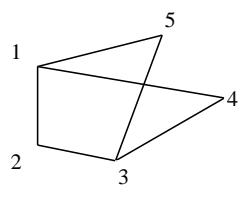
Printre operațiile fundamentale efectuate asupra structurilor de date se numără și traversarea acestora. Această operație constă într-o căutare efectuată asupra tuturor elementelor ei. Cu alte cuvinte, traversarea poate fi privită și ca un caz special de căutare, deoarece constă în regăsirea tuturor elementelor structurii.

Strategia parcurgerii în adâncime a unui graf neorientat presupune traversarea unei muchii incidente în vârful curent, către un vârf adjacent nedescoperit. Când toate muchiile vârfului curent au fost explorate, se revine în vârful care a condus la explorarea nodului curent. Procesul se repetă până în momentul în care toate vârfurile au fost explorate.

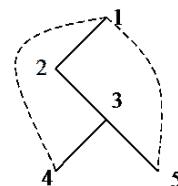
Această strategie a parcurgerii DF funcționează respectând mecanismul LIFO. Vârful care este eliminat din stivă nu mai are nici o muchie disponibilă pentru traversare. Aceleași reguli se respectă și la parcugerea în adâncime a grafurilor orientate (*digrafuri*).

În procesul de parcugere, muchiile unui graf neorientat se pot împărți în:

- *muchii de arbore (avans)*, folosite pentru explorarea nodurilor; ele constituie un graf parțial format din unul sau din mai mulți arbori ai parcurgerii DF.
- *muchii de întoarcere (înapoi)*, care unesc un nod cu un strămoș al său în arborele DF.



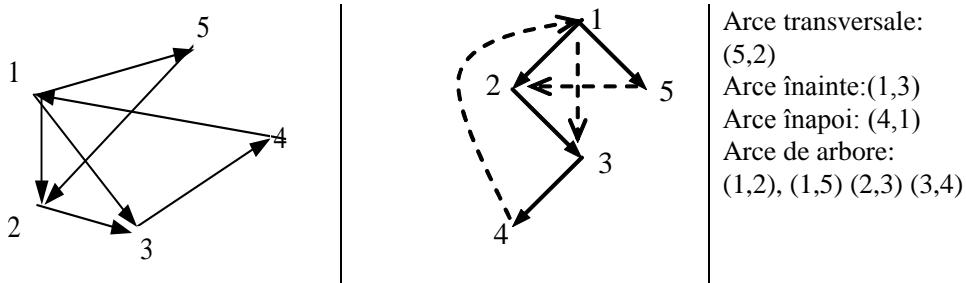
Parcugerea DF: 1 2 3 4 5



Muchiile de întoarcere: (1,4), (1,5)

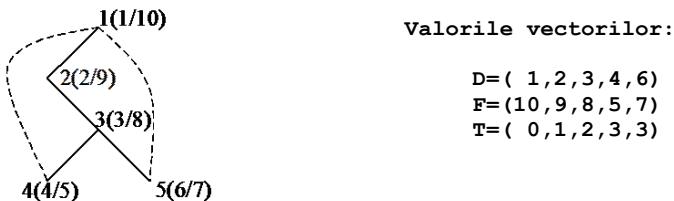
În cazul digrafurilor (grafurilor orientate), se disting următoarele grupe de arce:

- *arce de arbore*, folosite pentru explorarea vârfurilor; ele constituie un graf parțial format din unul sau din mai mulți arbori ai parcurgerii DF.
- *arce înapoi*, care unesc un nod cu un strămoș al său în arborele DF.
- *arce înainte*, sunt arce care nu aparțin arborelui DF și care conectează un vârf cu un descendant al său.
- *arce transversale*, sunt arce care conectează două vârfuri ce nu se află în relația ascendent-descendent.



În implementarea recursivă a parcurgerii *DF*, prezentată în continuare, se folosesc următoarele tablouri unidimensionale:

- $T[N]$, în care, pentru fiecare vârf, se va reține vârful predecesor (părinte) în parcurgere;
- $D[N]$, în care, pentru fiecare vârf, se memorează momentul “descoperirii”, moment care coincide cu momentul de introducere în stivă;
- $F[N]$, vector în care va memora momentul de “finish” în explorare, care coincide cu momentul extragerii din stivă;
- $Use[N]$, vector în care se codifică, în timpul parcurgerii, starea unui nod: vizitat sau nevizitat.



Fie graful $G=(V,E)$, unde V este mulțimea nodurilor și E mulțimea muchiilor. Notăm cu N cardinalul lui V și cu M cardinalul lui E .

```

1 | Depth_First (G=(V,E), D[N], F[N], T[N])    //complexitate O(N+M)
2 | Use ← False;                                //nici un nod nu este vizitat
3 | timp ← 0;
4 | pentru nod ← 1,N executa
5 |   daca Not(Use[nod]) atunci Parcurge_df(nod)
6 |
7 |
8 | Parcurge_df(nod)
9 | Use[nod] ← TRUE; timp ← timp+1; D[nod]←timp;
10 | scrie nod;
11 | i ← prim_vecin(nod);
12 | cat timp lista_vecinilor(nod)≠∅ executa
13 |   daca not(Use[i]) atunci
14 |     T[i]←nod;
15 |     parcurge_df(i);
16 |
17 |     i ← urmator_vecin(nod);
18 |
19 |   timp ← timp+1;
20 |   F[nod] ← timp;

```

Implementarea în limbaj a subprogramului ce realizează parcurgerea în adâncime a unui graf, prezentată în continuare, ia în considerare următoarele declarații:

```

#include <vector>
#include <stdio.h>
#include <string.h>
#define MAX_N 1001
#define pb push_back

using namespace std;

vector <int> L[MAXN];
int n, m, i, x, y, nrc, timp;
int T[MAXN], F[MAXN];
bool U[100001];
    
```

```

#include <stdio.h>
#include <string.h>
#define MAXN 10001

struct lista
{
    int nod;
    lista *urm;
} *G[MAXN];
int N, M, T[MAXN], D[MAXN], F[MAXN],
timp;
char U[MAXN];
    
```

Ca și în pseudocod, s-a preferat implementarea în manieră recursivă, iar graful este considerat a fi memorat cu ajutorul listelor de adiacență (vezi 2.1.2).

<pre> 1 void dfs(int x){ 2 3 int i; 4 U[x] = true; 5 D[x] = ++timp; 6 for(i = 0; i < L[x].size(); i++) 7 if (!U[L[x][i]]){ 8 T[L[x][i]] = x; 9 dfs(L[x][i]); 10 } 11 F[x] = ++timp; 12 } 13 } 14 </pre>	<pre> void DF(int nod) { lista *p; U[nod] = 1; D[nod] = ++timp; printf("%d ",nod); for (p = G[nod]; p != NULL; p = p->urm) if (!U[p->nod]) { T[p->nod] = nod; DF(p->nod); } F[nod] = ++timp; } </pre>
---	---

2. Parcugerea grafurilor în lățime BF - Breadth First

Fie graful $G=(V,E)$, unde V este mulțimea nodurilor și E mulțimea muchiilor. Realizați un subprogram care să permită afișarea nodurilor în cazul parcurgerii în lățime-BF.

Soluție:

Strategia parcurgerii *BF* funcționează respectând mecanismul de tip *FIFO*. Ea se bazează pe traversarea tuturor muchiilor disponibile din nodul curent către noduri adiacente nedescoperite, care vor fi astfel vizitate. După acest proces, nodul explorat este scos din coadă, prelucrându-se succesiv toate nodurile ajunse în vârful cozii.

Acest mecanism permite identificarea lanțurilor de lungime minimă de la nodul de start către toate vârfurile accesibile lui din graf. Arboarele *BF*, ce cuprinde muchiile traversate în parcugerea în lățime, are proprietatea de a fi format doar din lanțuri de lungime minimă, lanțuri ce unesc nodul de start al parcurgerii cu toate vârfurile accesibile acestuia.

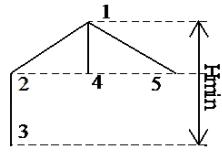
Aceleași reguli se respectă și la parcugerea în lățime a grafurilor orientate (*digrafuri*).

Algoritmul lui *Lee* de determinare a lanțului minim dintre două vârfuri ale unui graf se bazează tocmai pe această strategie de traversare în lățime, nodul de plecare fiind unul dintre cele două vârfuri date.

În implementarea iterativă a parcurgerii *BF*, prezentată în continuare, se folosesc următoarele tablouri unidimensionale:

- $T[N]$, în care, pentru fiecare vârf, se va reține vârful predecesor (părinte) în parcugere;
- $D[N]$, în care, pentru fiecare vârf, se memorează lungimea lanțului minim către nodul de start;
- $C[N]$, vector (coadă) în care se va memora ordinea de parcugere a nodurilor în *BF*;
- $Use[N]$, vector în care se codifică, în timpul parcurgerii, starea unui nod: vizitat sau nevizitat.

Pentru graful considerat anterior ca exemplu, arboarele *BF* este următorul:



Valorile vectorilor:

$$C = (1, 2, 4, 5, 3)$$

$$D = (0, 1, 2, 1, 1)$$

$$T = (0, 1, 2, 1, 1)$$

Fie graful $G=(V,E)$, unde V este mulțimea nodurilor și E , mulțimea muchiilor. Notăm cu N cardinalul lui V și cu M cardinalul lui E .

```

1 | Breadth_First (G=(V,E), C[N] T[N], Nod_Start)
2 |                                         /*complexitate: O(M+N) */
3 | Use ← FALSE;
4 | Coada ← {Nod_Start}      //nodul de start este introdus in coada
5 | Use[nod] ← True;
6 | cat_timp Coada ≠ φ executa
7 |     v ← varf_Coada; i ← prim_vecin(v);
8 |     cat_timp lista_vecinilor(v) ≠ φ executa
9 |         daca not(Use[i]) atunci
10 |             T[i] ← v; Use[i] ← TRUE;
11 |             D[i] ← D[v]+1;
12 |             Coada ← {i};
13 |
14 |             i ← urmator_vecin(v);
15 |
16 |
17 | COADA ⇒ v;

```

Implementarea în limbaj a subprogramului ce realizează parcurgerea în lățime a unui graf, prezentată în continuare, ia în considerare următoarele declarații:

```

#include <queue>
#include <vector>
#include <stdio.h>
#include <cstring>
#define MAXN 10001
#define pb push_back

using namespace std;

vector <int> G[100001];
queue <int> Q;
int N, M, i, x, y, P, Lg[MAXN], T[MAXN];
bool U[MAXN];

```

```

#include <stdio.h>
#include <string.h>
#define MAX_N 1001
struct lista
{
    int nod;
    lista *urm;
} *G[MAX_N];

int N, M;
int T[MAX_N], D[MAX_N], C[MAX_N];
char U[MAX_N];

```

Ca și în pseudocod, s-a preferat implementarea în manieră iterativă, iar graful este considerat a fi memorat cu ajutorul listelor de adiacență.

```

1 | void bfs(int x){
2 |
3 |     Q.push(x); Lg[x] = 0; U[x] = true;
4 |
5 |     while (!Q.empty()){
6 |         x = Q.front();
7 |         for (i = 0; i < G[x].size(); i++)
8 |             if (!U[G[x][i]])
9 |                 {
10 |                     Q.push(G[x][i]);
11 |                     Lg[G[x][i]] = Lg[x]+1;
12 |                     U[G[x][i]] = true;
13 |                     T[G[x][i]] = x;
14 |                 }
15 |         Q.pop();
16 |     }
17 |

```

```

void bfs(int start) {
    lista *p;
    int nod, st, dr;
    st = dr = 1;
    C[start]=start; U[start] = 1;
    for (D[start]=0;st<=dr;st++)
    {nod=C[st];
        for (p = G[nod];p != NULL;
            p = p->urm)
            if (!U[p->nod])
            {
                U[C[st+dr] = p->nod] = 1;
                D[p->nod] = D[nod]+1;
                T[p->nod] = nod;
            }
    }
}

```

3. Ciclu eulerian

Fie $G=(V,E)$, graf neorientat conex, în care fiecare nod are gradul par. Notăm cu N cardinalul lui V și cu M , cardinalul lui E . Să se determine un ciclu eulerian al grafului G , altfel spus, un ciclu simplu de lungime M .

Exemplu:

Considerând graful G în care $N=6$, $M=10$ și arcele: $(1,2)$, $(1,3)$, $(2,3)$, $(2,4)$, $(2,5)$, $(3,4)$, $(3,5)$, $(4,5)$, $(4,6)$, $(5,6)$, se va afișa: $1, 3, 5, 6, 4, 5, 2, 4, 3, 2, 1$.

Soluție:

Pentru a construi un ciclu eulerian se va parcurge graful folosind o strategie asemănătoare cu cea a parcurgerii în adâncime a unui graf, strategie care respectă mecanismul *LIFO*.

Înaintarea către un vîrf adjacent vîrfului curent se va face simultan cu eliminarea muchiei respective. În acest fel, nodurile nu vor mai fi marcate (ca la parcurgerea *DF*) pentru a putea fi vizitate de mai multe ori.

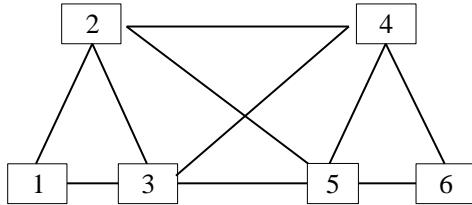
De fiecare dată când nodul curent este eliminat din stivă, acesta este afișat sau salvat într-o coadă. Această coadă va reține, în ordine, nodurile ciclului eulerian. Procesul se repetă până în momentul în care stiva devine vidă, deci toate muchiile au fost traversate.

```

1 | Ciclu_Euler (Nod)           /*complexitate: O (M+N) */
2 |
3 |   i ← prim_vecin(nod);
4 |   cat timp lista_vecinilor(nod) ≠ φ executa
5 |     elimin(i,nod)           //elimin pe i din vecinii lui nod
6 |     elimin(nod,i)          //elimin pe nod din vecinii lui i
7 |     ciclu_euler(i)
8 |     i ← urmator_vecin(nod);
9 |
10|   Coada ← {nod};

```

Considerăm graful următor și desemnăm ca nod de start pentru construirea ciclului eulerian, nodul 1:



Pasul 1:

Se parcurge începând cu nodul 1

Stiva = $(1,2,3,1)$

Coada = \emptyset

Pasul 2:

Iese din stivă nodul 1, salvându-se în coadă:

Stiva = $(1,2,3)$

Coada = (1)

Pasul 3:

Se continuă parcurgerea:

Stiva = $(1,2,3,4,2,5,3)$

Coada = (1)

Pasul 4:

Iese din stivă nodul 3, salvându-se în coadă:

Stiva = $(1,2,3,4,2,5)$

Coada = $(1,3)$

Pasul 5:

Se continuă parcurgerea:

Stiva = $(1,2,3,4,2,5,4,6,5)$

Coada = $(1,3)$

Pasul 6:

Iese din stivă nodul 5, salvându-se în coadă:

Stiva = $(1,2,3,4,2,5,4,6)$

Coada = $(1,3,5)$

Pasul 7:

Toate nodurile sunt extrase din stivă și introduse în coadă. La finalul algoritmului:

Stiva = \emptyset

Coada = $(1,3,5,6,4,5,2,4,3,2,1)$.

Implementarea în limbaj a subprogramului ce realizează determinarea unui ciclu eulerian, prezentată în continuare, ia în considerare următoarele declarații:

```
#include <cstdio>
#include <vector>
#include <list>
#include <cstring>
#define MAXN 1001

using namespace std;

vector<int> G[MAXN];
list <int> Q;
int nc, nr, i, k, N, M, x, y;
bool ok; char s[100000];

#include <stdio.h>
#define MAX_N 1001
#define MAX_M 100001

int N, M, C[MAX_M], nc;
char G[MAX_N][MAX_N];
```

Ca și în pseudocod, s-a preferat implementarea în manieră recursivă, iar graful este considerat a fi memorat cu ajutorul matricei de adiacență. Datorită acestui mod de memorare, implementarea în limbaj conduce la o complexitate pătratică $O(N^2)$.

```
1 void euler(int x) {
2     vector<int>::iterator it;
3
4     Q.pb(x);
5     while (!Q.empty()) {
6         x = Q.front();
7         if (G[x].empty()) {
8             Q.pop_front();
9             printf("%d ", x);
10        }
11    else {
12        int i = G[x].back();
13        Q.push_front(i);
14        G[x].pop_back();
15        for (it=G[i].begin(); it!=G[i].end(); ++it)
16            if (*it==x) {G[i].erase(it); break;}
17    }
18 }
19 }
20 }
```

```
void euler(int nod)
{
int urm;

for (urm = 1; urm <= N; urm++)
{
if (G[nod][urm])
{
    G[nod][urm] = 0;
    G[urm][nod] = 0;
    euler(urm);
}
C[nc++] = nod;
}
```

În cazul în care se dorește un lanț eulerian, nu un ciclu, se poate aplica același algoritm începând dintr-un nod cu grad impar, dacă există vreunul. Un lanț eulerian se poate forma numai dacă există zero sau două noduri cu grad impar.

4. Matricea drumurilor - Algoritmul lui Warshall

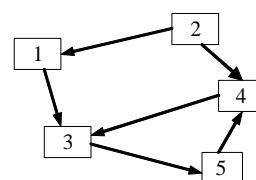
Fie $G=(V,E)$, graf orientat cu n vârfuri și m arce. Să se determine, pentru orice pereche de vârfuri $x, y \in V$, dacă există sau nu un drum format din unul sau din mai multe arce între x și y .

Soluție:

Numim matricea drumurilor sau a închiderii tranzitive, o matrice pătratică $D(N,N)$ în care elementul $D[i,j]=1$, dacă există drum între nodul i și nodul j și 0, în caz contrar.

De exemplu, pentru graful ilustrat alăturat, matricea drumurilor este următoarea:

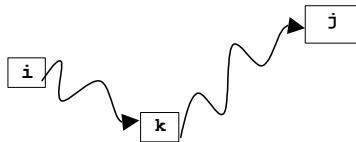
```
0 0 1 1 1
1 0 1 1 1
0 0 1 1 1
0 0 1 1 1
0 0 1 1 1
```



Algoritmul Warshall construiește matricea închiderii tranzitive D , plecând de la matricea de adiacență G .

Inițial, matricea D indică prezența drumurilor de lungime 1 între orice pereche de vârfuri adiacente.

Algoritmul parcurge de n ori matricea D , câte o dată pentru fiecare nod k al grafului. Astfel, pentru fiecare nod k , între orice pereche de noduri i și j din graf, fie s-a descoperit deja un drum ($D[i,j]=1$), fie acesta poate fi construit prin concatenarea drumurilor de la i la k și de la k la j , dacă acestea există.



Luând în considerare caracteristica logică a valorilor elementelor matricei D , atunci regula descrisă mai sus poate fi exprimată astfel:

$$D[i,j] = D[i,j] \text{ OR } \{D[i,k] \text{ AND } D[k,j]\}$$

```

1 | Warshall (G=(V,E))
2 | ///*complexitate: O(N^3) */
3 | D ← G;
4 | pentru k ← 1,n executa
5 |   pentru i ← 1,n executa
6 |     pentru j ← 1,n executa
7 |       D[i,j] ← D[i,j] SAU {D[i,k] SI D[k,j]}
8 |
9 |
10| 
```

Implementarea în limbaj a subprogramului ce realizează determinarea matricei închiderii tranzitive, prezentată în continuare, ia în considerare următoarele declarații:

```
#define MAX_N 101
#define MAX_M 1001
int N, M, C[MAX_M], nc;
char G[MAX_N][MAX_N], D[MAX_N][MAX_N];
```

Ca și în pseudocod, graful este considerat a fi memorat cu ajutorul matricei de adiacență. Complexitatea algoritmului lui *Warshall* este cubică $O(N^3)$.

```

1 | void Warshall()
2 | {
3 |   int i,j,k;
4 |   for (i = 1; i <= N; i++)
5 |     for (j = 1; j <= N; j++)
6 |       D[i][j] = G[i][j];
7 |     for (k = 1; k <= N; k++)
8 |       for (i = 1; i <= N; i++)
9 |         for (j = 1; j <= N; j++)
10|           if (!D[i][j])
11|             D[i][j] = D[i][k] && D[k][j];
12| } 
```

5. Componente conexe

Fie $G=(V,E)$ un graf neorientat. Se dorește determinarea componentei conexe cu număr maxim de noduri din G . Componenta va fi identificată printr-unul dintre vâfurile sale și numărul de vârfuri din care este formată.

Exemplu: Considerând graful G în care $N=6$, $M=6$ și arcele: (1,2), (3,4), (3,5), (4,5), (4,6), (5,6) se va afișa: 3 4 (nodul 3 face parte din componenta conexă formată din 4 noduri).

Soluție:

Problema va fi rezolvată prin determinarea tuturor componentelor conexe ale grafului și identificarea componentei cu număr maxim de noduri. Ne reamintim că o componentă conexă este un subgraf maximal în raport cu proprietatea de conexitate.

Pentru a descompune graful în componente conexe, vom proceda în felul următor: vom realiza o

parcurgere DF din nodul 1, determinându-se componenta conexă din care acestea fac parte. Vom continua algoritmul cu o nouă parcursere efectuată dintr-un nod nevizitat anterior. Procedeul continuă până când s-au vizitat toate nodurile.

Algoritmul de descompunere în componente conexe a unui graf neorientat este prezentat în pseudocod, în continuare.

```

1 | Componente_conexe ( $G=(V, E)$ )           /*complexitate:  $O(M+N)$  */
2 |   nrc  $\leftarrow 0$ ;                         // nr de componente conexe
3 |   Use  $\leftarrow$  False;                      // nici un nod selectat
4 |   pentru  $i \leftarrow 1, n$  executa
5 |     daca Not(Use[i]) atunci
6 |       nrc  $\leftarrow$  nrc + 1;
7 |       parcurge_df(i)
8 |
9 |

```

Pentru a identifica cea mai numeroasă componentă conexă, vom determina în cadrul fiecărei parcurgeri DF , numărul de noduri selectate.

6. Tare-conexitate

Fie $G=(V,E)$ un graf orientat. Realizați un program care afișează vârfurile fiecărei componente tare conexe în care se descompune graful G .

Exemplu: Considerând digraful G în care $N=5$, $M=6$ și arcele: $(1,2)$, $(1,5)$, $(2,3)$, $(3,4)$, $(3,5)$, $(4,1)$, se vor afișa două componente tare conexe:

1 4 3 2
5

Soluție:

În cazul digrafurilor (grafurilor orientate), o componentă tare conexă reprezintă un subgraf maximal în care, oricare două vârfuri sunt accesibile unul din celălalt.

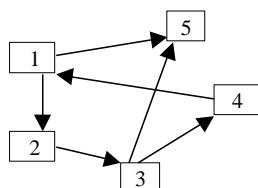
În cadrul unei componente tare conexe, inversarea sensului de deplasare nu implică blocarea accesului către vreunul din vârfurile sale.

Algoritmul propus identifică componenta tare conexă a unui vârf ca fiind intersecția dintre mulțimea nodurilor accesibile din el în graful inițial G și mulțimea nodurilor accesibile din el în graful transpus Gt (obținut prin inversarea sensurilor arcelor).

Acest algoritm se bazează pe parcurgerea DF a celor două grafuri, de aici și complexitatea liniară a acestuia $O(N+M)$. Operațiile efectuate sunt:

- Parcurserea DF a grafului inițial (G) pentru memorarea explicită a stivei ce conține vârfurile grafului în ordinea crescătoare a timpilor de finish.
- Parcurserea DF a grafului transpus (Gt) începând cu ultimul vârf reținut în stivă către primul. Parcurserea se reia din fiecare vârf rămas nevizitat la parcurgerile anterioare. Vârfurile fiecărui arbore DF obținut la acest pas reprezintă câte o componentă tare conexă.

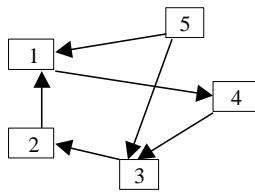
Pentru graful:



Pasul 1:

Stiva DF cuprinzând nodurile în ordinea crescătoare a timpilor de finish este:

$$St = (4, 5, 3, 2, 1).$$



Pasul 2:

Parcurgerea DF pe graful transpus începând cu $St[2]$ acceseează vârfurile din prima componentă tare conexă: {1,4,3,2}.

Parcurgerea DF din primul vârf rămas neselectat $St[2]$ acceseează vârfurile celei de a doua componente tare conexe: {5}.

Implementarea în limbaj a subprogramelor prezentate în continuare ia în considerare următoarele declarații:

```
#include <cstdio>
#include <vector>
#include <cstring>
#define pb push_back
#define MAXN 10001
using namespace std;

vector<int> G[MAXN], GT[MAXN];
int nc, nr, i, k, N, M, x, y, St[MAXN];
bool sel[MAXN];
```

```
#include <stdio.h>
#include <string.h>
#define MAX_N 10001

struct lista
{
    int nod;
    lista *urm;
} *G[MAX_N], *GT[MAX_N];
int N, M, ST[MAX_N], nst;
char U[MAX_N];
```

Tabloul GT va reține graful transpus asociat lui G . Subprogramul *citeste_graf* preia datele referitoare la graful G și construiește matricele de adiacență ale celor două grafuri G și GT .

```
1 void DF(int x){
2     int i;
3     sel[x]=true;
4     for(i = 0; i < G[x].size(); ++i)
5         if (!sel[G[x][i]]) DF(G[x][i]);
6     St[++nr]=x;
7 }
8
9 void DFT(int x, bool k){
10    int i;
11    sel[x]=true;
12    if (k) printf("%d ", x);
13    for(i = 0; i < GT[x].size(); ++i)
14        if (!sel[GT[x][i]])
15            DFT(GT[x][i], k);
16 }
17
18 int main(){
19     . . .
20     scanf("%d %d\n", &N, &M);
21     for (i=1; i<=M; ++i){
22         scanf("%d %d\n", &x, &y);
23         G[x].pb(y); GT[y].pb(x);
24     }
25     nr=0; nc=0;
26     memset(sel, false,sizeof(sel));
27     for(i=1; i<=N; i++)
28         if (!sel[i]) DF(i);
29     memset(sel, false,sizeof(sel));
30     for(i =N; i>0; --i)
31         if (!sel[St[i]]){
32             DFT(St[i], false);
33             nc++;
34         }
35     printf("%d\n", nc);
36     memset(sel, false,sizeof(sel));
37     for(i=N; i>0; --i)
38         if (!sel[St[i]]){
39             DFT(St[i], true);
40             printf("\n");
41         }
42     return 0;
}
```

```
void DF1(int nod)
{
    lista *p;

    U[nod] = 1;
    for (p = G[nod]; p != NULL;
         p = p->urm)
        if (!U[p->nod])
            DF1(p->nod);
    ST[nst++] = nod;
}

void DF2(int nod)
{
    lista *p;

    U[nod] = 1;
    printf("%d ", nod);
    for (p = GT[nod]; p != NULL;
         p = p->urm)
        if (!U[p->nod])
            DF2(p->nod);
}

void citeste_graf(void)
{.....}

int main(void)
{int i; citeste_graf();
 for (i = 1; i <= N; i++)
    if (!U[i]) DF1(i);
 memset(U, 0, sizeof(U));
 for (i = N-1; i >= 0; i--)
    if (!U[ST[i]])
        DF2(ST[i]);
    putchar('\n');
}  return 0;
}
```

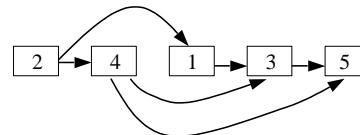
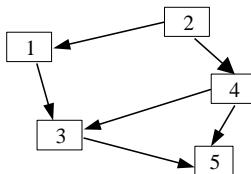
7. Sortarea topologică a unui digraf

Fie $G=(V,E)$ un graf orientat aciclic. Realizați un program care ordonează vârfurile grafului după următoarea regulă: pentru orice arc $(x,y) \in E$, vârful x apare, în sirul ordonat, înaintea vârfului y .

Exemplu: Considerând digraful G în care $N=5$, $M=6$ și arcele: $(1,3), (2,1), (2,4), (3,5), (4,3), (4,5)$, se va afișa $2\ 4\ 1\ 3\ 5$.

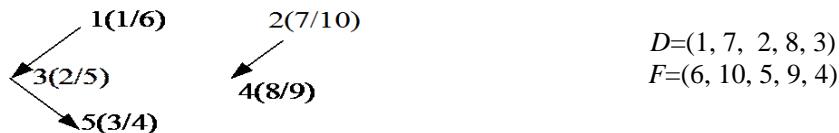
Soluție:

Pentru a înțelege mai bine modul de ordonare a vârfurilor, realizat de sortarea topologică, să urmărim graful din exemplu:



O ordonare corectă a vârfurilor este: $2\ 4\ 1\ 3\ 5$

Presupunem că se realizează o parcurgere DF a grafului prezentat. Vectorii D și F care rețin timpii de intrare, respectiv timpii de ieșire din stivă sunt:



Dacă vârfurile sunt reținute într-o listă, în ordinea ieșirii lor din stivă, atunci această listă va conține: $S_t=(5, 3, 1, 4, 2)$. Afisarea conținutului ei în ordine inversă, adică în ordinea inversă timpilor de finish, va reprezenta ordinea vârfurilor obținută prin sortarea topologică $2, 4, 1, 3, 5$.

Să presupunem vârfurile digrafului ca niște evenimente și fiecare arc (x,y) considerăm că indică faptul că evenimentul x trebuie să se execute înaintea evenimentului y . Plecând de la aceste considerante, deducem că sortarea topologică induce o ordine asupra evenimentelor, astfel încât acestea să poată fi executate.

Să asociem următoarele evenimente grafului anterior: Nodul 1 - Mă îmbrac; Nodul 2 - Mă trezesc; Nodul 3 - Servesc masa; Nodul 4 - Mă spăl; Nodul 5 - Plec din casă. Atunci ordinea evenimentelor dată de sortarea topologică este: Mă trezesc, mă spăl, mă îmbrac, servesc masa, plec din casă. Fie graful $G=(V,E)$, unde V este mulțimea vârfurilor și E , mulțimea arcelor. Notăm cu N cardinalul lui V și cu M , cardinalul lui E .

```

1 | Sortare_topologică_DF (G=(V, E), St[N]) //complexitate O(N+M)
2 | nr ← 0; //nr de vârfuri extrase din stiva
3 | Use ← False
4 | pentru nod ← 1, N executa
5 |   daca Not(Use[nod]) atunci Parcurge_df(nod)
6 |
7 |   pentru i ← N, 1, -1 executa scrie St[i]
8 |
9 | Parcurge_df(nod)
10|   Use[nod] ← TRUE; i ← prim_vecin(nod);
11|   cat_timp lista_vecinilor(nod) ≠ φ executa
12|     daca not(Use[i]) atunci parcurge_df(i);
13|     i ← urmator_vecin(nod);
14|
15|   nr ← nr+1; St[nr] ← nod;
16|
17|
  
```

Implementarea în limbaj a subprogramului ce realizează sortarea topologică a unui digraf aciclic, prezentată în continuare, ia în considerare următoarele declarații:

```
#include <vector>
#include <stdio.h>
#include <string.h>
#include <algorithm>
#define pb push_back

using namespace std;
vector< int > L[50005], C;
vector <int> :: iterator it;
int n, m, i, x, y ; bool sel[50005];

#include <stdio.h>
#define MAX_N 1001
struct lista
{
    int nod;
    lista *urm;
} *G[MAX_N];
int N, M, ST[MAX_N], nst;
char U[MAX_N];
```

Ca și în pseudocod, s-a preferat implementarea în manieră recursivă, iar graful este considerat a fi memorat cu ajutorul listelor de adiacență

<pre> 1 void load(){ 2 scanf("%d %d\n", &n, &m); 3 for (i=1; i<= m; i++){ 4 scanf("%d %d\n", &x, &y); 5 L[x].pb(y); 6 } 7 } 8 9 void dfs(int x){ 10 vector <int> :: iterator it; 11 sel[x]=true; 12 for(it=L[x].begin();it!=L[x].end();it++) 13 if (!sel[*it]) dfs(*it); 14 C.pb(x); 15 } 16 17 int main(){ 18 ... 19 load(); 20 memset(sel, false, sizeof(sel)); 21 for (i=1;i<=n; i++) 22 if (!sel[i]) dfs(i); 23 24 reverse(C.begin(),C.end()); 25 26 for(it=C.begin(); it!=C.end(); it++) 27 printf("%d ",*it); 28 printf("\n"); 29 }</pre>	<pre> void DF(int nod) { lista *p; U[nod] = 1; for (p = G[nod]; p != NULL; p = p->urm) if (!U[p->nod]) DF(p->nod); ST[nst++] = nod; } int main(void) { int i; citeste_graf(); for (i = 1; i <= N; i++) if (!U[i]) DF(i); for (i = N-1; i >= 0; i--) printf("%d ", ST[i]); return 0; }</pre>
--	--

O altă modalitate de implementare a sortării topologice ține cont de observația că, la un moment dat, un eveniment poate fi executat, dacă nu există nici un alt eveniment de care acesta depinde, care să nu fi fost executat anterior.

Revenind la modelarea pe digrafuri, gradul interior al unui vârf reprezintă tocmai numărul de evenimente (vârfuri) de care acesta depinde.

Înțial, în graf trebuie identificate vârfurile cu gradul interior nul, ele fiind plasate primele într-o coadă care va reține, la finalul algoritmului, ordinea vârfurilor date de sortarea topologică. Vom parcurge graful în lățime, pornind din primul vârf plasat în coadă. La fiecare pas, vom decrementa gradele tuturor vârfurilor adiacente spre interior cu acestea. În coadă vor fi adăugate doar vârfurile vecine cu cel curent, neselectate anterior și care au gradul interior nul. Algoritmul se încheie când toate vârfurile au fost plasate în coadă.

Pentru o mai bună înțelegere, să privim exemplul următor în care vectorul $Deg(N)$ reține gradele interioare ale vârfurilor, iar coada $C(N)$ va memora vârfurile în ordinea indusă de sortarea topologică:

Inițial vectorii conțin:

$$Deg=(1,0,2,1,2)$$

$$C=(2)$$

a) Se parcurge în lățime din nodul 2

$$Deg=(0,0,2,0,2)$$

$$C=(2,4,1)$$

b) Se parcurge în lățime din nodul 4

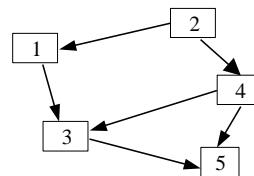
$$Deg=(0,0,1,0,1)$$

$$C=(2,4,1)$$

d) Se parcurge în lățime din nodul 1

$$Deg=(0,0,0,0,1)$$

$$C=(2,4,1,3)$$



e) Se parcurge în lățime din nodul 3

$$Deg=(0,0,0,0,0)$$

$$C=(2,4,1,3,5)$$

Implementarea în limbaj a subprogramului ce realizează sortarea topologică a unui digraf aciclic, folosind parcurgerea *BF*, prezentată în continuare, ia în considerare următoarele declarații:

```

#include <vector>
#include <stdio.h>
#include <cstring>
#include <queue>
#include <algorithm>
#define pb push_back

using namespace std;
vector <int> G[50005];
queue <int> Q;
vector <int> :: iterator it;
int N, M, i, x, y, deg[50005];
bool U[50005];
  
```

```

#include <stdio.h>
#define MAX_N 10001
struct lista
{
    int nod;
    lista *urm;
} *G[MAX_N];

int N, M, C[MAX_N], deg[MAX_N];
char U[MAX_N];
  
```

Graful este considerat a fi memorat cu ajutorul listelor de adiacență.

```

1 void load(){
2     scanf("%d %d\n", &N, &M);
3     for (i=1; i<= M; i++){
4         scanf("%d %d\n", &x, &y);
5         G[x].pb(y);
6         deg[y]++;
7     }
8 }
9
10 void sort_BF() {
11     vector<int> :: iterator it;
12     int i, x;
13
14     for (i = 1; i <= N; i++)
15         if (deg[i] == 0){
16             Q.push(i); U[i]=true;
17         }
18
19     while (!Q.empty()) {
20         x=Q.front();
21         printf("%d ", x);
22         for(it=G[x].begin();it!= G[x].end(); it++)
23         {
24             deg[*it]--;
25             if (!U[*it] && deg[*it] == 0){
26                 Q.push(*it);
27                 U[*it] = true;
28             }
29         }
30         Q.pop();
31     }
  
```

```

void sort_BF(void)
{lista *p;
int i, st = 0, dr = -1;
for (i = 1; i <= N; i++)
    if (deg[i] == 0){
        C[++dr] = i;
        U[i]=1;
    }
for (; st <= dr; st++){
    p=G[C[st]];
    while (p != NULL) {
        deg[p->nod]--;
        if (!U[p->nod] &&
            deg[p->nod] == 0){
            C[++dr] = p->nod;
            U[p->nod] = 1;
        }
        p = p->urm;
    }
}
}
  
```

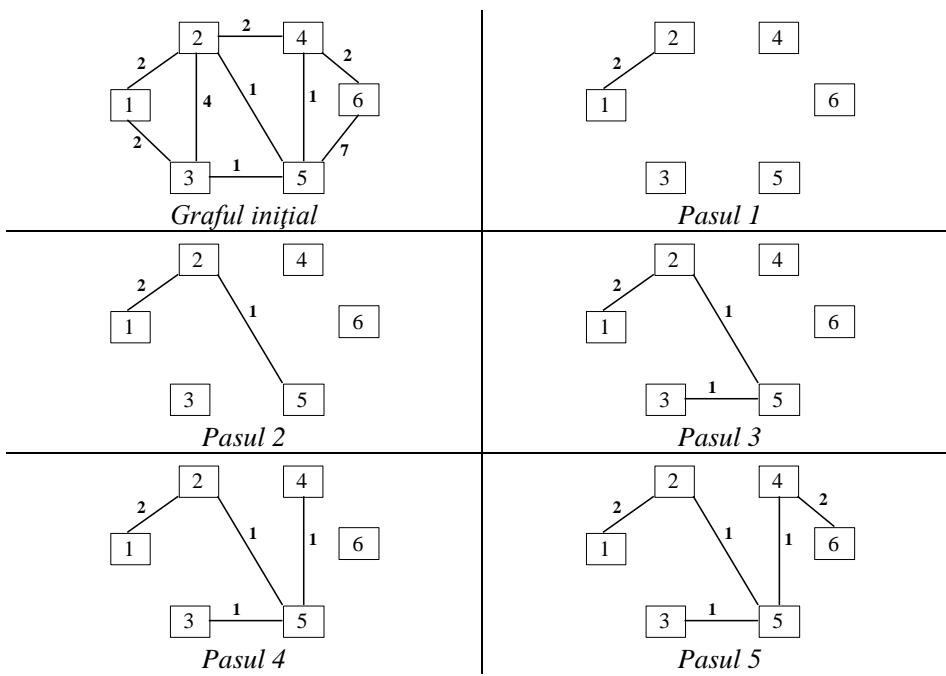
8. Arbore parțial de cost minim (A.P.M) – Algoritmul lui Prim

Fie $G=(V,E)$ un graf neorientat conex, cu costuri asociate muchiilor. Un arbore parțial al lui G este un graf parțial conex și fără cicluri. Realizați un program care determină un arbore parțial de cost minim, adică un arbore parțial pentru care suma costurilor tuturor muchiilor sale este minimă.

Soluție:

Algoritmul lui Prim construiește arborele parțial de cost minim, pornind de la un nod oarecare considerat rădăcină. La fiecare pas al algoritmului, la arbore se va adăuga un nou vârf. Acesta are proprietatea că este conectat de unul dintre vârfurile prezente deja în arbore, printr-o muchie de cost minim.

Să privim modul de construcție al A.P.M. cu ajutorul algoritmului lui *Prim* pe graful următor, considerând ca rădăcină nodul 1:



S-a obținut un A.P.M plecând de la nodul 1, costul acestuia fiind 7.

Transcrierea în pseudocod a algoritmului folosește următoarele structuri de date:

- tabloul D , în care elementul $D[x]$ va reține costul minim prin care putem „lega” nodul x , neconectat la arbore, de orice nod al arborelui.
- tabloul T , în care elementul $T[x]$ va reține nodul din arbore de care nodul x a fost conectat cu costul $D[x]$.
- Lista Q conține pe tot parcursul algoritmului nodurile grafului G neconectate la arbore.

```

1  Prim (G=(V,E,cost), rad)           //complexitate O(N*N)
2    Q ← V;                         //lista Q contine initial toate nodurile din G
3    D ← ∞;
4    D[rad] ← 0; T[rad] = 0;
5    cat timp (Q ≠ ∅) executa
6      x ← minim (Q) ;
7      Q ⇒ {x}
8      pentru fiecare y ∈ Q, adiacent cu x executa
9        daca (cost[x,y] < d[y] ) atunci
10          T[y] = x;
11          D[x] = cost[x,y];
12
13
14

```

Implementarea în limbaj a algoritmului lui *Prim*, prezentată în continuare, ia în considerare următoarele declarații:

```
#include <stdio.h> //O(N*N)
#include <string.h>
#define MAXN 101
#define INF 3000000
int N, M, R, C[MAXN][MAXN], D[MAXN],
T[MAXN], cost,
bool U[MAXN];
```

Graful este memorat cu ajutorul matricei costurilor. Parametrul x indică nodul desemnat ca rădăcină a arborelui parțial de cost minim. Lista Q a fost codificată cu ajutorul vectorului Use , astfel $Use[i]=true$, dacă nodul i aparține arborelui și $false$, în caz contrar.

Vectorul D s-a inițializat cu valorile plasate pe linia x a matricei ponderilor deoarece, la prima iterație a algoritmului, arcele minime prin care un vârf din graf poate fi conectat la rădăcina x sunt chiar costurile arcelor ce pleacă din x .

```
1 void prim(int x) {
2     int i, min, nod;
3
4     memset(U, false, sizeof(U));
5     memset(T, 0, sizeof(T));
6
7     for (i = 1; i <= N; i++)
8         D[i] = C[x][i], T[i] = x;
9     U[x] = 1;
10
11    while (1)
12    {
13        min = INF; nod = -1;
14        for (i = 1; i <= N; i++)
15            if (!U[i] && min > D[i])
16                min = D[i], nod = i;
17
18        if (min == INF) break;
19
20        U[nod] = 1;
21        cost += D[nod];
22        printf("%d %d\n", T[nod], nod);
23
24        for (i = 1; i <= N; i++)
25            if (D[i] > C[nod][i])
26            {
27                D[i] = C[nod][i];
28                T[i] = nod;
29            }
30    } printf("%d\n", cost);
31 }
```

9. Arbore parțial de cost minim (A.P.M) – Algoritmul lui Kruskal

Fie $G=(V,E)$ un graf neorientat conex, cu costuri asociate muchiilor. Un arbore parțial al lui G este un graf parțial conex și fără cicluri. Realizați un program care determină un arbore parțial de cost minim, adică un arbore parțial pentru care suma costurilor tuturor muchiilor sale este minimă.

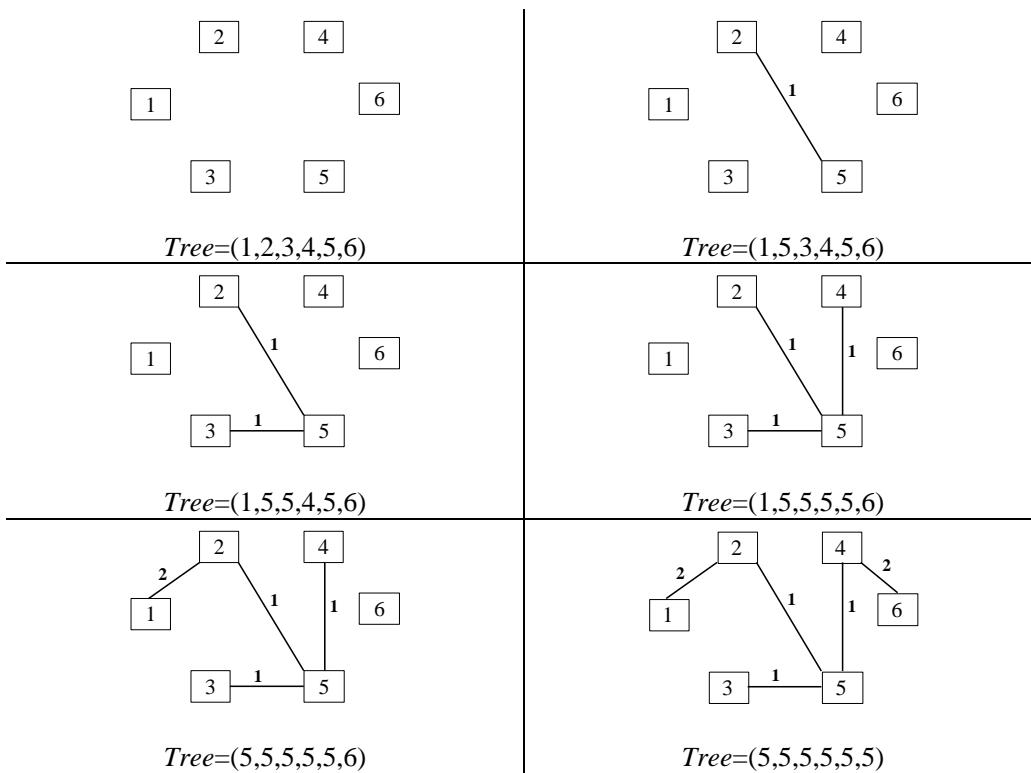
Soluție:

Considerăm N cardinalul mulțimii V și M cardinalul mulțimii E . Algoritmul lui Kruskal construiește arborele parțial de cost minim, pornind de la N arbori disjuncți, notați $T_1, T_2 \dots T_N$. Fiecare vârf al grafului definește la momentul inițial, câte un arbore. La fiecare pas al algoritmului vor fi aleși doi arbori care se vor unifica. În finalul algoritmului se va obține un singur arbore care va constitui un APM al grafului G . Proprietatea de *acoperire minimă* a arborelui rezultat este dată de modul de alegere a celor doi arbori care vor fi unificați la fiecare etapă a algoritmului. Regula respectată este următoarea:

Se identifică muchia de cost minim, nefolosită anterior, și care are vârfurile extreme în doi arbori disjuncți. În felul acesta, prin unirea celor doi arbori va rezulta tot un arbore (nu se vor crea cicluri).

Să privim modul de construcție al A.P.M. cu ajutorul algoritmului lui *Kruskal* pe graful luat ca exemplu la prezentarea algoritmului lui *Prim*:

Vectorul *Tree* codifică arborii disjuncți astfel $\text{Tree}[x]=y$ semnifică faptul că vârful x se află în arborele cu numărul de ordine y .



S-a obținut un A.P.M plecând de la nodul 1, costul acestuia fiind 7.

În transcrierea algoritmului în pseudocod, mulțimea A desemnează muchiile arborelui de acoperire minimă. Subprogramul *reuneste*(x, y) realizează unificarea arborilor disjuncți în care se regăsesc nodurile x și y .

```

1 | Kruskal (G=(V,E,cost))           //complexitate O(N*M)
2 |   A ← Ø;                         //lista muchilor din A.P.M
3 |   pentru orice varf i din V execută
4 |     Tree[i] ← i
5 |
6 |   Sortare a listei muchiilor ∈ E, crescator după cost
7 |   pentru fiecare (x,y) ∈ E execută
8 |     dacă Tree[x]≠Tree[y] atunci
9 |       A U {(x,y)};    reuneste(x,y);
10 |
11 |
12 |   returnează A
13 |

```

Implementarea în limbaj a algoritmului lui *Kruskal*, prezentată în continuare, ia în considerare următoarele declarații:

```

#define pb push_back // O(MlogN + MlogM)
#define f first      //utilizand paduri disjuncte
#define s second
#define mp make_pair

using namespace std;

int N, M, T[200010], x, y, c, i;
vector<pair<int, int>> Sol;
vector<pair<int, pair<int, int>>> L;

```

```

#include <stdio.h> //O(M*N)
#include <stdlib.h>
#include <string.h>
#define MAX_N 101
#define MAX_M 1001
#define INF 0x3f3f
struct muchie
{ int x, y, c; } e[MAX_M];
int N, M, T[MAX_N], cost;

```

Vectorul T are aceeași semnificație ca a vectorului $Tree$ prezentat anterior în exemplu și în pseudocod.

<pre> 1 int Rad(int nod) 2 { 3 if (T[nod] != nod) 4 T[nod] = Rad(T[nod]); 5 return T[nod]; 6 } 7 8 int main() 9 { 10 . . . 11 12 scanf("%d %d", &N, &M); 13 for (i = 0; i < M; i++){ 14 int x, y, c; 15 scanf("%d %d %d", &x, &y, &c); 16 L.pb(mp(c, mp(x, y))); 17 } 18 19 sort(L.begin(), L.end()); 20 21 for (int i = 0; i <= N; i++) 22 T[i]=i; 23 24 int Cost = 0; 25 26 for (i = 0; i < M; i++){ 27 int n1 = L[i].s.f; 28 int n2 = L[i].s.s; 29 30 if (Rad(n1) != Rad(n2)){ 31 Cost += L[i].f; 32 Sol.pb(mp(n1, n2)); 33 T[T[n2]] = T[n1]; 34 } 35 } 36 37 printf("%d\n", Cost); 38 printf("%d\n", Sol.size()); 39 for (i=0;i < Sol.size(); i++) 40 printf("%d %d\n", Sol[i].f, Sol[i].s); 41 42 return 0; 43 44 45 </pre>	<pre> void reuneste(int i, int j) { int k; i = T[i]; j = T[j]; if (i == j) return; for (k = 1; k <= N; k++) if (T[k] == i) T[k] = j; } int comp_muchie(const void *i, const void *j) { return ((int *) i)[2] - ((int *) j)[2]; } void kruskal(void) { int i, j, k, c; qsort(e, M, sizeof(e[0]), comp_muchie); for (i=1;i<=N;i++) T[i]=i; for (k = 0; k < M; k++){ i = e[k].x; j = e[k].y; c = e[k].c; if (T[i]==T[j]) continue; reuneste(i, j); cost += c; printf("%d %d %d\n", i, j, c); } printf("Cost minim = %d\n", cost); } </pre>
---	---

10. Puncte de articulație - critice

Un nod dintr-un graf $G=(V, E)$ neorientat conex este punct de articulație (critic), dacă și numai dacă prin eliminarea lui, împreună cu muchiile incidente acestuia, se pierde proprietatea de conexitate. Realizați un program care determină mulțimea punctelor critice dintr-un graf.

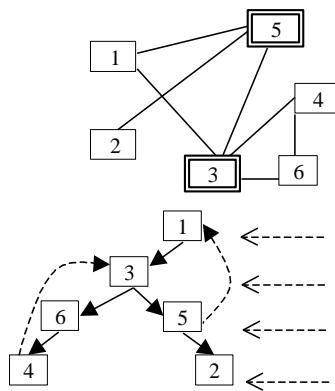
Soluție:

Determinarea mulțimii punctelor de articulație poate fi realizată printr-un algoritm liniar ($O(m+n)$). Acesta are la bază o parcurgere DF în care se rețin mai multe informații despre fiecare nod, informații care vor conduce, în final, la identificarea punctelor de articulație.

Pentru un nod $x \in V$ se vor identifica:

- numărul nivelului atins în parcurgerea DF , memorat în vectorul nv pe poziția x ($nv[x]$);
- numărul minim al nivelului care poate fi atins din x folosind descendenții săi și cel mult o muchie de întoarcere. Intuitiv este vorba de “cel mai de sus” nivel care poate fi atins din x prin intermediul muchiilor de întoarcere accesibile din el sau din descendenții lui. Acest număr va fi reținut în vectorul L pe poziția x ($L[x]$);
- vârful părinte în arborele DF , reținut în vectorul t pe poziția x ($t[x]$).

Dacă dintr-un nod x din graf nu se poate ajunge pe un nivel strict mai mic decât al tatălui său din arborele DF ($nv[t[x]] \leq L[x]$), atunci $t[x]$ este punct de articulație; eliminarea acestuia, împreună cu muchiile adiacente, ar duce la izolarea nodului x .



Considerăm graful din figura alăturată. El conține două puncte de articulație: nodul 3 și nodul 5.

Arborele DF al acestuia cu rădăcina în nodul 1 are patru nivele:

Vectorii:

$$t = (0, 5, 1, 6, 3, 3)$$

Nivel 1
Nivel 2
Nivel 3
Nivel 4

$$nv = (1, 4, 2, 4, 3, 3)$$

$$L = (1, 4, 1, 2, 1, 2)$$

$L[3]=1$ deoarece nivelurile minime atinse de descendenții săi sunt:

- nivelul 2 pentru nodul 4 ($L[4]=2$);
- nivelul 1 pentru nodul 5 ($L[5]=1$);
- nivelul 2 pentru nodul 6 ($L[6]=2$);
- nivelul 4 pentru nodul 2 ($L[2]=4$).

Nivelul minim atins din nodul 3 prin intermediul descendenților săi și al unei muchii de întoarcere este nivelul 1 ($L[3]=1$).

Cum pentru nodul 3 există descendentul direct nodul 6, care nu poate atinge un nivel mai mic decât cel pe care este situat el, rezultă că 3 este punct de articulație. Analog pentru nodul 5.

De reținut că nodul rădăcină al arborelui DF este punct de articulație dacă are mai mult de un singur descendent direct.

Implementarea în limbaj a subprogramului *df* ce identifică mulțimea punctelor critice, prezentată în continuare, ia în considerare următoarele declarații:

```

const MAX_N=1001;
type plista=^lista;
    lista=record nod:integer;
        urm:Plista;
    end;
var G:array[0..max_n]of plista;
    t,L,nv:array[0..max_n]of integer;
    rad,nr,i,n,m:integer;
    c,u:array[0..max_n]of byte;
begin
    ...  

procedure df(nod:integer);
var p:plista;
begin
    U[nod]:=1;
    L[nod]:=nv[nod];
    p:=G[nod];
    while p<>nil do begin
        if (U[p^.nod]=0) then begin
            nv[p^.nod]:= nv[nod]+1;
            T[p^.nod]:= nod;
            if (nod =rad) then inc(nr);
            DF(p^.nod);
            if (L[nod]>L[p^.nod]) then
                L[nod] := L[p^.nod];
            if (L[p^.nod]>=nv[nod]) then
                c[nod] := 1;
        end
        else
            if (p^.nod <>T[nod]) and
                (L[nod] > nv[p^.nod]) then
                    L[nod]:= nv[p^.nod];
        p:=p^.urm;
    end;
end;
begin
    citeste_graf;
    for i:=1 to n do
        if u[i]=0 then begin
            nv[i]:=1;
            rad:=i;
            nr:=0;
            DF(i);
            if nr>1 then c[rad]:=1
            else c[rad]:=0;
        end;
        for i:=1 to n do
            if c[i]<>0 then write(i,' ');
    end.

```

```

#include <stdio.h>
#define MAX_N 1001
struct lista
{
    int nod;
    lista *urm;
} *G[MAX_N];

int N, M, T[MAX_N], L[MAX_N],
      nv[MAX_N], rad, nr;
char U[MAX_N], c[MAX_N];

```

Vectorul *C* va reține pentru fiecare nod, valoarea 0 dacă nodul este critic și 1, în caz contrar. Vectorul *U* codifică, în timpul parcurgerii *DF*, starea unui nod: vizitat sau nevizitat.

Variabila *nr* contorizează numărul de descendenți ai nodului considerat rădăcină în parcurgerea *DF*.

Graful *G* se consideră a fi memorat cu ajutorul listelor de adiacență.

<pre> 1 ... 2 procedure df(nod:integer); 3 var p:plista; 4 begin 5 U[nod]:=1; 6 L[nod]:=nv[nod]; 7 p:=G[nod]; 8 while p<^{>}nil do begin 9 if (U[p^.nod]=0) then begin 10 nv[p^.nod]:= nv[nod]+1; 11 T[p^.nod]:= nod; 12 if (nod =rad) then inc(nr); 13 DF(p^.nod); 14 if (L[nod]>L[p^.nod]) then 15 L[nod] := L[p^.nod]; 16 if (L[p^.nod]>=nv[nod]) then 17 c[nod] := 1; 18 end 19 else 20 if (p^.nod <^{>}T[nod]) and 21 (L[nod] > nv[p^.nod]) then 22 L[nod]:= nv[p^.nod]; 23 p:=p^.urm; 24 end; 25 end; 26 27 begin 28 citeste_graf; 29 for i:=1 to n do 30 if u[i]=0 then begin 31 nv[i]:=1; 32 rad:=i; 33 nr:=0; 34 DF(i); 35 if nr>1 then c[rad]:=1 36 else c[rad]:=0; 37 end; 38 for i:=1 to n do 39 if c[i]<>0 then write(i,' '); 40 end. </pre>	<pre> ... void DF(int nod) {lista *p; U[nod] = 1; L[nod] = nv[nod]; for (p = G[nod]; p != NULL; p = p->urm) if (!U[p->nod]) { nv[p->nod] = nv[nod]+1; T[p->nod] = nod; if (nod == rad) nr++; DF(p->nod); if (L[nod] > L[p->nod]) L[nod] = L[p->nod]; if (L[p->nod] >= nv[nod]) c[nod] = 1; } else if (p->nod != T[nod] && L[nod] > nv[p->nod]) L[nod] = nv[p->nod]; } int main(void) {int i; citeste_graf(); for (i = 1; i <= N; i++) if (!U[i]) {nv[i] = 1; rad = i; nr = 0; DF(i); c[rad] = nr > 1; } for (i = 1; i <= N; i++) if (c[i]) printf("%d ", i); return 0; } </pre>
---	---

2. Muchii critice – punți

O muchie dintr-un graf $G=(V, E)$ neorientat conex este punte (muchie critică), dacă și numai dacă, prin eliminarea sa, se pierde proprietatea de conexitate. Realizați un program care determină

mulțimea muchiilor critice dintr-un graf.

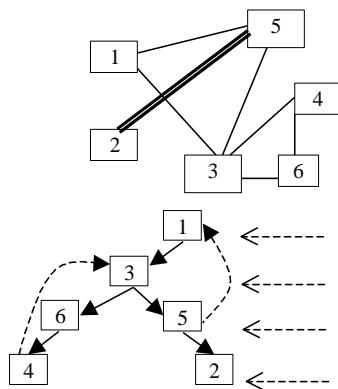
Soluție:

Determinarea mulțimii punțiilor poate fi realizată printr-un algoritm liniar ($O(m+n)$). Acesta are la bază o parcurgere DF în care se rețin mai multe informații despre fiecare nod, informații care vor conduce, în final, la identificarea punțiilor.

Pentru un nod $x \in V$ se vor identifica:

- numărul nivelului atins în parcurgerea DF , memorat în vectorul nv pe poziția x ($nv[x]$);
- numărul minim al nivelului care poate fi atins din x folosind descendenții săi și cel mult o muchie de întoarcere. Intuitiv este vorba de “cel mai de sus” nivel care poate fi atins din x prin intermediul muchiilor de întoarcere accesibile din el sau din descendenții lui. Acum număr va fi reținut în vectorul L pe poziția x ($L[x]$);
- vârful părinte în arborele DF , reținut în vectorul t pe poziția x ($t[x]$).

O observație necesară este că muchiile de întoarcere din arborele DF nu pot fi muchii critice, deoarece acestea încid un ciclu, iar eliminarea lor nu strică proprietatea de conexitate. Așadar, singurele muchii care vor fi verificate sunt muchiile de arbore. Dacă dintr-un nod x din graf nu se poate ajunge pe un nivel mai mic sau egal decât al tatălui său din arborele DF ($nv[t[x]] < L[x]$), atunci muchia ($t[x], x$) este critică.



Considerăm graful din figura alăturată.
El conține o muchie critică între nodurile 2 și 5.

Arborele DF al acestuia cu rădăcina în nodul 1 are patru nivele:

Nivel 1	Vectorii:
	$t = (0, 5, 1, 6, 3, 3)$
Nivel 2	$nv = (1, 4, 2, 4, 3, 3)$
Nivel 3	
Nivel 4	$L = (1, 4, 1, 2, 1, 2)$

Nivelul minim atins din nodul 2 prin intermediul descendenților săi și al unei muchii de întoarcere este nivelul 4 ($L[2]=4$), iar nivelul predecesorului său (nodul 5) este 3 ($nv[t[2]] = 3$).

Implementarea în limbaj a subprogramului DF ce identifică mulțimea muchiilor critice, prezentată în continuare, ia în considerare următoarele declarații:

```

const MAX_N=1001;
type plista=^lista;
  lista=record
    nod:integer;c:boolean;urm:Plista;
  end;
var G:array[0..max_n]of plista;
  t,L,nv:array[0..max_n]of integer;
  i,n,m:integer; p:plista;
  u:array[0..max_n]of byte;
include <stdio.h>
#define MAX_N 1001
struct lista {
  int nod; char c;
  lista *urm;
} *G[MAX_N];
int N, M, T[MAX_N], L[MAX_N],
nv[MAX_N];
char U[MAX_N];
  
```

Vectorul U codifică, în timpul parcurgerii DF , starea unui nod: vizitat sau nevizitat.

Graful G se consideră a fi memorat cu ajutorul listelor de adiacență, iar pentru fiecare element din listele de adiacență se va memora o variabilă de tip *boolean*(Pascal)/*char* (C/C++) care va fi *true/1* dacă muchia respectivă este critică.

```

1   ...
2   procedure df(nod:integer);
3   var p:plista;
4   begin
5     U[nod]:=1;
6     L[nod]:=nv[nod];
7     p:=G[nod];
8     while p<>nil do begin
9       if (U[p^.nod]=0) then begin
10         nv[p^.nod]:= nv[nod]+1;
11         T[p^.nod]:= nod;
12         DF(p^.nod);
13         if (L[nod]>L[p^.nod]) then
14           L[nod]:= L[p^.nod];
15         if (L[p^.nod]>nv[nod])then
16           p^.c := true;
17       end
18     else
19       if (p^.nod <>T[nod])and
20         (L[nod] > nv[p^.nod])then
21           L[nod]:= nv[p^.nod];
22     p:=p^.urm;
23   end;
24 end;
25
26 begin
27   citeste_graf;
28   for i:=1 to n do
29     if u[i]=0 then
30       begin
31         nv[i]:=1;
32         DF(i);
33       end;
34   for i:=1 to n do
35     begin
36       p:=G[i];
37       while p<>nil do
38         begin
39           if p^.c then
40             write(i,' ',p^.nod);
41             p:=p^.urm;
42           end;
43         end;
44   end.
45

```

```

...
void DF(int nod)
{lista *p;
U[nod] = 1;
L[nod] = nv[nod];
for (p = G[nod]; p != NULL;
      p = p->urm)
  if (!U[p->nod]) {
    nv[p->nod] = nv[nod]+1;
    T[p->nod] = nod;
    DF(p->nod);
    if (L[nod] > L[p->nod])
      L[nod] = L[p->nod];
    if (L[p->nod] > nv[nod])
      p->c = 1;
  }
else
  if (p->nod != T[nod] &&
      L[nod] > nv[p->nod])
    L[nod] = nv[p->nod];
}

int main(void)
{int i;
lista *p;
citeste_graf();
for (i = 1; i <= N; i++)
  if (!U[i])
    { nv[i] = 1;
      DF(i);
    }
for (i = 1; i <= N; i++)
  for (p = G[i]; p != NULL;
        p = p->urm) if (p->c)
printf("%d %d\n", i, p->nod);
return 0;
}

```

3. Componente biconexe

Prin definiție, un graf $G=(V, E)$ este biconex dacă nu conține puncte de articulație. Prin componentă biconexă se înțelege un subgraf maximal în raport cu proprietatea de biconexitate. Realizați un program care determină muchiile fiecărei componente biconexe a unui graf.

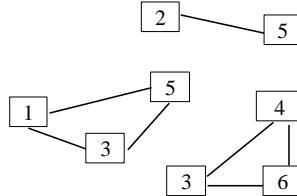
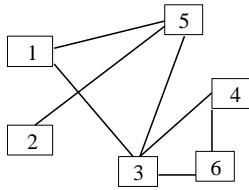
Soluție:

Algoritmul de determinare a componentelor biconexe are la bază parcurgerea DF a grafului, de aici și complexitatea liniară a acestuia ($O(m+n)$). De fapt, algoritmul este o extensie a algoritmului pentru determinarea punctelor de articulație:

- parcurgerea DF pentru determinarea numărului minim al nivelului care poate fi atins din fiecare nod folosind descendenții acestuia și cel mult o muchie de întoarcere. Aceste numere vor fi reținute în vectorul L pe poziția x ($L[x]$).
- în timpul parcurgerii DF se vor afișa muchiile din fiecare componentă biconexă. Această operație se va realiza memorându-se explicit o stivă cu muchiile parcuse. Când se determină un nod x din graf care nu poate ajunge pe un nivel strict mai mic decât al tatălui său din arborele DF ($nv[t[x]] \leq L[x]$), se va afișa o nouă componentă biconexă. Ea va fi formată din muchiile din stivă, operația de

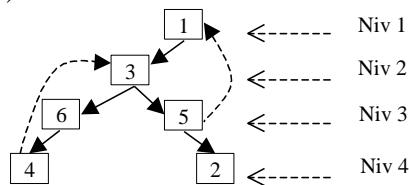
extragere din stivă se oprește la găsirea muchiei ($t[x]$, x) în vârful stivei.

Considerând ca exemplu graful următor, se vor obține trei componente biconexe:



Etapele algoritmului:

a)



După parcurgerea DF se vor obține vectorii:
 $t=(0,5,1,6,3,3)$
 $nv=(1,4,2,4,3,3)$
 $L=(1,4,1,2,1,2)$

b)

Pasul 1:

În momentul găsirii nodului 6 care nu poate ajunge mai sus, stiva conține:

$$st=((1,3),(3,6),(4,6),(3,4))$$

Se elimină muchii din stivă până la găsirea muchiei ($t[6], 6 = (3, 6)$).

La final stiva va conține:

$$st=((1,3))$$

Pasul 2:

Se găsește nodul 2 care nu poate ajunge mai sus:

$$st=((1,3),(3,5),(2,5))$$

Se afișează componenta biconexă formată din succesiunea de muchii până la muchia (2,5).

La final stiva va conține:

$$st=((1,3),(3,5))$$

Pasul 3:

Se găsește nodul 3 care nu poate ajunge mai sus:

$$st=((1,3),(3,5),(1,5))$$

Se afișează componenta biconexă formată din succesiunea de muchii până la muchia (1,3).

Pasul 4:

Stiva vidă, algoritmul ia sfârșit.

Implementarea în limbaj a subprogramului DF care identifică componente biconexe, prezentată în continuare, ia în considerare următoarele declarații:

```

const MAX_N=101; MAX_M=1001;
type plista=^lista;
    lista=record
        nod:integer; urm:plista;
    end;
var G:array[0..MAX_N]of Plista;
U,T,L,nv:array[0..MAX_N]of
integer;
st:array[0..MAX_M,0..2]of
integer;
lung,N,M:integer;
  
```

```

#include <stdio.h>
#define MAX_N 101
#define MAX_M 1001
struct lista
{ int nod;
  lista *urm; } *G[MAX_N];
int N, M, T[MAX_N], L[MAX_N],
nv[MAX_N], st[MAX_M][2], lung;
char U[MAX_N];
  
```

Vectorul U codifică, în timpul parcurgerii DF , starea unui nod: vizitat sau nevizitat. Subprogramele $push()$ și $pop()$ implementează operațiile de introducere, respectiv extragere din stivă a

unei muchii.

<pre> 1 ... 2 procedure push(i,j:integer); 3 begin 4 st[lung][0]:=i;st[lung][1]:=j; 5 inc(lung); 6 end; 7 procedure pop(var i,j:integer); 8 begin 9 dec(lung); i:=st[lung][0]; 10 j:=st[lung][1]; 11 end; 12 procedure DF(nod:integer); 13 var p:plista; x,y:integer; 14 begin 15 U[nod]:=1; 16 L[nod]:= nv[nod]; 17 p:=G[nod]; 18 while p<>nil do begin 19 if (p^.nod<>T[nod]) and 20 (nv[nod]>nv[p^.nod]) then 21 push(p^.nod,nod); 22 if U[p^.nod]=0 then begin 23 nv[p^.nod]:=nv[nod]+1; 24 T[p^.nod]:= nod; 25 DF(p^.nod); 26 if (L[nod]>L[p^.nod]) then 27 L[nod]:= L[p^.nod]; 28 if (L[p^.nod]>=nv[nod]) then 29 begin 30 repeat 31 pop(x, y); 32 write('(' ,x, ' ',y, ') '); 33 until not((x<>nod) or 34 (y<>p^.nod)) 35 or not ((x<> p^.nod) 36 or (y<>nod)); 37 writeln; 38 end; 39 end else 40 if (p^.nod<>T[nod]) and 41 (L[nod]>nv[p^.nod]) then 42 L[nod]:= nv[p^.nod]; 43 p:=p^.urm; 44 end; 45 end; </pre>	<pre> ... void push(int i, int j) { st[lung][0] = i; st[lung++][1] = j; } void pop(int *i, int *j) { *i = st[--lung][0]; *j = st[lung][1]; } void DF(int nod) { lista *p; int x, y; U[nod] = 1; L[nod] = nv[nod]; for (p = G[nod]; p != NULL; p = p->urm) { if (p->nod != T[nod] && nv[nod] > nv[p->nod]) push(p->nod, nod); if (!U[p->nod]) { nv[p->nod] = nv[nod]+1; T[p->nod] = nod; DF(p->nod); if (L[nod] > L[p->nod]) L[nod] = L[p->nod]; if (L[p->nod] >= nv[nod]) do { pop(&x, &y); printf("(%d %d ", x, y); } while ((x != nod y != p->nod) && && (x != p->nod y != nod)); printf("\n"); } else { if (p->nod != T[nod] && L[nod] > nv[p->nod]) L[nod] = nv[p->nod]; } } } </pre>
---	---

Noțiuni introductive

Un **graf** (neorientat sau orientat) este o pereche ordonată de mulțimi $G = (V, E)$.

Mulțimea V este o mulțime nevidă și finită de elemente denumite **vârfurile** grafului.

Mulțimea E este o mulțime de perechi de vârfuri din graf.

În cazul grafurilor neorientate, perechile de vârfuri din mulțimea E sunt neordonate și sunt denumite **muchii**. Perechea neordonată formată din vârfurile x și y se notează $[x, y]$; vârfurile x și y se numesc **extremitățile muchiei** $[x, y]$.

În cazul grafurilor orientate, perechile de vârfuri din mulțimea E sunt ordonate și sunt denumite **arce**. Perechea ordonată formată din vârfurile x și y se notează (x, y) ; vârful x se numește **extremitate inițială** a arcului (x, y) , iar vârful y se numește **extremitate finală** a arcului (x, y) .

Dacă există un arc sau o muchie cu extremitățile x și y , atunci vârfurile x și y sunt **adiacente**; fiecare extremitate a unei muchii/unui arc este considerată **incidentă** cu muchia/arcul respectiv.

Vom considera că extremitățile unei muchii, respectiv ale unui arc, sunt distincte (adică graful nu conține bucle).

Observații

1. Cu ajutorul unui graf neorientat putem modela ***o relație simetrică*** între elementele unei mulțimi, în timp ce cu ajutorul unui graf orientat modelăm ***o relație care nu este simetrică***.
2. Între oricare două vârfuri ale unui graf poate exista cel mult o muchie/arc. Dacă între două vârfuri există mai multe muchii/arce atunci structura se numește **multigraf**. Nu vom lucra cu structuri multigraf.
3. În practică, informațiile asociate unui graf pot fi oricât de complexe, dar, pentru a simplifica, vom considera că vârfurile grafului sunt etichetate cu numere naturale de la 1 la n (unde cu n vom nota numărul de vârfuri din graf). Această numerotare nu este o restrângere a generalității (de exemplu, numărul vârfului poate fi considerat poziția pe care sunt memorate într-un vector informațiile asociate vârfului).
4. În unele lucrări de specialitate, un vârf al grafului se numește **nod**.

1. Reprezentare vizuală a grafurilor

- Fiecarui vârf din graf îi corespunde un punct în plan care are asociat numărul vârfului. Pentru o mai mare lizibilitate, un vârf se reprezintă ca un cerc sau pătrat în interiorul căruia se specifică numărul vârfului.
- Dacă graful este neorientat, vom reprezenta fiecare muchie ca o linie (dreaptă sau curbă), care unește cele două extremități ale muchiei.
- Dacă graful este orientat, vom reprezenta fiecare arc ca o săgeată (dreaptă sau curbă) dinspre extremitatea inițială către extremitatea finală a arcului.

Graf neorientat $G = (V, E)$.

$$V = \{1, 2, 3, 4, 5, 6\},$$

$$E = \{(1,2), (1,3), (1,5)(2,5), (4,5)\}$$

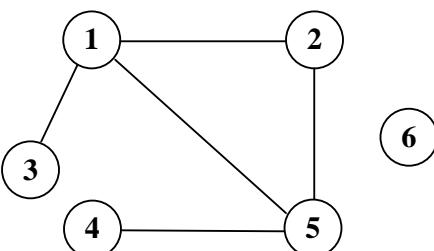


Figura 1
Graf neorientat

Graf orientat $G = (V, E)$.

$$V = \{1, 2, 3, 4, 5, 6\},$$

$$E = \{(1,2), (1,3), (2,5), (4,5), (5,1)(5,4)\}$$

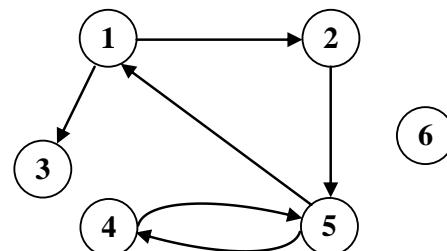


Figura 2
Graf orientat

2. Gradul unui vârf

Fie $G = (V, E)$ un *graf neorientat*. Se numește **grad** al unui vârf x numărul de muchii incidente cu vârful respectiv. Gradul vârfului x se notează $d(x)$.

Se numește **vârf izolat** un vârf care are grad 0.

Se numește **vârf terminal** un vârf cu gradul 1.

De exemplu, pentru graful din figura 1:

x	1	2	3	4	5	6
$d(x)$	3	2	1	1	3	0

Vârfuri izolate: 6

Vârfuri terminale: 3, 4

Fie $G = (V, E)$ un *graf orientat* și x un vârf din graf.

Gradul exterior al vârfului x se notează $d^+(x)$ și este egal cu numărul de arce care au ca extremitate inițială pe x .

Gradul interior al vârfului x se notează $d^-(x)$ și este egal cu numărul de arce care au ca extremitate finală pe x .

De exemplu, pentru graful din figura 2:

x	1	2	3	4	5	6
$d^+(x)$	2	1	0	1	2	0
$d^-(x)$	1	1	1	1	2	0

Observații

1. Suma gradelor unui graf neorientat este egală cu dublul numărului de muchii din graf.
2. Suma gradelor interioare ale vârfurilor unui graf orientat este egală cu suma gradelor exterioare ale vârfurilor grafului și este egală cu numărul de arce din graf.

3. Lanț, ciclu, drum, circuit

Se numește **lanț** într-un graf neorientat, o secvență de vârfuri $[x_1, x_2, \dots, x_p]$, cu proprietatea că oricare două vârfuri consecutive din secvență sunt adiacente.

Un lanț este **elementar** dacă el nu conține de mai multe ori același vârf.

Un lanț este **simplu** dacă el nu conține de mai multe ori aceeași muchie.

Se numește **ciclu** un lanț simplu pentru care extremitatea inițială coincide cu extremitatea finală. Ciclul se numește **elementar** dacă nu conține de mai multe ori același vârf (exceptând extremitățile sale).

Se numește **lungime a unui lanț** numărul de muchii conținute.

De exemplu, pentru graful neorientat din figura 1:

Lanț: [3,1,2,5,1,3,1,5,4] - lungime 8.

Lanț elementar: [3,1,2,5] - lungime 3, [4,5,2,1,3] - lungime 4.

Lanț simplu: [3,1,2,5,1], [4,5,1].

Ciclu: [1,2,5,1,2,5,1].

Ciclu elementar: [1,2,5,1].

Se numește **drum** într-un graf orientat o secvență de vârfuri (x_1, x_2, \dots, x_p) , astfel încât pentru oricare două vârfuri consecutive x_i și x_{i+1} există arcul (x_i, x_{i+1}) .

Drumul se numește **elementar** dacă nu conține de mai multe ori același vârf.

Drumul se numește **simplu** dacă nu conține de mai multe ori același vârf.

Se numește **circuit** un drum simplu pentru care extremitatea inițială coincide cu extremitatea finală. Circuitul se numește **elementar** dacă nu conține de mai multe ori același vârf (exceptând extremitățile sale).

Se numește **lungime a unui drum** numărul de arce conținute.

De exemplu, pentru graful orientat din figura 2:

Drum: $(1,2,5,1,3)$ – lungime 4.

Drum elementar: $(4,5,1,2)$ – lungime 3.

Drum simplu: $(5,4,5,1,2,5), (1,2)$.

Circuit: $(4,5,4,5,4), (1,2,5,1)$.

Circuit elementar: $(1,2,5,1)$.

Un lanț/drum/ciclu/circuit elementar se numește **hamiltonian** dacă el trece prin toate vâfurile grafului.

Un lanț/drum/ciclu/circuit elementar se numește **eulerian** dacă el trece prin fiecare muchie/arc a/al grafului o singură dată.

4. Grafuri asociate unui graf dat

Fie $G = (V, E)$ un graf *orientat* sau *neorientat*.

Graful $G' = (V, E')$ se numește **graf parțial** al lui G dacă $E' \subset E$.

Observație: Un graf parțial al lui G se obține eliminând muchii/arce din graful G .

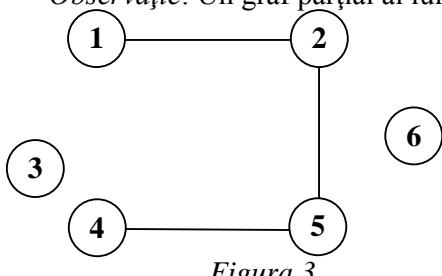


Figura 3

Graf parțial obținut din graful din figura 1 prin eliminarea muchiilor $[1,3], [1,5]$.

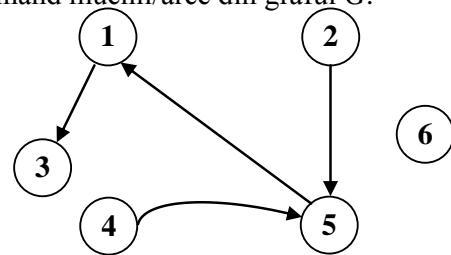


Figura 4

Graf parțial obținut din graful din figura 2 prin eliminarea arcelor $(1,2), (5,4)$.

Graful $G'' = (V'', E'')$ se numește **subgraf** al lui G dacă $V'' \subset V$, iar E'' este mulțimea tuturor muchiilor/arcelor din E cu proprietatea că au ambele extremități în V'' .

Se spune că subgraful G'' este induș (sau generat) de mulțimea de vârfuri V'' .

Observație: Un subgraf al lui G se obține eliminând vârfuri din graful G împreună cu toate muchiile/arcele incidente cu acestea.

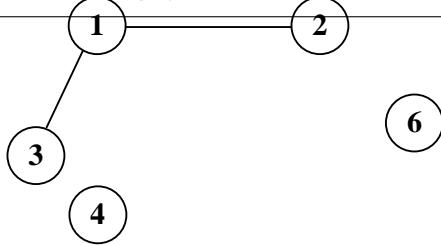


Figura 5

Subgraf obținut din graful din figura 1 prin eliminarea vârfului 5 și a tuturor muchiilor incident cu acesta: [1,5], [2,5], [4,5].

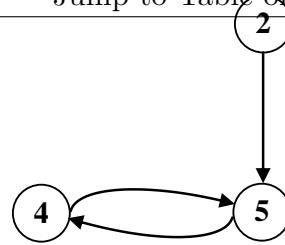


Figura 6

Subgraf obținut din graful din figura 2 prin eliminarea vârfurilor 1, 3 și 6 și a tuturor arcelor incidente cu acestea: (1,2), (1,3), (5,1).

Graful $G'' = (V'', E'')$ se numește **subgraf parțial** al lui G dacă $V'' \subset V$, iar E'' este inclusă în mulțimea tuturor muchiilor/arcelor din E cu proprietatea că au extremitățile în V'' .

Observație: Un subgraf parțial al lui G se obține eliminând vârfuri din graful G , toate muchiile/arcele incidente cu vârfurile eliminate, precum și alte muchii/arce din graf.

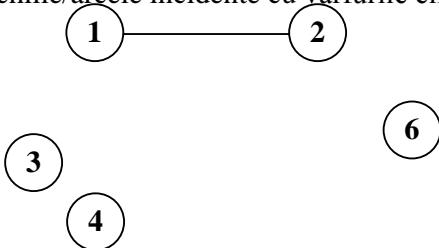


Figura 7

Subgraf parțial obținut din graful din figura 1 prin eliminarea vârfului 5 și a muchiilor [1,3], [1,5], [2,5], [4,5].

Exerciții

Fie G un graf orientat sau neorientat cu n vârfuri și m muchii/arce.

1. Numărul de grafuri parțiale ale lui G este 2^m .

Soluție: Numerotăm muchiile grafului de la 1 la m . Fiecare graf parțial îi putem asocia în mod biunivoc o funcție $f: \{1, 2, \dots, m\} \rightarrow \{0, 1\}$ astfel:

$$f(i) = \begin{cases} 1, & \text{dacă muchia numerotată } i \text{ aparține grafului parțial} \\ 0, & \text{dacă muchia numerotată } i \text{ nu aparține grafului parțial} \end{cases}$$

Numărul de grafuri parțiale este egal cu numărul de funcții definite, adică 2^m (considerăm că un graf este parțial al său).

2. Numărul de subgrafuri ale lui G este $2^n - 1$.

Soluție: Pentru a genera un subgraf, trebuie să selectăm mulțimea vârfurilor sale, mulțimea muchiilor/arcelor fiind unic determinată de mulțimea vârfurilor selectate. Mulțimea $\{1, 2, \dots, n\}$ are 2^n submulțimi, dintre care trebuie să eliminăm mulțimea vidă. Deci, există $2^n - 1$ subgrafuri ale unui graf cu n vârfuri (considerăm că un graf este subgraf al său).

Fie $G = (V, E)$ un graf *orientat*.

Graful $G^T = (V, E^T)$ se numește **graf transpus** al grafului G dacă $E^T = \{(y, x) | (x, y) \in E\}$.

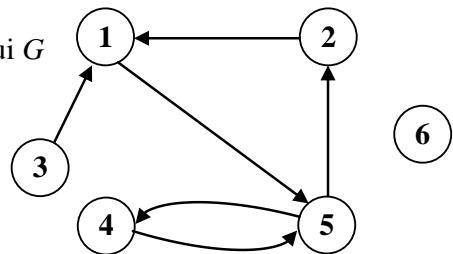


Figura 9

Graful transpus al grafului orientat din figura 2

5. Tipuri speciale de grafuri

Graf complet

Un graf *orientat* sau *neorientat* se numește **complet** dacă oricare două vârfuri din graf sunt adiacente.

Observație: Graful neorientat complet cu n vârfuri se notează K_n și conține $\frac{n(n-1)}{2}$ muchii.

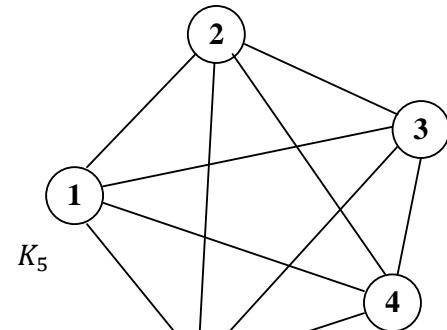
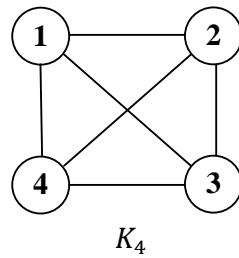
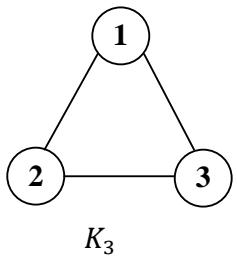


Figura 10

Grafuluri neorientate complete cu 3, 4 și 5 vârfuri

Pentru un număr de vârfuri fixat, grafuri neorientate complete sunt unic, dar grafurile orientate complete sunt mai multe.

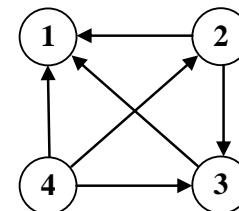
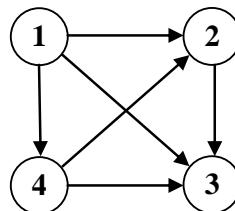


Figura 11

Grafuluri orientate complete cu 4 vârfuri

Graf antisimetric

Un graf *orientat* se numește **antisimetric** dacă pentru oricare două vârfuri din graf x și y dacă există arcul (x, y) , atunci nu există arcul (y, x) .

Observație: Orice relație de ordine între elementele unei mulțimi poate fi modelată cu ajutorul unui graf orientat antisimetric (vâfurile grafului corespund elementelor mulțimii; dacă elementul x este în relația de ordine respectivă cu elementul y , atunci în graf va exista arcul (x, y) ; graful astfel definit este antisimetric, deoarece relația de ordine este antisimetrică).

Graful din figura 9 nu este antisimetric deoarece există vâfurile 4 și 5 pentru care avem arcele $(4,5)$ și $(5,4)$.

Graf turneu

Un graf *orientat* complet și antisimetric se numește graf **turneu**.

Graf bipartit

Un graf neorientat $G = (V, E)$ se numește **bipartit** dacă mulțimea vârfurilor sale poate fi partionată în două submulțimi A și B nesecare ($A \cup B = V, A \cap B = \emptyset$) astfel încât orice muchie are o extremitate în A și una în B .

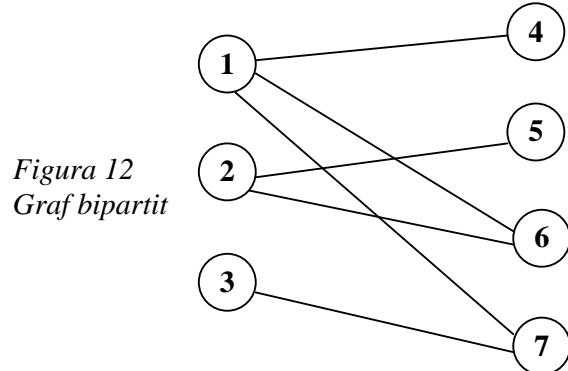


Figura 12
Graf bipartit

Graf bipartit complet

Un graf bipartit se numește **complet** dacă fiecare vîrf din mulțimea A este adiacent cu fiecare vîrf din mulțimea B .

Observație: Dacă numărul de vârfuri din mulțimea A este p , iar numărul de vârfuri din mulțimea B este q , graful bipartit complet se notează K_{p-q} și conține $p - q$ muchii.

Graf regulat

Un graf neorientat se numește **regulat** dacă toate vârfurile sale au același grad.

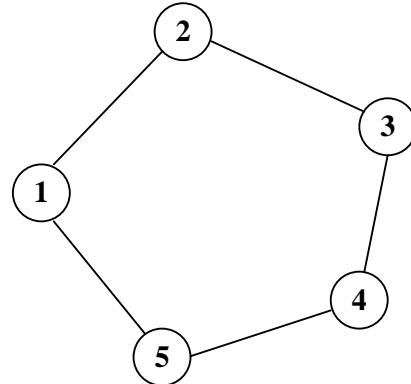


Figura 13
Graf regulat

15 ALGORITMI

BFS	DFS	BKT
<ul style="list-style-type: none"> • verificare graf conex • deb + ofig coresp conex • lungimea min este lsf • grafuri bipartite • diametru minim • algoritmul lui lee 	<ul style="list-style-type: none"> • sortare topologică • graf acyclic • clase de echivalență 	<ul style="list-style-type: none"> • graf binarizat

15.1 BFS

bfs - breadth first search

Initial, $\forall v \in V$ are eticheta $label(v) < 0$.

```

 $label(s) \leftarrow 0$      $parent(s) \leftarrow 0;$ 
creează coada  $Q$  conținând  $s$ ;
while  $Q \neq \emptyset$  do
{
     $v \leftarrow$  vârful din capul cozii  $Q$ ;
    șterge vârful din capul cozii  $Q$ ;
    for  $w \in A(v)$  do
    {
        if  $label(w) < 0$  then
        {
             $label(w) \leftarrow label(v) + 1$ ;
             $parent(w) \leftarrow v$ ;
            introdu  $w$  în coada  $Q$ 
        }
    }
}
```

15.2 BFS - liste alocate dinamic

```

void BFS (int k, int x)
{
    int p, u, i;
    nod *q;
    p = u = k;
    c[p] = x;
    viz[x] = 1;
    viz[u] = qv[x];
    while (p <= u)
        x = c[p + 1];
        if (q = priu[x], q != 0)
            u = q;
            viz[q] = -1;
            c[t + u] = u;
            viz[u] = qv[u];
            viz[u] += qv[u];
}

```

15.3 BFS - liste alocate static

```

BFS - LISTE ALOCATE STATIC

void BFS (int x, int k)
{
    int c[100], p, u, i, v;
    p = u = 1;
    c[p] = x;
    vj[x] = 1;
    if (uj[x][0] % 2 == 0) uj[k] ++;

    while (p <= u)
    {
        x = c[p++];
        for (i = 1; i <= u; x = a[x][i]; i++)
        {
            v = a[x][i];
            if (uj[v] == -1)
                c[++u] = v;
            vj[v] = k;
            if (uj[v][0] % 2 == 0) uj[k]++;
        }
    }

    void cover()
    {
        int i;
        for (i = 1; i <= u; i++)
        {
            if (uj[i] == -1)
                g++;
            BFS(i, g);
        }
    }
}

```

15.4 BFS -matrice de adiacenta

BFS - MATRICE DE ADIACENTA

```
void BFS (int x)
{
    int p, u, i;
    p = u = 1;
    c[p] = x;
    v[i][x] = 1;
    while (p <= u)
    {
        x = c[p+1];
        for (i = 1; i <= u; i++)
            if (a[x][i] == 1 && v[i] == 0)
            {
                c[++u] = i;
                v[i] = 1;
            }
    }
}
```

15.5 DFS - nerecursiv - matrice de adiacenta

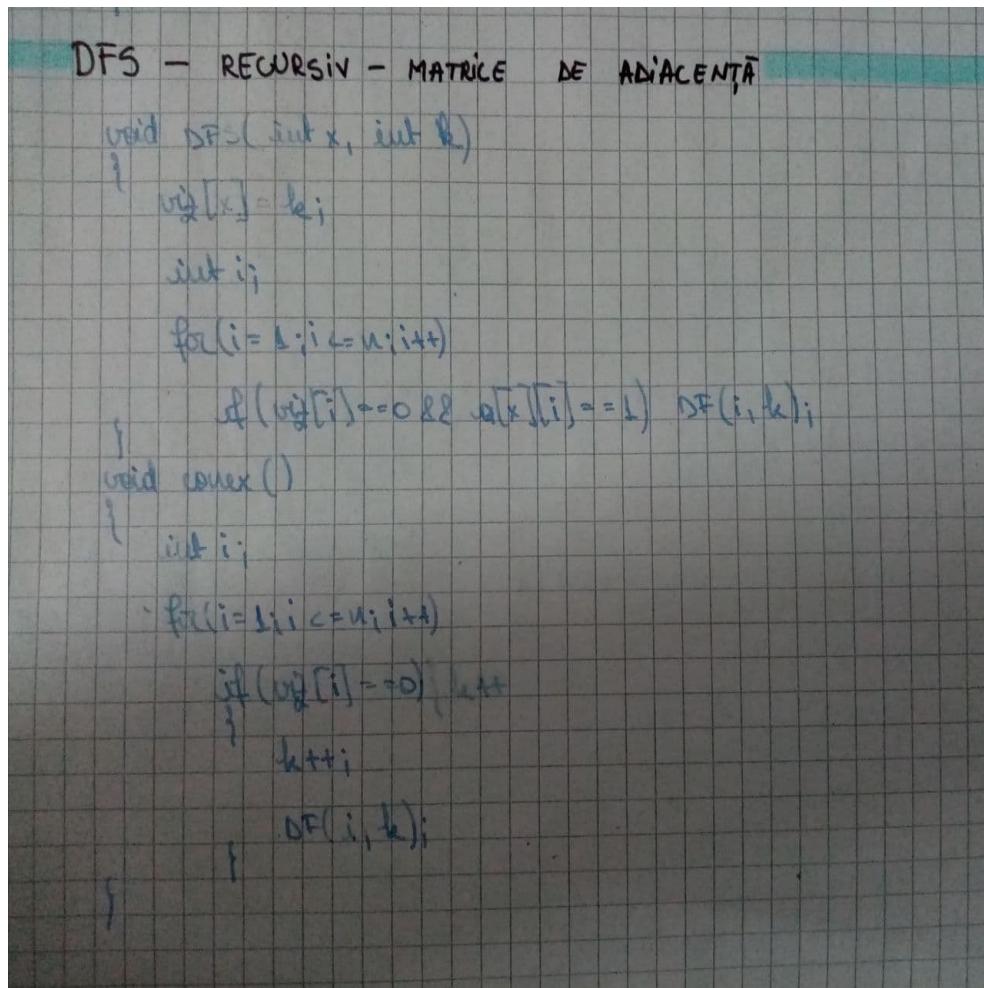
```

DFS - NERECURSIV - MATRICE DE ADIACENTA

void DFS(int x)
{
    int vf, i, ok;
    vf = 1;
    if (vf == x) vis[vf] = 1;
    while (vf > 0)
    {
        x = st[vf];
        ok = 0;
        for (i = 1; i <= n; ok++)
            if (a[x][i] == 1 && vis[i] == 0)
            {
                vf = i;
                vis[i] = 1;
                ok = 1;
                break;
            }
        if (ok == 0) vf--;
    }
}

```

15.6 DFS - recursiv - matrice de adiacenta



15.7 Algoritmul lui Dijkstra

https://www.youtube.com/watch?v=_1HSawdgXpI

Rol :

- pentru gasirea durmului de cost minim

Observatii :

- algoritmul PSP este o extensie a algoritmului lui dijkstra si se face doar in cazul in care se ia un subgraf a unui graf foarte foarte mare
- functioneaza numai pentru grafurile orientate (un singur sens)
- fiecare muchie a grafului trebuie sa aiba un cost
- functionaza si pe graf conex si neconex
- ai cost negativ \Rightarrow nu poti aplica
- nu poti pleca din nodul de start nicaieri \Rightarrow nu poti aplica

- daca se poate aplica: alegi mereu nodul vecin nevizitat cu costul cel mai mic si faci update la valoare; te opresti cand termini de vizitat toate nodurile.

Functionalitate :

- alegem nodul de start
- alg calculeaza distanta de la nodul ales la celelalte noduri
- distanta la fiecare nod este calculata in vectorul distanta[] cu n elemente
- Distanta[i] reprezinta distanta minima la un nod de start cu nodul numerotat 'i'
- 2 liste, unul cu nodurile vizitate, si unul cu nodurile nevizitate si al fiecare operatie se actualizeaza lista
 - la inceput in tabel/vector cu distanta de la x la y va fi initializat cu infinit, deoarece ne trebuie drumul minim
 - se compara muchiile nodului de start si se merge pe cea de cost minim, si asta se face la fiecare nod intalnit (graf neorientat)
 - in cazul grafului orientat se ia in functie de sensul muchiei

ALGORITM - gandire :

```

initializez: distanta de la nod la acelasi este = 0
initializez: distanta de la nod la celelalte = infinit
repeat
| vizitez nodurile nevizitate cu lungimea cea mai mica
| pentru nodul curent se examineaza vecinii nevizitati
| pentru nodul curent se calculeaza distanta de la fiecare vecin
| daca distanta calculata nodului este mai mica decat distanta stiuta, se da update la
|   distanta_minima
| update nodurile precedente pentru a updata lista cu distanta
| adaugat nodul current la lista cu nodurile vizitate
until (toate nodurile sunt vizitate)
  
```

15.8 Algoritmul Ford-Fulkerson

<https://www.youtube.com/watch?v=Tl90tNtKvxs>

15.9 Algoritmul de colorare greedy

atentie! este un algoritm greedy, NU e o colorare optima

<https://www.youtube.com/watch?v=vGjsi8NIpSE>

15.10 Algoritmul Bellman-Ford

<https://www.youtube.com/watch?v=obWXjtg0L64>

Teorie :

- distanta minima de la un nod la altul

- diferenta dintre Djk si Bell este ca Djk= greedy && Bell \neq greedy + Bell lucreaza si cu costuri negative

- nu functioneaza pe graf cu cicluri negative \implies nu exista drum de cost minim

*Complexitate : $\mathcal{O}(|V| * |E|)$*

INPUT :

- Graf orientat $G = (V, E)$;
- costul muchiilor fara cicluri negative;
- nod C sa apartina grafului V.

OUTPUT :

- Pentru toate nodurile u accesibile din s, $dist(u)$ este setat cu distanta de la s la u;

ALGORITM :

```

for  $u \in V$  do
    let  $dist(u) \leftarrow \infty$ ;
    let  $prev(u) \leftarrow NULL$ ;
end for
let  $dist(s) \leftarrow 0$ ;
repeat
     $|V| - 1 \times$  : // de  $|V| - 1$  ori; de ex: pentru 6 noduri = 5 iteratii
    for  $e \in E$  do
         $update(e)$ ;
    end for
     $update((u,v), \text{apartine } E)$ ;
    let  $dist(v) \leftarrow \min\{dist(v), dist(u) + l(u, v)\}$ ;
until

```

15.11 Metoda generala MST - Algoritmul lui Prim

<https://www.youtube.com/watch?v=cplfcGZmX7I>

Observatii :

- se foloseste un vector pentru a tine evidenta nodurilor vizitate
- este un algoritm greedy (alege cel mai mic cost si cel mai apropiat de radacina/mostly)

INPUT :

- graf conex $G=(V,E)$ cu cost

OUTPUT :

- MST definit de un array

Procedure prim(G, w)

```

for  $u \in V$  do
    let  $cost(u) \leftarrow \infty$ ;

```

```

let  $prev(u) \leftarrow NULL$ ;
end for
pick any initial node  $u_0$ ;
let  $cost(u_0) \leftarrow 0$ ;
let  $H \leftarrow makequeue(V)$ ; —> coada de prioritate, folosind costurile ca si key
while H not empty do
    let  $v \leftarrow deletemin(H)$ ;
    for  $u, z \in E$  do
        if  $cost(z) > w(u, z)$  then
            let  $cost(z) \leftarrow w(u, z)$ ;
            let  $prev(z) \leftarrow v$ ; decrease(H,z);
        end if
    end for
end while

```

*Complexitate timp : $\mathcal{O}(|V + E| * \log|V|)$*

15.12 Algoritmul lui Kruskal

<https://www.youtube.com/watch?v=71UQH7Pr9kU>

Observatii :

- in final, rezolva tot problema MST
- tot un alg greedy
- difera prin complexitate timp si prin faptul ca nu ai libera te sa alegi de unde vrei sa incepi, ci se ia muchia cu costul cel mai mic, dupa iar se face la fel pana se ajunge la un connected MST

*Complexitate timp : $\mathcal{O}(|E| * \log|E|)$*

INPUT :

- graf conex cu costuri

OUTPUT :

- MST

```

procedure kruskal(G,w)
    for  $u \in V$  do
        makeset(u);
    end for
    X=
    sort edges E by cost;
    for  $u, v \in E$  do
        if  $find(u) \neq find(v)$  then
            add edge u,v to X

```

```

        union(u,v)
end if
end for

```

UNDE MAKESET ESTE DEFINIT ASTFEL:

```

procedure makeset(x)
    let  $pi(x) \leftarrow x$ ;
    let  $rank(x) \leftarrow 0$ ;

```

UNDE FIND ESTE DEFINTI ASTFEL:

```

function find(x)
    while  $x \neq pi(x)$  do
        let  $x \leftarrow pi(x)$ ;
    end while
    return x

```

SI UNDE UNION ESTE DEFINIT ASTFEL:

```

procedure union(x,y)
    let  $r_x \leftarrow find(x)$ ;
    let  $r_y \leftarrow find(y)$ ;
    if  $r_x == r_y$  then
        return null
    end if
    if  $rank(r_x) > rank(r_y)$  then
        let  $pi(r_y) \leftarrow r_x$ ;
    else
        let  $pi(r_x) \leftarrow r_y$ ;
    end if
    if  $rank(r_x) == rank(r_y)$  then
        let  $rank(r_y) \leftarrow rank(r_y) - 1$ 
    end if

```

15.13 Algoritmul lui Hopcroft si Karp - pentru gasirea cuplajului maxim

<https://www.youtube.com/watch?v=CSUEVu-qUgM>

Gandire :

- avem nevoie de un graf bipartit
- BFS, DFS
- M este multimea cuplajului maxim

ALGORITM :

1. M = multume vida

2. repeat

- BFS - pentru a constui nivele diferite a grafului cu noduri unmatched (din afara cuplajului)

- se maresteste cuplajul curent M cu un set maximal de noduri si drumuri disjuncte de cea mai scurta lungime (disjoint shortest-lengths paths) (using DFS)

until nu mai exista drumuri de maximizat

3. return M

```

 $M_0 \leftarrow \emptyset; i = 0;$ 
while ( $\exists$  drumuri de creștere relativ la  $M$ ) do
    fie  $P_i$  un drum de creștere minim relativ la  $M_i$ ;
     $M_{i+1} \leftarrow M_i \Delta P_i$ ;
     $i++$ ;
end while

```

Lema1 : Dupa n iteratii cel mai scurt drum, trebuie sa fie macar de lungime M.

Lema2 : $|M'| - |M| \leq \frac{|V|}{n}$, unde n este lungimea drumului

Complexitate temp : $\mathcal{O}(\sqrt{V})$

15.14 Algoritmul de Unificare

ALGORITMUL DE UNIFICARE

```
void unificare()
{
    int i, j, cx, cy;
    for(i=1; i<=n; i++) c[i]=i;
    for(i=1; i<=m; i++)
    {
        cx=c[v[i].x];
        cy=c[v[i].y];
        if(cx!=cy)
            for(j=1; j<=n; j++)
                if(c[j]==cx) c[j]=cy;
    }
}
```

15.15 Algoritmul lui Lee

```

ALGORITMUL LUI LEE

struct coord { int x, y; } c[100];
int intersect ( int x, int y ) { return ( x >= 1 && x <= n && y >= 1 && y <= m ); }

void lee ( int x, int y )
{
    p = m + 1; c[m].x = x; c[m].y = y; a[x][y] = 1;
    while ( p <= n )
    {
        x = c[p].x; y = c[p].y;
        for ( i = 0; i < 8; i++ )
        {
            xv = x + dx[i]; yv = y + dy[i];
            if ( intersect ( xv, yv ) && a[xv][yv] == 0 )
                a[xv][yv] = a[x][y] + 1;
            c[++n].x = xv; c[n].y = yv;
        }
    }
}

int main ()
{
    while ( 1 ); lee ( x, y );
    cout << a[x][y] - 1;
}

```

15.16 Grafuri bipartite

```

GRAFURI BIPARTITE

int Bipart (int x)
{
    p = u = l;
    c[l] = r; v[r][x] = l;
    while (p <= u)
    {
        x = c[p++];
        for (i = l; i <= u; i++)
            if (a[i][x] == 1) | & (v[r][i] == 0) { c[++u] = i; v[r][i] = j - v[r][x]; }
            else if (v[r][x] == v[r][i]) return 0;
        return 1;
    }
    cout << main();
    cin >> a;
    if (Bipart(1) == 0) cout << "NU";
    else { Bipart(1); Bipart(2); }
}

```

15.17 Sortare topologica

```

SORTARE TOPOLOGICA

void afis() {
    for(i=1; i<=n; i++) cout << v[i] << " ";
}

void DF (int x) {
    v[x] = 1;
    for(i=1; i<=n; i++) if(a[x][i] == 1 && v[i] == 0) DF(i);
    v[x] = 0;
}

void sorta () {
    for(i=1; i<=n; i++) if(v[i] == 0) DF(i);
}

void citire () {
    fin >> n >> m;
    for(i=1; i<=n; i++)
        for(j=1; j<=m; j++)
            fin >> x >> y;
            a[x][y] = 1;
}

int main () {
    citire ();
    sorta ();
    afis ();
}

```

15.18 Drum minim

```

DRUM MINIM

void BF(int p)
{
    if (l <= t[i], k)
        for (i = l; i <= u; i++) v[i] = -1;
    c[e] = p; v[e][p] = 0;
    while (f <= l)
    {
        p = c[f++];
        for (i = l; i <= u; i++) if (a[p][i] == 1 && v[i] == -1) { v[i] = p; c[i] = i; e[i] = i; }
    }

    void d(p, int x, int q)
    {
        k = 1; v[k] = x;
        while (x != p)
            if (i = l; i >= 1 - i; i--) tout << v[i] << " ";
        cout << min();
        while (1); BF(p);
        if (v[g] == -1) d(p, g, p);
        else tout << "No solution.";
    }
}

```

15.19 Verificare graf conex

```
VERIFICARE      GRAF      CONEX
void BFS( int k);
int conex()
{
    int i;
    for(i=1;i<=n;i++)
        if(vst[i]==0) return 0;
    return 1;
```

15.20 Determinarea si afisarea componentelor conexe

DETERMINAREA și AFISAREA COMPONENTELOR CONEXE

```
void citire();
void BFS(int x, int nr);
void afis(int k)
{
    int i;
    for(i=1; i<=nr; i++)
        if((w[i] == k) && c[i] == 0)
            cout << "n";
}
int main()
{
    int i;
    for(i=1; i<=nr; i++)
        if((w[i] == 0) && c[i] == 0)
            BFS(i, nr);
    afis(nr);
}
```

15.21 Determinarea si afisarea componentelor conexe - optimizat - numarul maxim de varfuri de grad par

COMP CONEXE CU OPTIM
nr. max. de vf. de grad par

```

void afis()
{
    fin>>v;
    int x,y;
    while (fin>>x>>y)
    {
        a[x][0]++; a[y][0]++;
        a[x][a[x][0]] = y;
        a[y][a[y][0]] = x;
    }
}

void conex()
{
    for(i=1;i<=n;i++)
    {
        if(v[i][i]==0)
        {
            g++; BFS(i,g);
        }
    }
}

int main()
{
    afis(); conex();
    for(i=1;i<=g;i++)
        if(un[i]>max) max=un[i];
    for(i=1;i<=g;i++)
        if(un[i]==max) dpo(i);
}

```

void BFS (int x, int k)
{
 p = u = l;
 c[p] = x; v[k][x] = k;
 if((a[x][0])%2==0) w[k][k]++;
 while (p<=u)
 {
 x = c[p+1];
 for(i=1;i<=n;i++)
 {
 if(v[i][x]==0)
 {
 c[i+1] = x; v[k][x] = k;
 if((a[x][i])%2==0) w[k][k]++;
 }
 }
 }
}

15.22 Lungime minima intre 2 varfuri

```

LUNGIME MINIMA INTRE 2 VF.

void dist(int x)
{
    u=p=1;
    c[p]=x;
    for(i=1;i<=u;i++) d[i]=-1;
    d[1]=0;
    while(p<=u)
    {
        x=c[p+1];
        for(i=1;i<=u;i++)
            if(a(x)[i]==1&& d[i]==-1)(c[++u]=i; d[i]=d[x]+1);
    }
}
int main()
{
    cin>>s>>t;
    dist(s);
    cout<<d[t];
}

```

15.23 Determinarea si afisarea unui ciclu eulerian

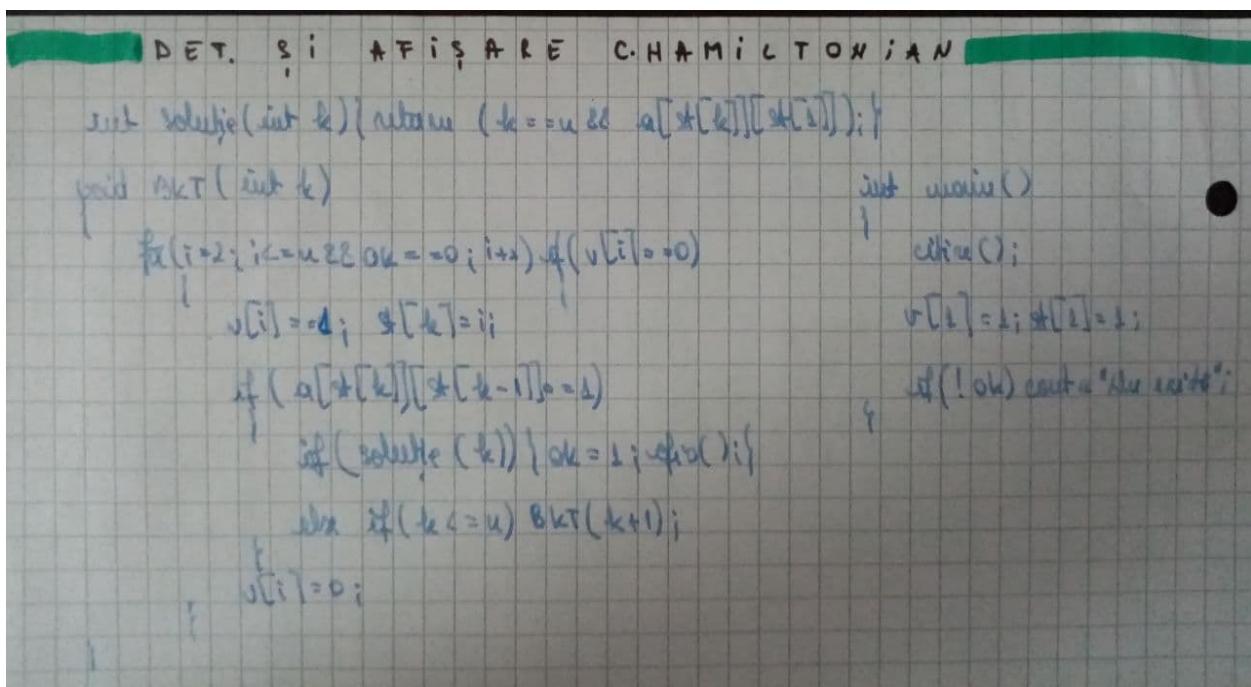
```

DET. SI AFISARE C. EULERIAN
void citire()
{
    int u, v;
    for (int i = 1; i <= u; i++)
    {
        cout << x << y;
        if (x == y) cout << " ";
        cout << grad[x] << " ";
        cout << grad[y] << " ";
        grad[x]++;
        grad[y]++;
    }
}

void DF(int x)
{
    int of = 1, st[of] = x;
    while (of > 0)
    {
        x = st[of];
        if ((grad[x] == 0)) cout << x << " "; of--;
        else
        {
            for (int i = 1; i <= u; i++)
                if (a[x][i] == 1)
                {
                    st[++of] = i;
                    a[x][i] = a[i][x] = 0;
                    grad[x]--;
                    grad[i]--;
                    break;
                }
        }
    }
}

```

15.24 Determinarea si afisarea unui ciclu hamiltonian



15.25 Clase de echivalenta

```

CLASE DE ECHIVALENTA

void DF (int x, int k)
{
    vis[x] = k;
    for (i=1; i<=n; i++) if (a[i][x] == 0 && vis[i] == 0) DF(i, k);
}

void soliq (int k)
{
    p=0; tout<-'{';
    for (i=1; i<=k; i++) if (vis[i] == k)
        {
            if (p>0) p=-1;
            else tout<-'}';
            tout<-i;
        }
    tout<-'}';
}

int main ()
{
    entree(); wr=0;
    for (i=1; i<=n; i++) if (vis[i] == 0) wr++; DF(i, wr); soliq(wr);
}

```