

Algoritmica grafurilor - Cursul 1

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.

1 Descrierea cursului

- Pagina cursului și alte informații

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

3 Vocabularul teoriei grafurilor

- 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 *

4 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 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 *

Pagina cursului și alte informații

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

Pagina cursului: <http://profs.info.uaic.ro/~olariu>

Obiective:

Cursurile acoperă noțiunile de bază în Teoria algoritmică a grafurilor. Cunoștințele acumulate vor fi aplicate în proiectarea unor algoritmi eficienți pentru rezolvarea problemelor de optimizare combinatorială.

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

Metode de predare:

Prezentarea video a notelor de curs, care vor fi posteate ca fișiere .pdf înaintea fiecărui curs. Aceste fișiere vor conține și exercițiile pentru seminarii.

Graph Algorithms * C. Croitoru - Graph Algorithms *

*C. Crișan - Graph Algorithms * C. Crișan - Graph Algorithms * C. Crișan - Graph Algorithms*
Seminarii: E. F. Olariu, A. C. Frăsinaru, A. Policiuc, V. Motroi

Fiecare seminar este dedicat unui număr de exerciții (postate în avans în notele de curs) cu scopul de a aprofunda concepțele introduse la curs. Studenții sunt încurajați să propună soluții originale.

C. Crișan - Graph Algorithms
Notarea:

- **Activitatea de la seminar:** prezență (max. 8 puncte), participare (max. 10 puncte) - max 18 puncte.
- **Teme pentru acasă:** trei seturi de exerciții, max. 14 puncte fiecare - max. 42 puncte
- **Examen final scris:** max. 60 puncte

Dintr-un maximum de 120 puncte limita de promovare este de 50 puncte.

of G .

Cuprinsul cursului:

- Vocabularul teoriei grafurilor.
- Probleme de drum: parcurgerea sistematică a grafurilor, drumuri de cost minim, conexiune.
- Arbori parțiali de cost minim: union-find, complexitate amortizată.
- Teoria cuplajelor.
- Fluxuri în rețele.
- Reduceri polinomiale între probleme de decizie pe grafuri.
- Abordări ale problemelor NP-dificele.
- Grafuri planare.

Aplicații ale teoriei grafurilor

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

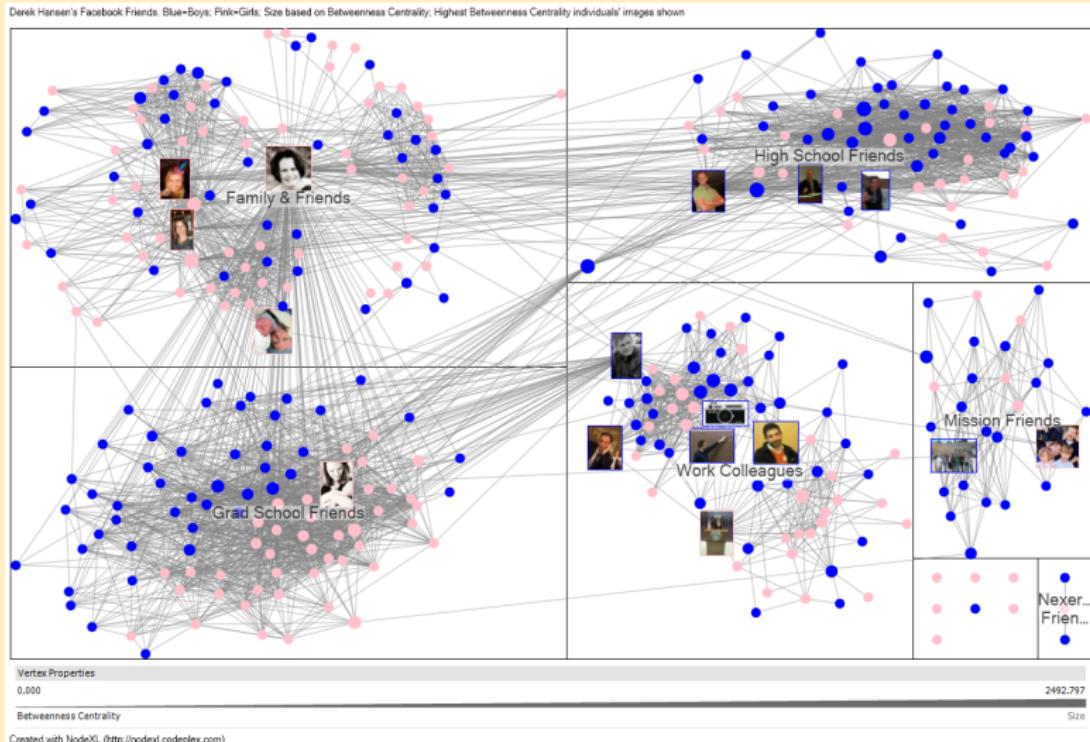


Figure: Facebook/Twitter

Aplicații ale teoriei grafurilor

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

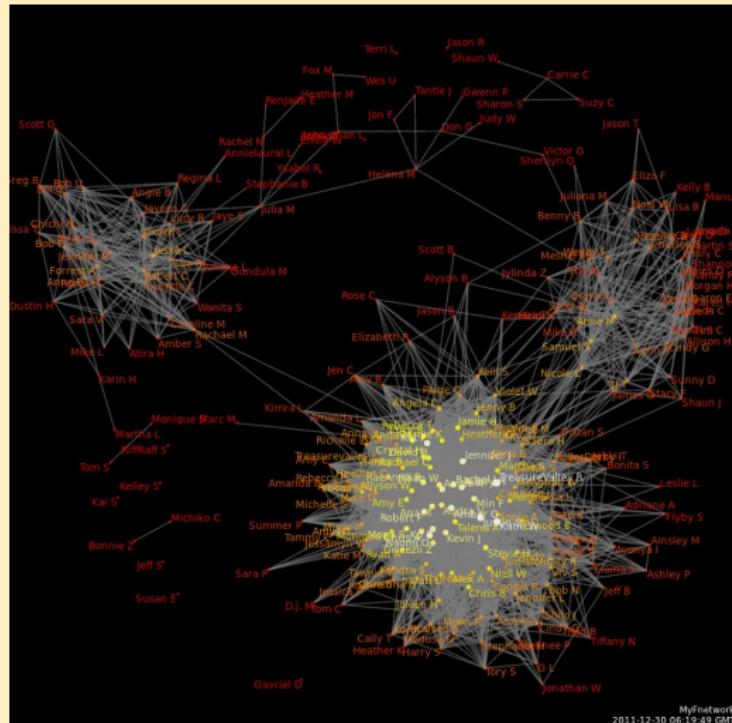


Figure: A Facebook network of a few users

Aplicații ale teoriei grafurilor

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;**
- **robustetea sau stabilitatea structurală a rețelei.**

Analiza rețelelor este folosită pentru

- **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

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

În afara studiului rețelelor sociale care sunt un subiect la modă astăzi mai sunt multe alte aplicații ale teoriei grafurilor:

- **Arbori parțiali de cost minim:** pentru conectarea eficientă a unor puncte de comunicație (e. g. în IT&C);
- **Drumuri și circuite Euleriene:** Problema poștașului 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;**

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

Aplicații ale teoriei grafurilor

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

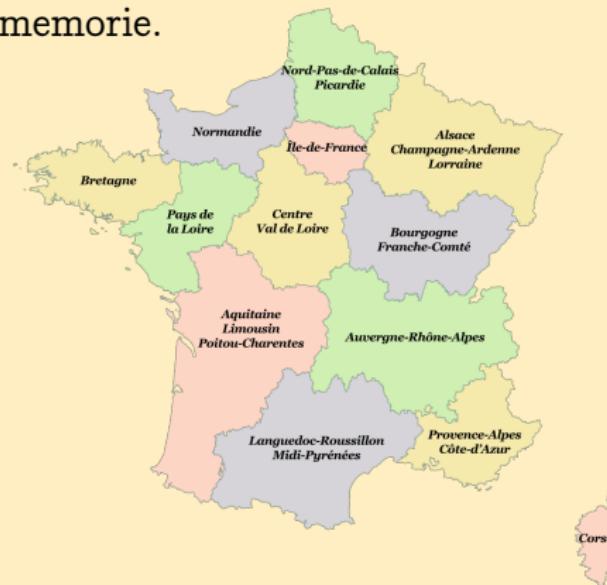


Figure: Regiunile Franței începând cu 2016

Aplicații ale teoriei grafurilor

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

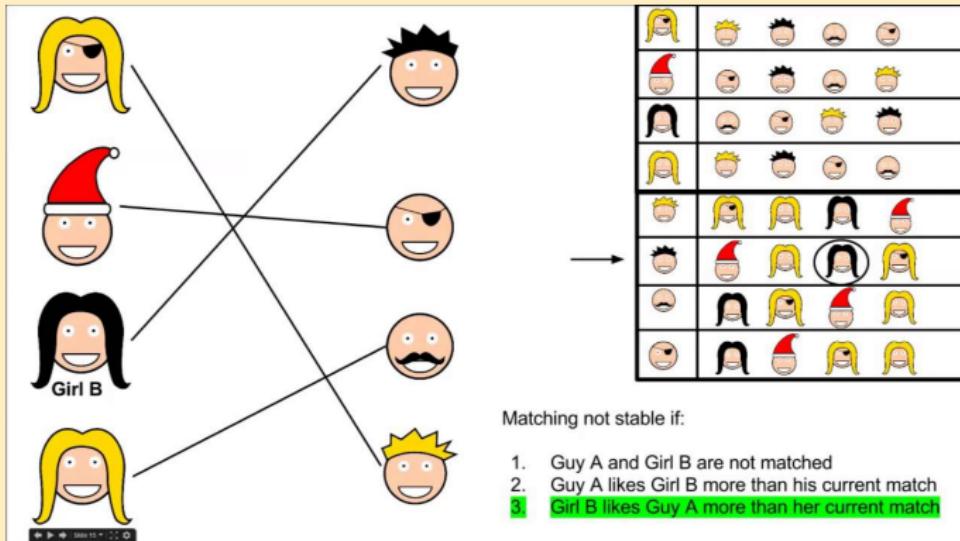


Figure: Gale Shapley algorithm

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 -
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 -
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).

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

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

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 *

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 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 .

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

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

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

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

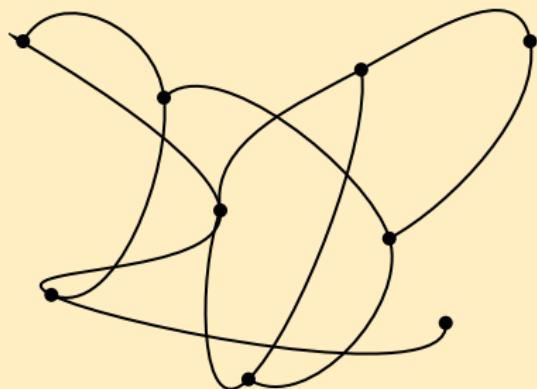
Definiția grafului

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

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.

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

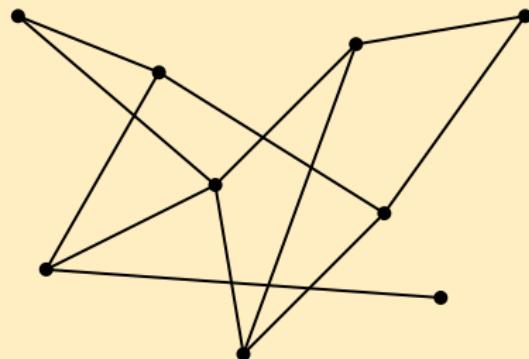
Un exemplu:



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 *

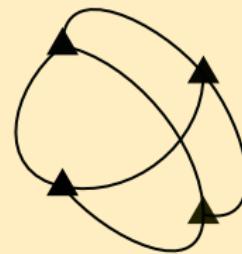
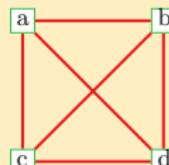
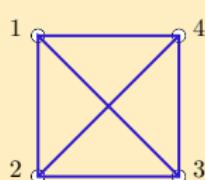
Definiția grafului

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

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



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 3

Mulțime stabilă (sau mulțime independentă de noduri) în $G = (V, E)$: $S \subseteq V$ astfel încât $\binom{S}{2} \cap E = \emptyset$.

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

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

Notație

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

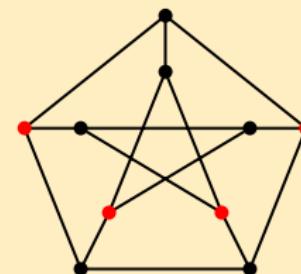
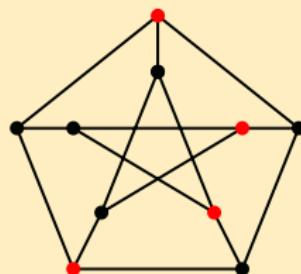
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

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



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 *

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
Algoritmi * 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 $e \cap f = \emptyset$.

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

Cu alte cuvinte, $M \subseteq E$ este un cuplaj dacă oricare două muchii ale sale nu noduri înn 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 $\nu(G)$.

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 Algorithms * C. Croitoru - Graph Algorithms *

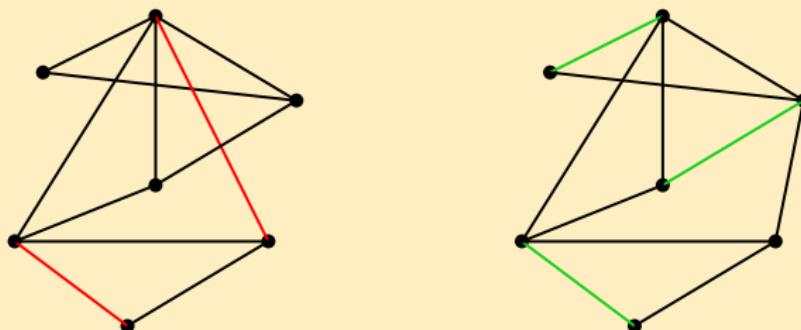
Definiția grafului

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 -

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 *

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 -

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 diferența între ele?

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 mulțimea 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

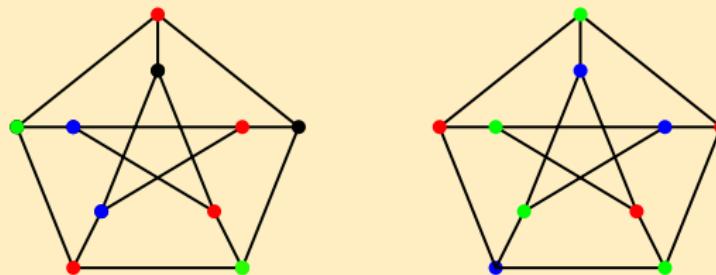
Numărul cromatic al grafului G este cea mai mică valoare a lui p astfel încât G are o p -colorare. Acest parametru se notează cu $\chi(G)$. ($\chi(G) \leq |G|$ - de ce?)

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.

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

Colorarea nodurilor: Problemă de optimizare

P3 Input: G graf.

Ouput: $\chi(G)$ 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 *

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 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 partiții a lui E cu cel mult p cuplaje.

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

Notație

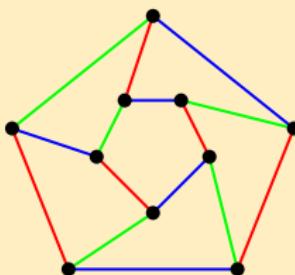
Indexul cromatic al grafului G : cea mai mică valoare a lui p astfel încât G are o p -muchie colorare. Acest parametru este notat cu $\chi'(G)$. ($\chi'(G) \leq |E(G)|$ - de ce?)

Definiția grafului

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

Colorarea muchiilor: **Problemă de optimizare**

P3 Input: G graf.

Ouput: $\chi'(G)$ 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
Algoritmi și C. Caiet de lucru în cadrul unei proiecte

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., $u_1v_1 \in E_1$) dacă și numai dacă $\varphi(u_1)$ și $\varphi(v_2)$ sunt adiacente în 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

$$G_1 \cong G_2.$$

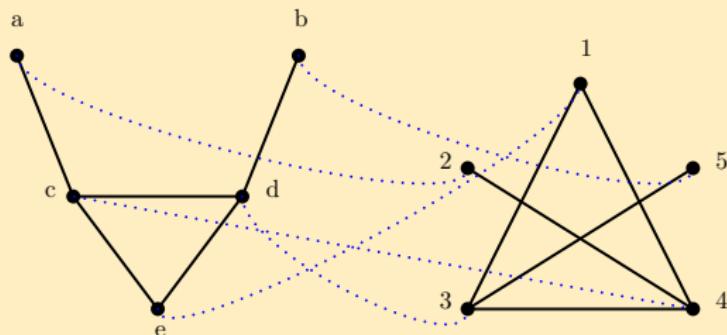
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

Grafurile de mai jos sunt izomorfe.



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

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**).

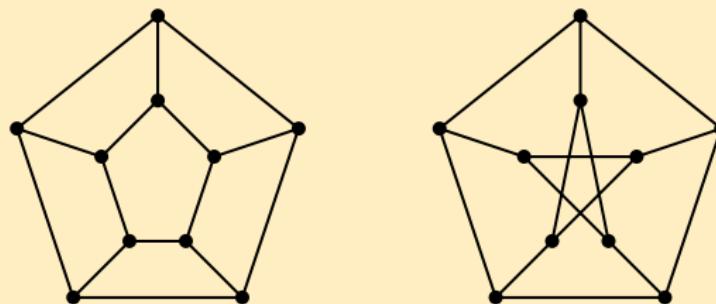
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru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

Următoarele două grafuri au aceleași ordine, dimensiuni și secvențe ale gradelor, dar nu sunt izomorfe (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 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
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exercițiu 1. Pentru $k \in \mathbb{N}^*$, considerăm $G_k = K_2 \times K_2 \times \dots \times K_2$ (de k ori).

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

Exercițiu 2.

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

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

Exercițiul 3.

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 .

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

Exercițiu 4. O sesiune de examene trebuie planificată folosind următoarele specificații: mulțimea examenelor este cunoscută; fiecare student trimite 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 aceeași zi?
- (b) Care este numărul minim de zile necesare organizării unei sesiuni?

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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. (exercițiul 4 - 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 multimea examenelor neplanificate încă), până când toate examenele vor fi planificate.

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

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 6. Un exemplu de optimizare a unui compilator este **tehnica alocării regiștrilor** de memorie: cele mai folosite variabile sunt ținute în registrii cu acces rapid ai procesorului pentru a le avea la îndemână în momentul când compilatorul are nevoie de ele (pentru anumite operații CPU).

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

- Care este numărul maxim de variabile de care nu este nevoie în același timp?
- Care este numărul minim de regiștri necesari acestor variabile?

Indicație: avem două tipuri de obiecte: **variabilele** (cu valorile lor) și **operațiile CPU** (sau operatorii) care folosesc una sau mai multe variabile.

Exerciții pentru seminarul din săptămâna viitoare

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

Exercițiul 7. (exercițiul 6 - continuare) Un student se întreabă dacă la întrebările de mai sus (care corespund unor probleme NP-hard) nu s-ar putea da un răspuns 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 alocare a regiștrilor astfel încât graful construit (ca mai sus) să fie tocmai G .

Studentul sugerează următoarea “abordare greedy” pentru a răspunde la a doua întrebare din exercițiul 6: începând cu primul registru, se alocă fiecărui registru nefolosit un număr maxim de variabile (dintre cele nealocate încă), până când toate variabilele vor fi alocate.

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

Exercitii pentru seminarul din săptămâna viitoare

Exercițiu 8. G este numit graf *autocomplementar* dacă G și complementul său, \overline{G} , sunt izomorfe ($G \cong \overline{G}$).

- (a) Arătați că un graf *autocomplementar* este conex și că $|G| \equiv 0$ sau $1 \bmod 4$.
 - (b) Determinați toate grafurile *autocomplementare* cu cel mult 7 noduri.
 - (c) Arătați că pentru orice graf G există un graf *autocomplementar* H astfel încât G este subgraf induș al lui H .

Algoritmica grafurilor - Cursul 2

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.

1 Vocabularul teoriei grafurilor

- **Variatii in definitia unui graf**
- **Grade**
- **Subgrafuri**
- **Operatii cu grafuri**
- **Clase de grafuri**
- **Drumuri si circuite**

2 Exercitii pentru seminarul de saptamana 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 * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Variații 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

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 $m : \binom{V}{2} \rightarrow \mathbb{N}$.

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

$e \in \binom{V}{2}$, 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 **multiplicitatea** $m(e)$.

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
Graful suport al unui graf, G , este graful obținut din G prin înlocuirea fiecărei muchii multiple printr-o simplă.

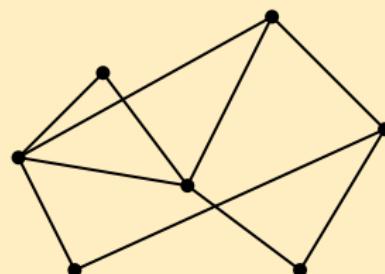
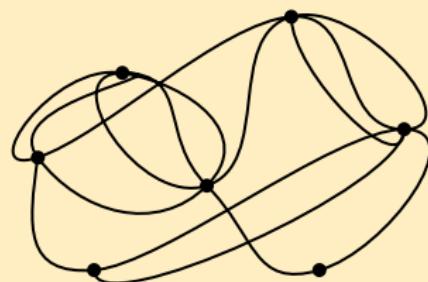
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Variatii în definitia 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 *

Variatii în definiția unui graf

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 **bucătă**.

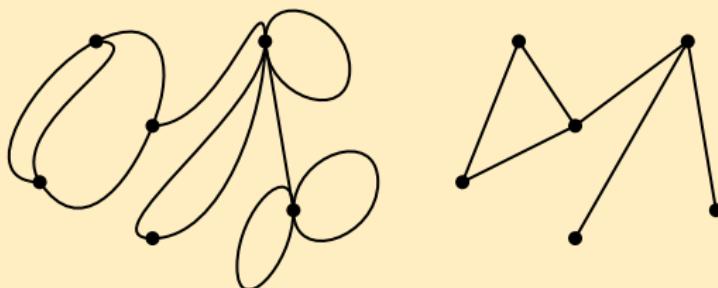
Graful suport al unui multigraf G este graful obținut din G prin înlăturarea fiecărei muchii multiple printr-una simplă și prin ștergerea buclelor.

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 pseudograaf 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 *

Variații 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 -

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 inițială 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.

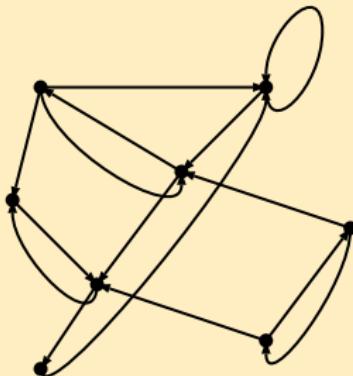
Graph Algorithms * C. Croitoru - Graph Algorithms *

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 *

Variații 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
Algoritma * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

- **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).

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

Variatii în definitia unui graf

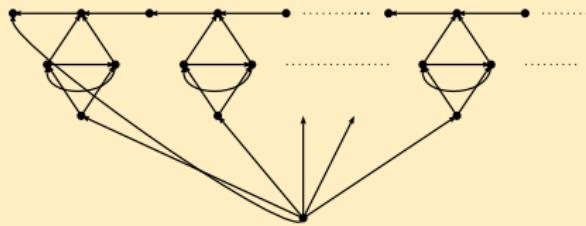
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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

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



The Danaids barrel

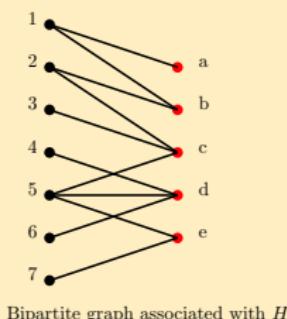
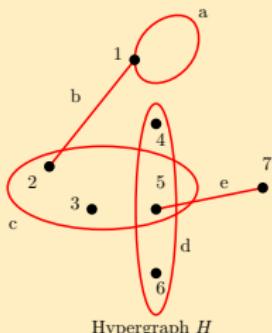


The sisyphus graph

Hipergrafuri (Sisteme de multimi finite)

- Muchiile, numite acum **hipermuchii**, 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 .

Fiecare hypergraf poate fi reprezentat ca un 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 -

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 $d_G(v) = 1$.
- **Gradul maxim** $\Delta(G)$ și **gradul minim** $\delta(G)$:

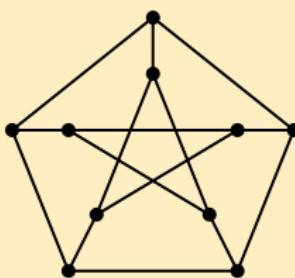
$$\Delta(G) = \max_{v \in V} d_G(v), \quad \delta(G) = \min_{v \in V} d_G(v).$$

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

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

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

Grade

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 $G = (V, E)$ 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 .

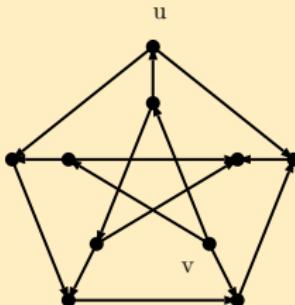
$$\sum_{v \in V} d_G^+(v) = \sum_{v \in V} d_G^-(v) = |E|.$$

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 *

Subgrafuri

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

Fie $G = (V(G), E(G))$ 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 inducător**: un subgraf H al lui G aşa încât $E(H) = \binom{V(H)}{2} \cap E(G)$. Dacă $A \subseteq V(G)$, **subgraful inducător de** A în G este notat cu $[A]_G$ sau $G[A]$.

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

Subgrafuri

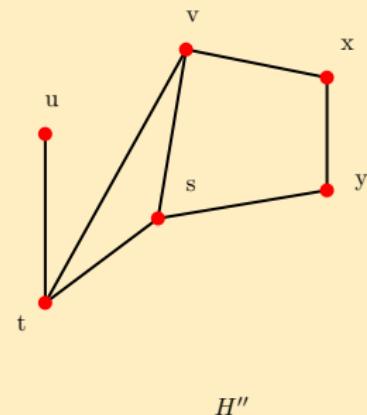
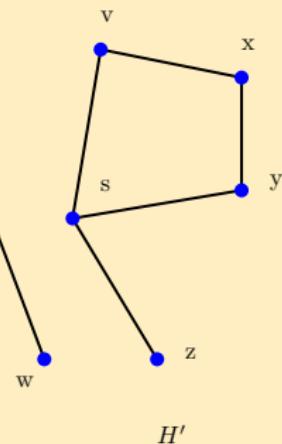
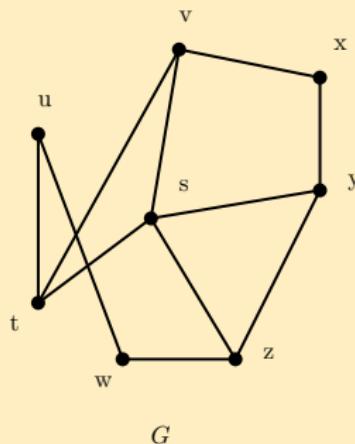
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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

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

Exemplu

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



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

Subgrafuri

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

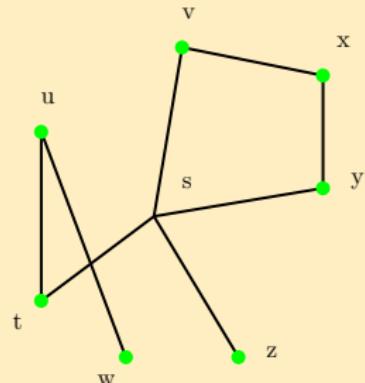
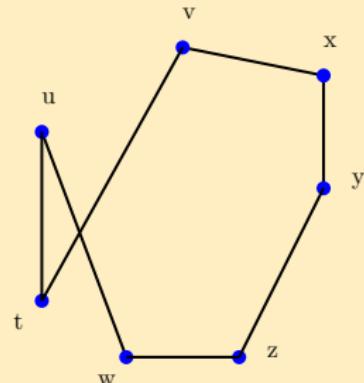
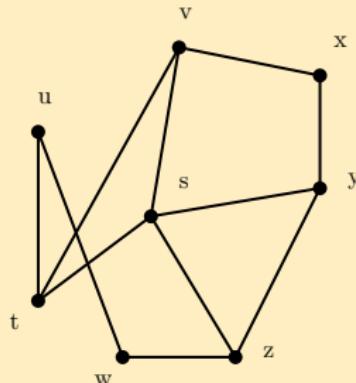
- Dacă $A \subseteq V(G)$, atunci subgraful $[V(G) \setminus A]_G$, notat prin $G - A$, este subgraful obținut din G prin stergerea nodurilor lui A . $G - \{u\}$ este numit **subgraf de stergere** și se notează, simplu cu $G - u$.
 - Dacă $B \subseteq E(G)$, atunci subgraful $\langle E(G) \setminus B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

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 -

Exemplu

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



Graph Algorithms * C. Croitoru - Graph Algorithms *

Operații cu grafuri

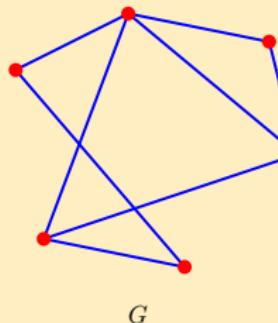
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Operație unară: $G = (V(G), E(G))$

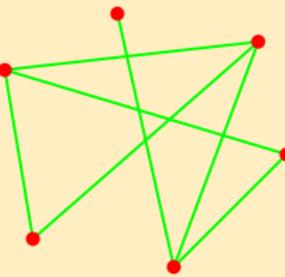
- **Complementul** unui graf G : un graf \overline{G} , cu $V(\overline{G}) = V(G)$ și

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

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



G



\overline{G}

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

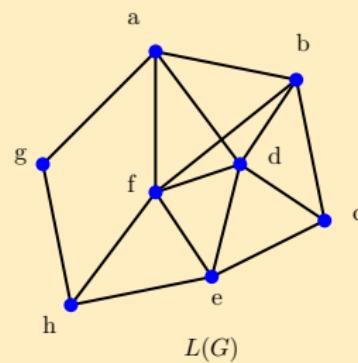
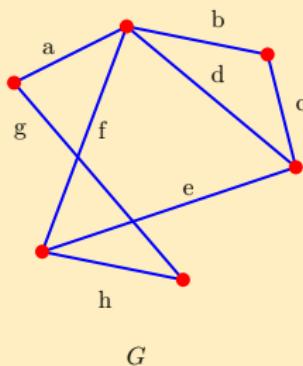
Operații cu grafuri

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

Operație unară: $G = (V(G), E(G))$

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

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



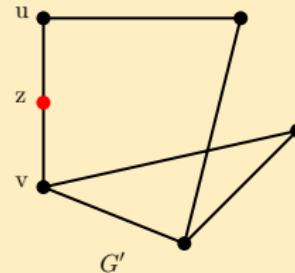
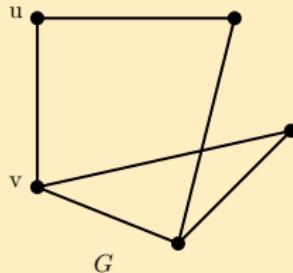
Operații cu grafuri

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 unară: $G = (V(G), E(G))$

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

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



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

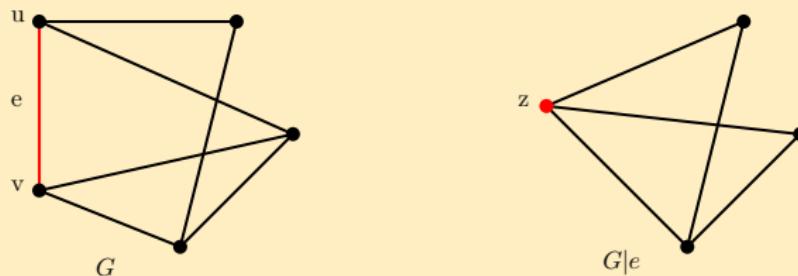
Operație unară: $G = (V(G), E(G))$

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

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

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

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



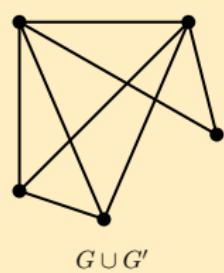
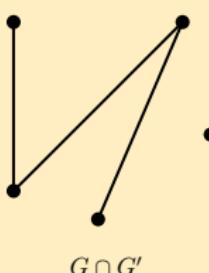
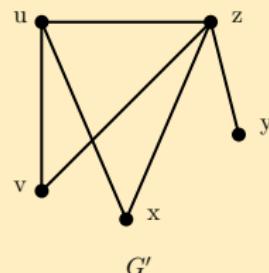
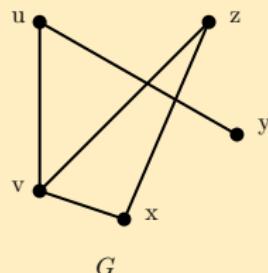
Operații cu grafuri

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

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

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

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



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

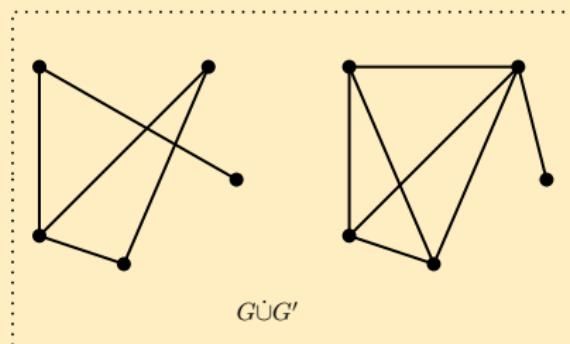
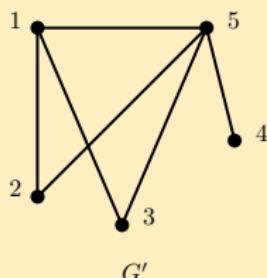
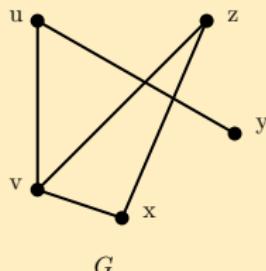
Operații cu grafuri

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

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 *

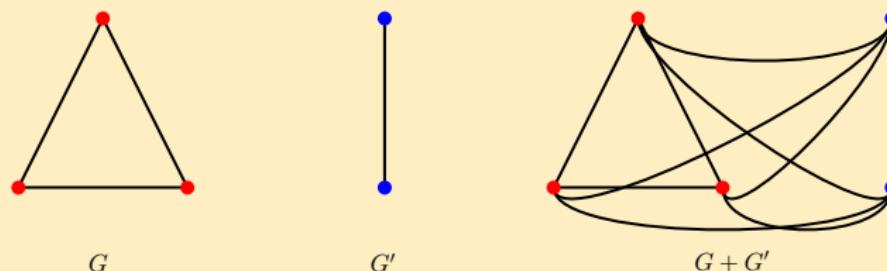
Operații cu grafuri

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)** $G + G' = \overline{G} \dot{\cup} \overline{G'}$.

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
- 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

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

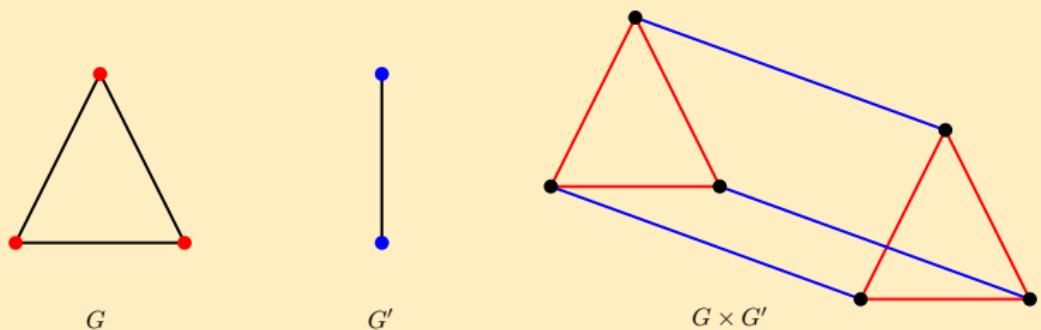
- **Produsul cartezian** al grafurilor G și G' : graful $G \times G'$ cu

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

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

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

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



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 $E(K_n) = \binom{V(K_n)}{2}$.

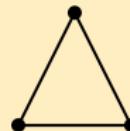
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru



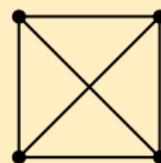
K_1



K_2



K_3



K_4



K_5

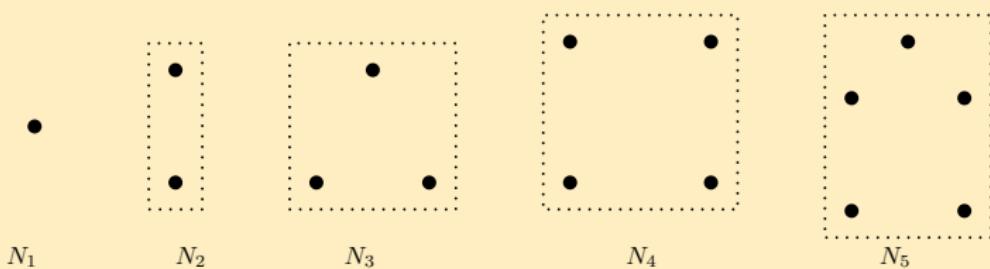
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

Clase de grafuri - Grafurile nule

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 nul de ordin n , N_n : $|V(K_n)| = n$ și $E(K_n) = \emptyset$.

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

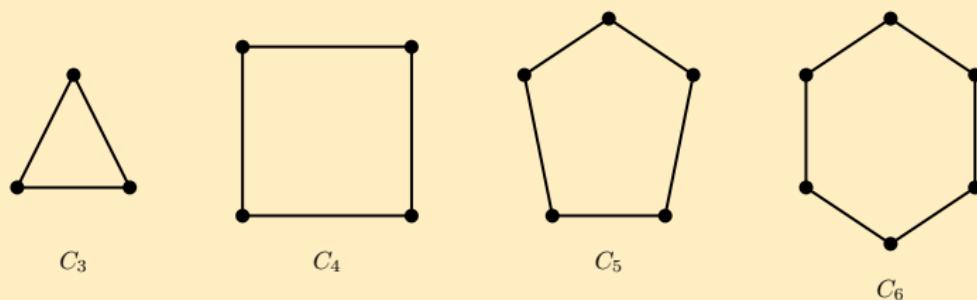


Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 $E(C_n) = \{12, 23, \dots, n-1n, n1\}$.

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

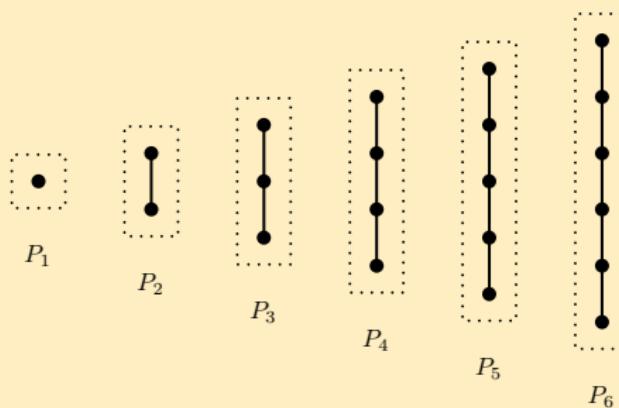


Clase de grafuri - Drumurile P_n

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 -

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



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 - Clicile

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

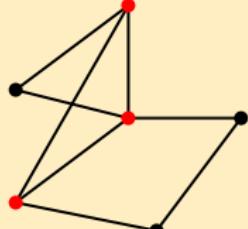
O submulțime de k noduri a graf G care induce un graf complet este numită o **k -clică**.

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

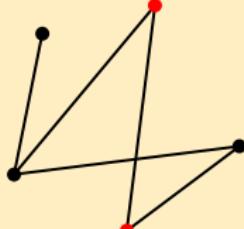
Remarcăm că $\omega(G) = \alpha(\overline{G})$.

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

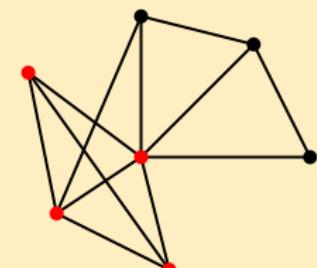
Exemplu



$$\omega(G) = 3$$



$$\omega(G) = 2$$



$$\omega(G) = 4$$

Clase de grafuri - Grafurile bipartite

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

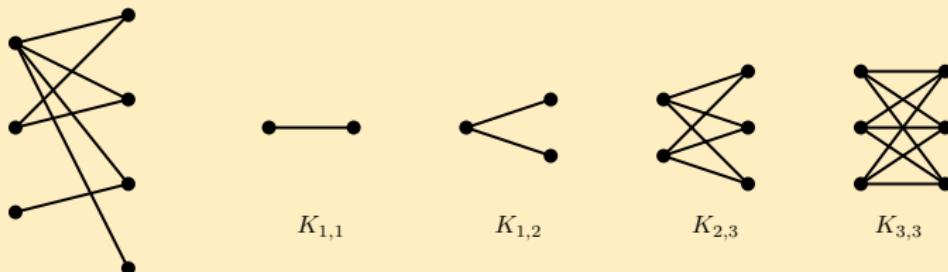
Graf bipartit: un graf G cu proprietatea că $V(G)$ poate fi partizionat în două clase care sunt mulțimi stabile.

Dacă $V(G) = S \cup T$, $S \cap T = \emptyset$, $S, T \neq \emptyset$, cu S și T mulțimi 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 $s = |S|$, $t = |T|$.

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

Exemplu



A bipartite graph

Clase de grafuri - Grafuri planare

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

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

Graf planar: un graf care poate fi reprezentat într-un plan astfel ca fiecare nod să îi corespundă un punct al aceluia plan și fiecare muchie 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, $\mathcal{O}(n + m)$).

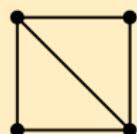
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Clase de grafuri - Grafuri planare

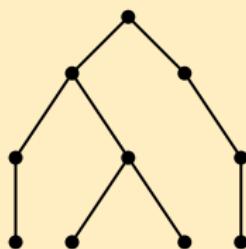
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

Exemplu

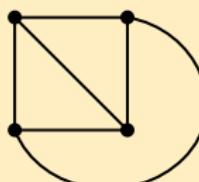
Grafuri planare.



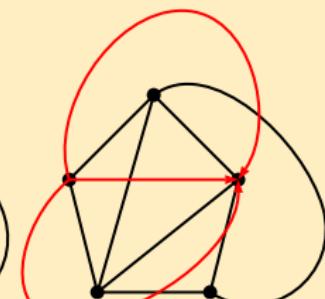
planar graph



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 inducător isomorf 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.

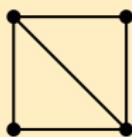
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

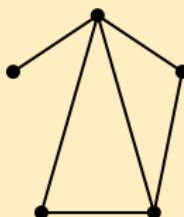
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru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

Exemplu

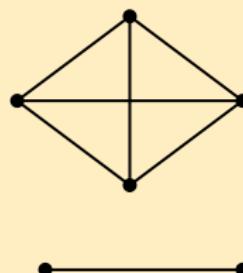
Grafuri \mathcal{F} -free.



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 -
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

$$(u =)v_0, v_0v_1, v_1, \dots, v_{r-1}, v_{r-1}v_r, v_r (= v).$$

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.

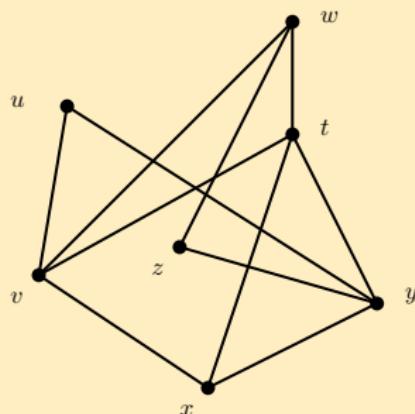
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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 -

Exemplu

Mersuri, parcursuri, drumuri.



walk:



trail:



path:



Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 închise, circuite

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

- **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 .

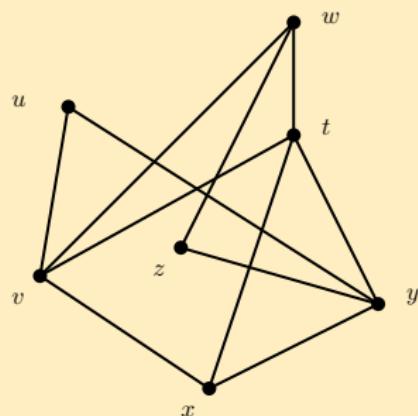
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C.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

Drumuri și circuite - Mersuri închise, circuite

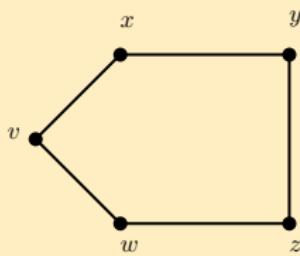
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Exemplu

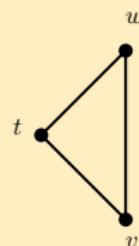
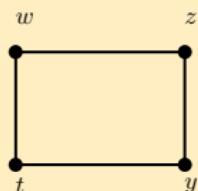
Mersuri închise, circuite.



odd circuits:



even circuit:



Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri și circuite - Distanță, diametru

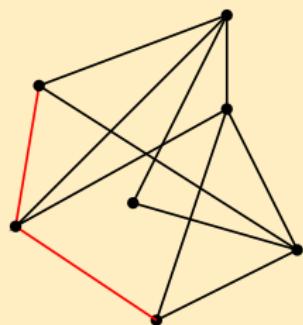
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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

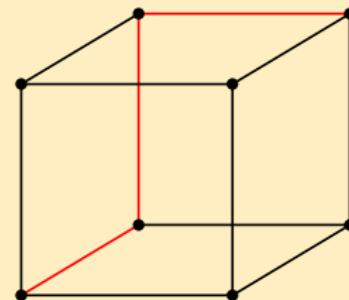
- **Distanța în G de la u la v , $d_G(u, v)$** lungimea celui mai scurt drum în G de la u la v (dacă există un astfel de drum).
- **Diametrul unui graf G , $d(G)$:**

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

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



$$d(G) = 2$$



$$d(G) = 3$$

Drumuri și circuite

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

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 *

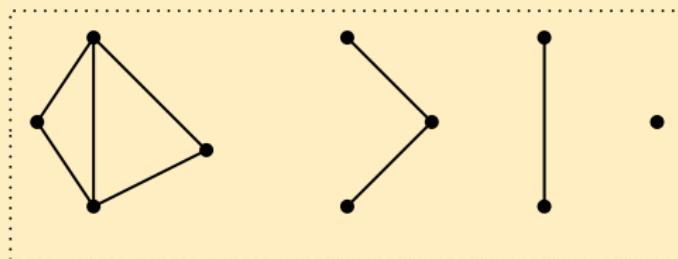
Fie $G = (V, E)$ un graf.

- **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 .

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

Exemplu



four connected components

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

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

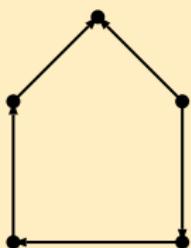
- **Digraf slab conex** (sau simplu, **conex**) graful său suport $G(D)$ este conex.
- **Digraf unilateral conex**: există un drum de la u la v sau de la v la u , pentru orice două noduri $u, v \in V$.
- **Digraf tare conex**: există un drum de la u la v , pentru orice două noduri $u, v \in V$.

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

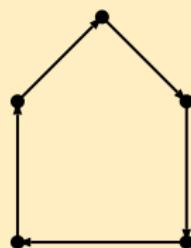
Drumuri și circuite - Conexiune

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 * C.

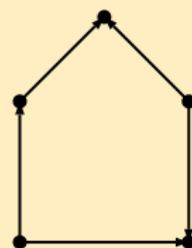
Exemplu



unilaterally connected



strongly connected



weakly connected

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

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

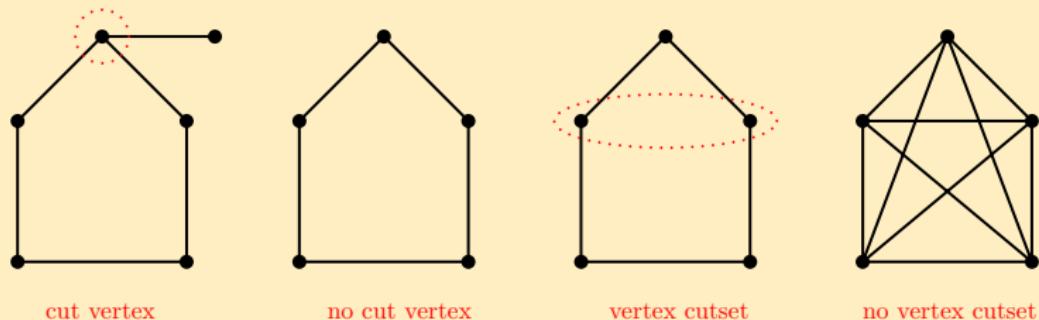
- **Punct de articulație (cut-vertex):** un nod $v \in V$ astfel că $G - v$ este neconex.
- **Mulțime de articulație (vertex cutset):** o mulțime de noduri $S \subseteq V$ așa încât $G - S$ este neconex.
- Un **arbore** este un graf conex fără circuite.
- Un graf ale cărui componente conexe sunt arbori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

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

Exemplu



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

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 mulțime 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

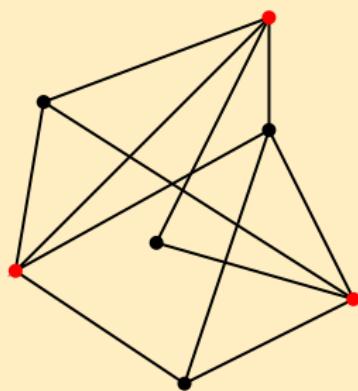
$$k(G) = \max \{p \in \mathbb{N}^* : G \text{ este } p\text{-conex}\}.$$

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 *

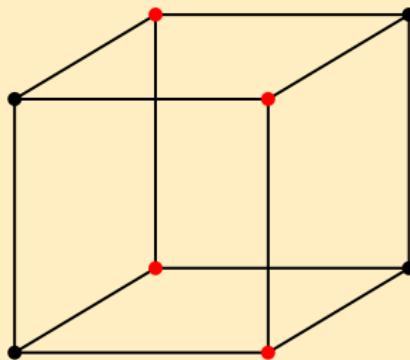
Drumuri și circuite - Conexiune

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

Exemplu



$$k(G) = 3$$



$$k(G) = 4$$

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

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.

- **Punte (bridge):** o muchie $e \in E$ astfel că $G - e$ nu este conex.
- **Mulțime de muchii de articulație (tăietură sau edge-cutset):**
O submulțime 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

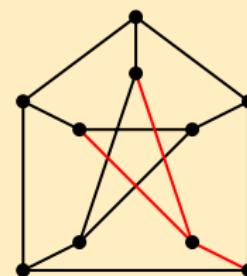
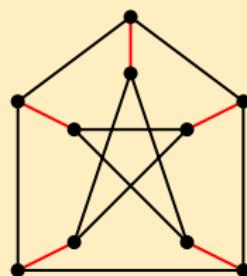
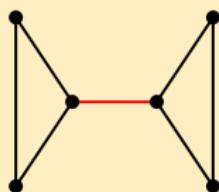
$$\lambda(G) = \max \{p \in \mathbb{N}^* : G \text{ este } p\text{-muchie-conex}\}.$$

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 * C.

Exemplu



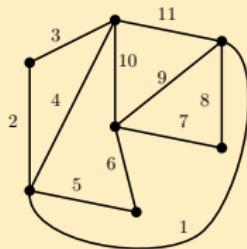
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru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

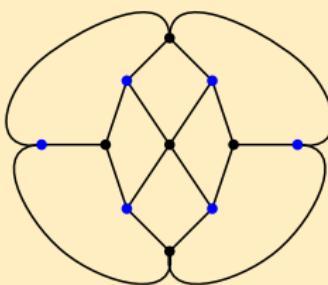
Fie G un (di)graf.

- 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 .

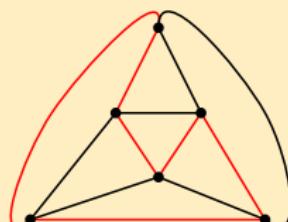
Recunoașterea (di)grafurilor Euleriene se face în timp polinomial (**Euler, 1736**).



an Eulerian graph



a non Hamiltonian graph
(bipartite of odd order)



a Hamiltonian graph

Drumuri și circuite - Grafuri Euleriene și 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 -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

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 ∈ NP?

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

Exercițiu 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.

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

Exercițiu 2.

Fie $P(n) =$ " În orice graf cu cel puțin n noduri există noduri distincte care sunt două câte două adiacente sau două câte două neadiacente."

Arătați că 6 este cea mai mică valoare a lui $n \in \mathbb{N}$ pentru care $P(n)$ este adevărată.

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

Exercițiu 3.

Fie D un turneu conținând un circuit C de lungime ≥ 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 .

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

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.

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

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 $n \geq 3$, atunci G este conex dacă și numai dacă conține două noduri nu sunt puncte de articulație.

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.

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 7

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

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

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.

* 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

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.

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

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.

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

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 $d(G) \leq 2$.

Algoritmica Grafurilor - Cursul 3

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.

1 Vocabularul teoriei grafurilor

- Matrici asociate

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

2 Structuri de date

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

3 Probleme de drumuri în (di)grafuri

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

- Drumuri de cost minim

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

4 Exercitii 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.

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

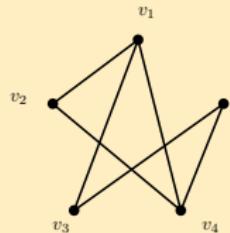
Matrici asociate - Matricea de adiacență

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

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

Exemplu

Un graf și matricea sa de adiacență.



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

Matrici asociate - Matricea de incidentă

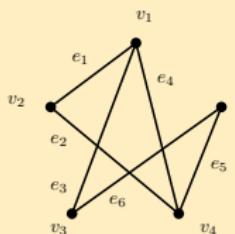
Fie G un graf cu $V(G) = \{v_1, \dots, v_n\}$ și $E(G) = \{e_1, \dots, e_m\}$.

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.



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

Matrici associate

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

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

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

Structuri de date pentru matricea de adiacență

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

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

- Dacă $A = (a_{ij})_{1 \leq i, j \leq n}$ este matricea de adiacență a lui G atunci, reprezentând-o ca un tablou 2-dimensional, avem nevoie de $\mathcal{O}(n^2)$ timp pentru inițializare (în funcție de limbajul de programare).
- Astfel, orice algoritm care reprezintă G cu matrice de adiacență are complexitatea timp (și spațiu) $\Omega(n^2)$.
- Testarea dacă două noduri fixate sunt adiacente se face în $\mathcal{O}(1)$, dar parcurgerea întregii multimi 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.

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

Structuri de date pentru listele de adiacență

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

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

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.

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

Probleme de drumuri - Breadth-First Search (BFS)

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

for $v \in V$ **do**

$label(u) \leftarrow -1$; $parent(u) \leftarrow -1$;

$label(s) \leftarrow 0$; $parent(s) \leftarrow 0$;

 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)

Proprietăți ale BFS. Nu este dificil de arătat că:

- $S = \{u \in V : \text{label}(u) \geq 0\}$;
 - $\forall u \in V, \text{label}(u) = d_G(s, u)$ (distanța în G de la s la u);
 - Variabila **parent** definește **arborele bfs** asociat parcurgerii din s : dacă G este graf atunci arborele bfs este un arbore parțial al componentei conexe conținând s ; dacă G este digraf atunci arborele bfs este o **arborescentă** (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).

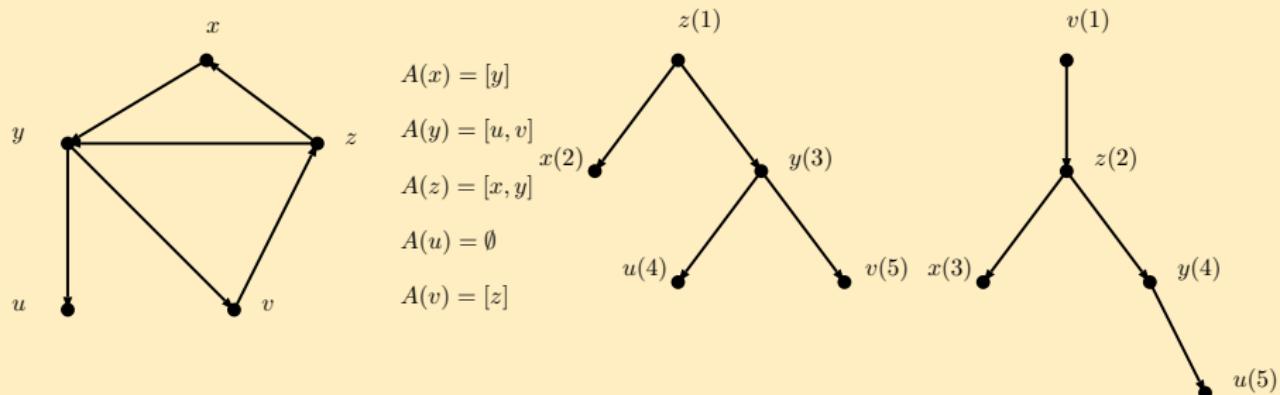
Probleme de drumuri - Breadth-First Search (BFS)

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

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

Exemplu

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



(Am marcat în paranteze ordinea de vizitare.)

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

Probleme de drumuri - Depth-First Search (DFS)

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

for $v \in V$ **do**

$label(u) \leftarrow -1$; $parent(u) \leftarrow -1$;

$label(s) \leftarrow 0$; $parent(s) \leftarrow 0$;

 crează o stivă S conținând s ; $n_S \leftarrow 0$;

while $S \neq \emptyset$ **do**

$u \leftarrow top(S)$;

if $((v \leftarrow next[A(u)]) \neq NULL)$ **then**

if $label(v) < 0$ **then**

$n_S ++$; $label(v) \leftarrow n_S$;

$parent(v) \leftarrow u$; $push(S, v)$;

else

$delete(S, u)$;

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

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

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).

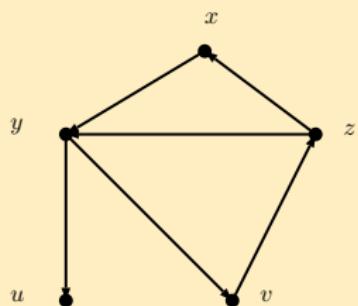
Algorithms * C. Croitoru - Graph Algorithms *

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 v și din x):



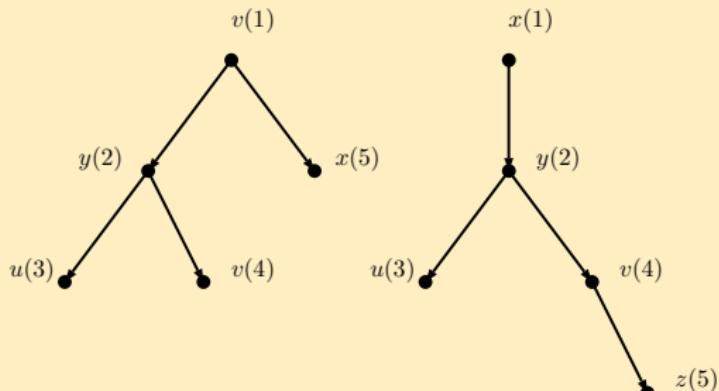
$$A(x) = [y]$$

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

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

$$A(u) = \emptyset$$

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



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

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

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

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

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

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 - 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 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

- 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

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

- În particular $a(P_{ii}) = 0$.

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

Principalele probleme de drum de cost minim

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

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

Drumul de cost minim între două noduri.

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

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

Drumurile de cost minim dintre un nod și toate celelalte noduri

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

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

Toate drumurile de cost minim.

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

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

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

Remarci 1

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

1. Rețele de comunicare (Communication Networks). Dignaful $G = (V, E)$ 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 parcugerea 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**:

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

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

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

Luând $a'(e) = -\log a(e)$,

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

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

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

2. Rețele PERT - Metoda drumului critic (Critical path method - CPM). PERT (Pert 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_i poate începe înainte ca activitatea A_j să se fi terminat.
- Pentru fiecare activitate A_i , timpul necesar finalizării t_i este cunoscut (sau doar estimat).

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

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 * C.

Asociem acestei problem un digraf aciclic astfel:

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

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

Drumuri de cost minim - Aplicații

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

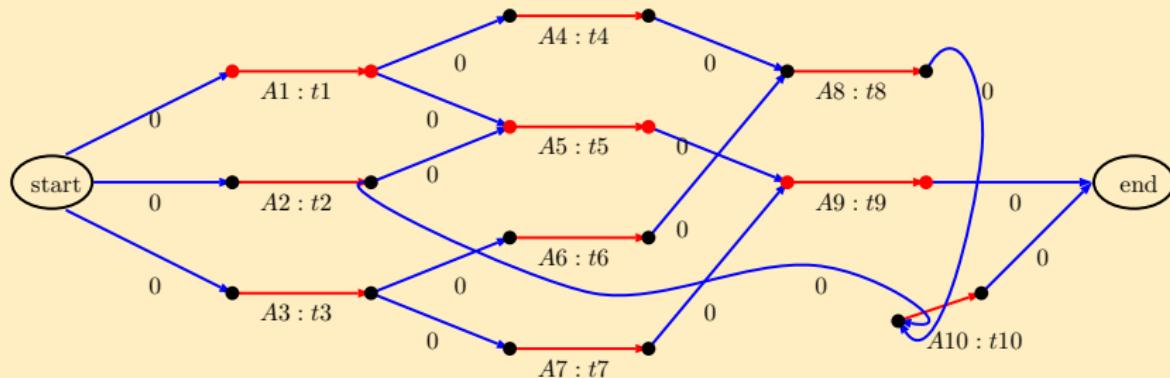
- Construcția digrafului este încheiată după adăugarea unui nod s corespunzând momentului inițial al proiectului, legat prin arce si_p pentru fiecare activitate A_p în care nu intră vreun arc și a unui nod t corespunzând finalizării proiectului, legat prin arce $j_p t$ pentru fiecare activitate A_p din care nuiese 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.

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 * 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 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

- 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

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

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

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

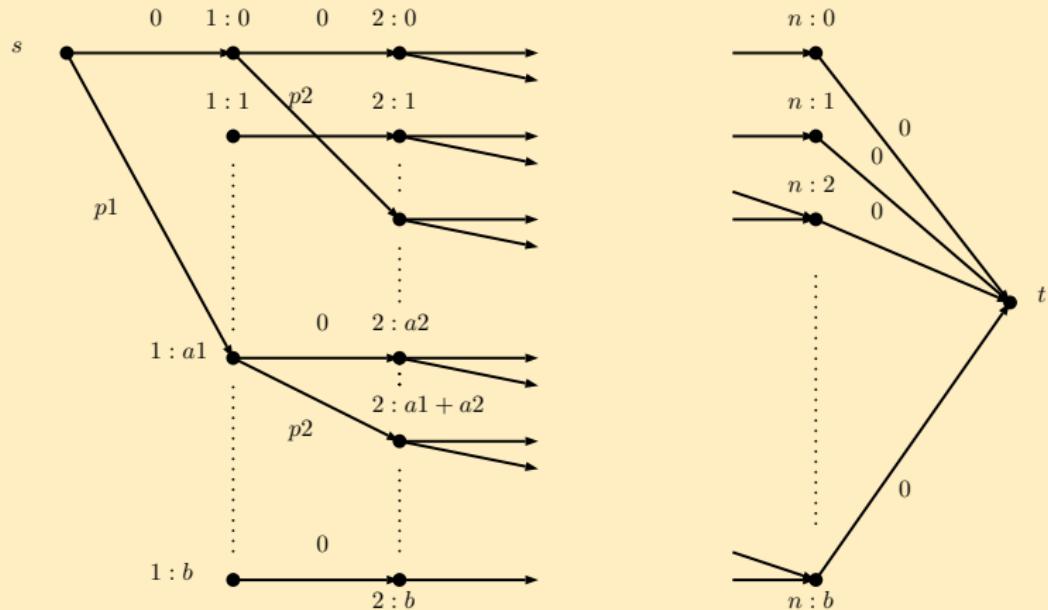
Remarci 2

- Fiecare drum de la s la t în G corespunde unei submulțimi de obiecte cu nivelul de umplere $\leq b$ și cu profitul total egal cu costul drumului. Deoarece, reciproc, fiecărei umpleri a rucsacului îi corespunde un drum 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).

Drumuri de cost minim - Aplicații

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

Exemplu



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

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

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

Drumuri de cost minim - Rezolvarea problemei P2

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

Proof:

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

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

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

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

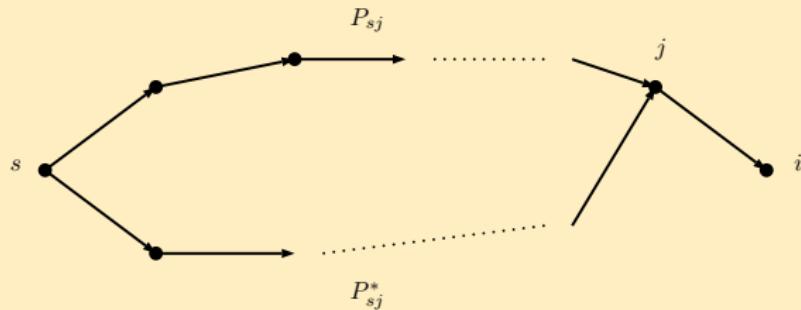
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru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

Drumuri de cost minim - Rezolvarea problemei P2

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

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).

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



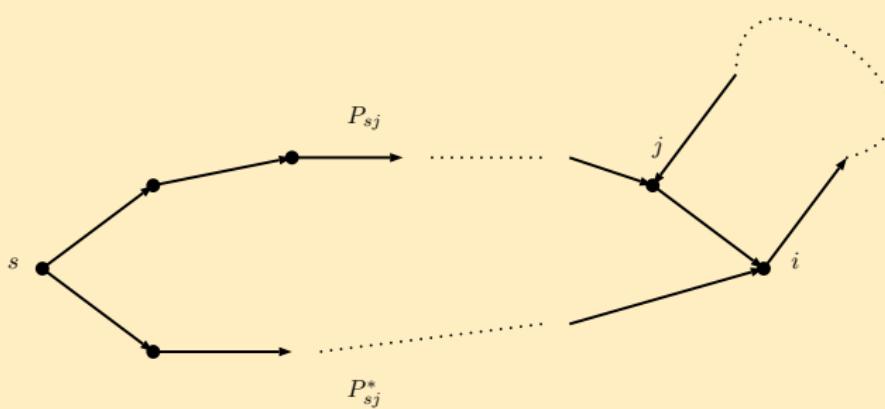
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Rezolvarea problemei P2

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

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

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

Drumuri de cost minim - Rezolvarea problemei P2

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

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

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

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

(a) Dacă $i = s$, atunci $u_s = 0$ și drumul P_{ss} 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

Drumuri de cost minim - Rezolvarea problemei P2

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

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

În fiecare iterație **while** $w \notin \{i_0, \dots, i_{k-i}\}$ (altfel obținem un circuit de cost 0, în contradicție cu ipoteza (I)).

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

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

Drumuri de cost minim - Rezolvarea problemei P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
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 satisfacă (B)).

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



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

Drumuri de cost minim - Rezolvarea problemei P2

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

Remarci 3

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

Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Rezolvarea problemei P2

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
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 rezolva **P2** în ipoteza

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

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

Drumuri de cost minim - Rezolvarea problemei P2

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

Remarci 5

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

Drumuri de cost minim - Rezolvarea problemei P2

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

Remarci 6

- Deoarece mulțimile 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 -

Exercițiu 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

Exercițiu 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 *

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 * C. Croitoru -

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 incidență nod-arc $Q \in M_{n \times m}(\{-1, 0, 1\})$.

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

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

Algorithms * C. Croitoru - Graph Algorithms *

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 * C. Croitoru

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 incidență a lui D , unde

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

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

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

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 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).

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

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.

- 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 * C. Croitoru

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

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

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

Exerciții pentru seminarul 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 * C. Croitoru

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

Exercice 9.

- 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 *

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

Exercise 10. Demonstrați că un graf G este bipartit dacă și numai dacă orice subgraf induc H al lui G satisface inegalitatea: $2\alpha(H) \geq |H|$.

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

Exercise 11. Fie $G = (S, T; E)$ un graf bipartit și $X \in \{S, T\}$. G se numește **X -lanț** dacă nodurile lui X pot fi ordonate: x_1, x_2, \dots, x_k ($|X| = k$) astfel ca

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

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

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

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

Exercice 12. 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 .

- a) Arătați că $b(G)$ este graf bipartit.
- b) 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.

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

Algoritmica grafurilor - Cursul 4

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

1

Probleme de drumuri în digrafuri

- Drumuri de cost minim - Problema P2 pentru dags: sortarea topologică
- Drumuri de cost minim - Problema P2 pentru costuri nenegative
- Drumuri de cost minim - Problema P2 pentru costuri reale
- Drumuri de cost minim - Solving all-pairs shortest drum problem

* P3

Înmulțirea (rapidă) a matricilor

2

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

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

Lema 1

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

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

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 contradicție

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

" \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 .

Drumuri de cost minim - Problema P2 pentru dags

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

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

□

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

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

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

Drumuri de cost minim - Problema P2 pentru dags

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

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

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

Drumuri de cost minim - Problema P2 pentru dags

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

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

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

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

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

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

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

```
u1 ← 0; before[1] ← 0;  
for (i = 2, n) do  
    ui ← ∞; before[i] ← 0;  
    for (j = 1, i - 1) do  
        if (ui > uj + aji) then  
            ui ← uj + aji; before[i] ← j;  
// complexitatea timp  $\mathcal{O}(n^2)$ ;
```

Drumuri de cost minim - Problema P2 pentru costuri nenegative

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

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

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

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

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

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

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

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

Initial $S = \{s\}$ 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.

Drumuri de cost minim - Problema P2 pentru costuri nenegative

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

Algoritmul lui Dijkstra

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

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

Demonstrația corectitudinii algoritmului lui Dijkstra

Deoarece (D) are loc după pasul de 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 a mai întâi că dacă j^* este astfel încât $u_{j^*} = \min \{u_j : j \in V \setminus S\}$, atunci

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

Să presupunem că $\exists P_{sj^*}^1 \in \mathcal{P}_{sj^*}$ 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 $a(P_{sk}^1) = u_k$. 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 iteratărea 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 iteratărea while: $\forall j \in V \setminus (S \cup \{j^*\})$

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

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

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

Demonstrația corectitudinii algoritmului lui Dijkstra (cont.)

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

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

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

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**).



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

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

Complexitatea timp a algoritmului lui Dijkstra

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

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

timpul general necesar algoritmului pentru a actualiza valorile u_j este

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

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

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

Complexitatea timp a algoritmului lui Dijkstra

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

Drumuri de cost minim - Problema P2 pentru costuri nenegative

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

Remarci 1

- Pentru a rezolva problema P1 folosind algoritmul lui Dijkstra, adăgăm un test de oprire a execuției când nodul t este introdus în S . În cazul cel mai nefavorabil complexitatea rămâne aceeași.
- O euristică interesantă care dirijează căutarea spre t se obține cu ajutorul unui **estimator consistent**, i. e. o funcție $g : V \rightarrow \mathbb{R}_+$ care satisfac 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.

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

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

Remarci 2

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

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

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

Algoritmiști C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 - Problema P2 pentru costuri nenegative

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

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

Drumuri de cost minim - Problema P2 pentru costuri reale

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

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.

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

Algoritmul Bellman-Ford-Moore

Definim

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

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

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

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

Drumuri de cost minim - Problema P2 pentru costuri reale

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

Algoritmul Bellman-Ford-Moore

```
us1 ← 0;  
for (i ∈ V \ {s}) do  
    ui1 ← asi; //evident (BM) are loc.  
for (m = 1, n - 2) do  
    for (i = 1, n) do  
        uim+1 ← min(uim, minj ≠ i (ujm + aji));
```

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

* 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 multimi de drumuri:

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

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

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

Atunci, $A = B \cup C$, și

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

Deoarece u_i^m satisfacă (BM) avem

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

Algoritmul Bellman-Ford-Moore

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

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

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

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

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

Remarci 4

Algoritmul Bellman-Ford-Moore

- Putem adăuga algoritmului următorul pas:

```

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

```

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

Drumuri de cost minim - Rezolvarea problemei P3

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

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

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

- Fie $u_{ij} = \min \{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$. Astfel avem de determinat matricea $U = (u_{ij})_{n \times n}$, când matricea de cost-adiacență A este dată.
- Fiecare drum de cost minim poate fi obținut în $\mathcal{O}(n)$ dacă, în timpul construcției matricei U , întreținem matricea $Before = (before_{ij})_{n \times n}$, unde $before_{ij} =$ nodul dinaintea lui j pe drumul de cost minim de la i la j din G .
- Dacă perechea (G, a) satisfac 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

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

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

Iterarea algoritmului lui Dijkstra

Dacă toate costurile sunt nenegative, putem rezolva **P3** aplicând algoritmul lui Dijkstra pentru $s \in \{1, \dots, n\}$, 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 $\alpha : V \rightarrow \mathbb{R}$ astfel încât $\forall ij \in E, \alpha(i) + a_{ij} \geq \alpha(j)$.

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

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

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

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

Drumuri de cost minim - Rezolvarea problemei P3

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

Iterarea algoritmului lui Dijkstra

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

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

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

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

Drumuri de cost minim - Rezolvarea problemei P3

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

Iterarea algoritmului lui Dijkstra

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 $\alpha(i) = u_i$, $\forall i \in V$ aşa încât condiția (*) să aibă loc.

Algoritmul Floyd - Warshall

Fie

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

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

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

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 -

Algoritmul Floyd - Warshall

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

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

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

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

Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Rezolvarea problemei P3

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

```
for (i = 1, n) do
    for (j = 1, n ) do
        before(i,j) ← i;
        if (i = j) then
            aii ← 0; before(i,i) ← 0;
    for (m = 1, n ) do
        for (i = 1, n) do
            for (j = 1, n ) do
                if (aij > aim + amj) then
                    aij ← aim + amj; before(i,j) ← before(m,j);
                    if (i = j și aij < 0) then
                        return "circuit negativ";
```

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

Drumuri de cost minim - Rezolvarea problemei P3

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

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

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

Notăm cu $U^m = (u_{ij}^m)_{1 \leq i, j \leq n}$ pentru $m \in \{0, 1, 2, \dots, n-1\}$, 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:

Graph Algorithms * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Rezolvarea problemei P3

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph 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 ( $m = \overline{0, n - 2}$ ) do
        for ( $i, j \in V$ ) do
             $u_{ij}^{m+1} = \min_{k \in V} (u_{ik}^m + a_{kj})$ ;
```

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

Drumuri de cost minim - Rezolvarea problemei P3

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

Înmulțirea (rapidă) a matricilor

Considerăm următorul “produs de matrici”

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

Operația \otimes este asociativă și este similară înmulțirii uzuale a matricilor.

Putem scrie $U^{m+1} = U^m \otimes A$ și, prin inducție, obținem

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

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

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

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

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

Algoritmus * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 următare

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

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$$ ($T >> 1$), 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

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

Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul de săptămâna următare

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

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

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

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

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 $\mathcal{O}(n + m)$?

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 următare

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

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

Fie

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

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

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

Exercițiu 4. (cont'd)

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

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

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

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

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

Exerciții pentru seminarul de săptămâna următare

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 (multimea nodurilor etichetate de algoritmul lui Dijkstra) de amândouă instanțele algoritmului ne oprim;
- returnează drumul de la s la u în G reunit cu inversul drumului de la t la u în G' .

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

Exerciții pentru seminarul de săptămâna următare

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

- Grafuri Algoritmiști - C. Croitoru - Grafuri Algoritmiști - C. Croitoru - Grafuri Algoritmiști - C.

Exercițiu 7. 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.

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

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

Exerciții pentru seminarul de săptămâna următare

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

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

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

Exercițiu 10. 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 *

Exerciții pentru seminarul de săptămâna următare

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

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

Arătați că $m \leq n$;

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

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

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

(b) Demonstrați că (a) este echivalentă cu

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

Algoritmica grafurilor - Cursul 5

Cuprins

1 Conexiune

- **Teorema lui Menger** - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
 - **p-conexiune**
 - **Teorema lui König**
 - **Teorema lui Hall**
 - **Teorema lui Dirac**

C. Croitor
Arbori

- **Elemente de bază**
 - **Generarea all arborii parțiali**

3 Exercitii *

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

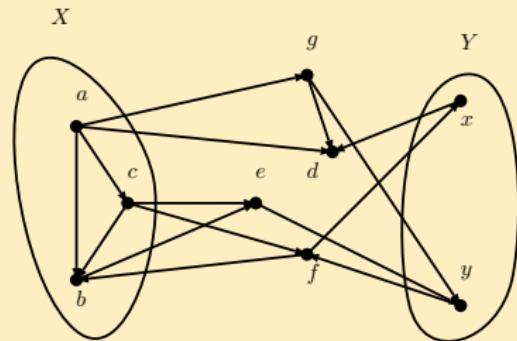
Conexiune - Teorema lui Menger și aplicații

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

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

- 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.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Conexiune - Teorema lui Menger și aplicații

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 $V(P) \cap Z \neq \emptyset$, pentru fiecare $P \in \mathcal{P}(X, Y; G)$.

Notăm cu

$$\mathbf{S}(X, Y; G) = \{Z : Z \text{ este mulțime } XY\text{-separatoare din } G\} \text{ și}$$

$$k(X, Y; G) = \min \{|Z| : Z \in \mathbf{S}(X, Y; G)\}$$

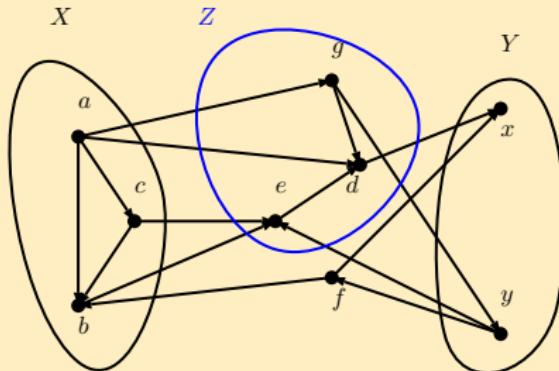
Din definiție urmează că:

- Dacă $Z \in \mathbf{S}(X, Y; G)$, atunci $\mathcal{P}(X, Y; G \setminus Z) = \emptyset$.
- $X, Y \in \mathbf{S}(X, Y; G)$. Iată

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



- 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 $T \in \mathbf{S}(X, Y; G)$.

Conexiune - Teorema lui Menger și aplicații

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 mulțimi XY -separatoare.)

Demonstrație:

$k(X, Y; G) \geq p(X, Y; G) = p$. Fie P_1, \dots, P_p XY -drumuri disjuncte din G ; $Z \cap V(P_i) \neq \emptyset, \forall Z \in S(X, Y; G)$. Deoarece P_i sunt disjuncte ($i = \overline{1, p}$):

$$|Z| \geq \left| Z \cap \left(\bigcup_{i=1}^p V(P_i) \right) \right| = \sum_{i=1}^p |Z \cap V(P_i)| \geq \sum_{i=1}^p 1 = p.$$

Astfel, $|Z| \geq p, \forall Z \in S(X, Y; G)$; urmează că $k(X, Y; G) \geq p$.

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

$k(X, Y; G) \leq p(X, Y; G) = p$. 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 $V(P_1) \cap V(P_2) = \{s, t\}$:

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

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 .

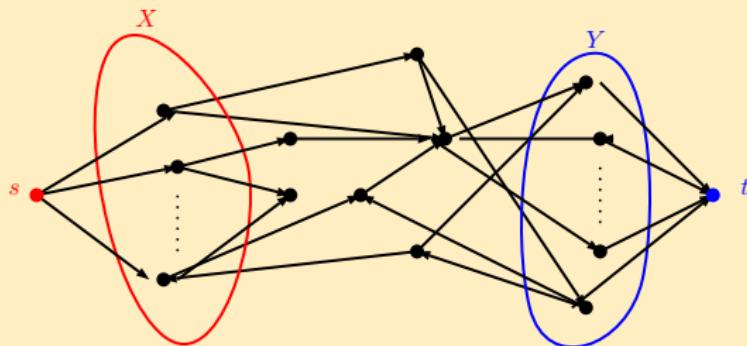
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Conexiune - Teorema lui Menger și aplicații

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

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

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

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 .
- Urmează că, pentru a calcula $k(G)$ - **numărul de conexiune pe noduri** al grafului G , trebuie aflat

$$\min_{st \notin E(G)} p(\{s\}, \{t\}; G),$$

care poate fi determinat în timp polinomial folosind fluxuri în rețele.

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

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 și X , Y 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

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 **transversal** 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.

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

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, $|\mathcal{A}(J)| \geq |r_{\mathcal{A}}(J)| \geq |J|$.

" \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

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

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 $G_{\mathcal{A}}$: urmărește 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

$$|X| \geq |I'| + |\mathcal{A}(I \setminus I')| \geq |I'| + |I \setminus I'| = |I|. \square$$

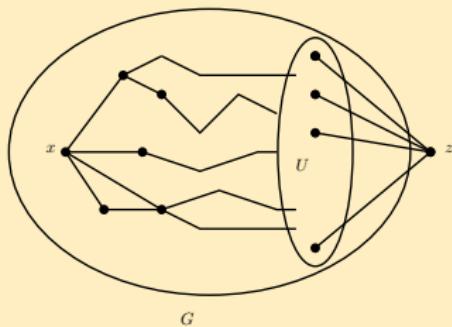
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

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

Aplicații: Teorema lui Dirac (structura grafurilor p -conexe)

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

Demonstrație (continuare). Atunci, G' este un graf p -conex. Într-adevăr, fie $A \subseteq V(G')$ cu $|A| \leq p - 1$. Dacă $A \subseteq V(G)$, atunci $G' - A$ este conex (din k -conexiunea lui G , $G - A$ este conex; cum $|A| < p$, $\exists u \in U \setminus A$ și, astfel, există $zu \in E(G' - A)$). Dacă $z \in A$, atunci $G' - A = G - A$ care este conex.

Lema urmează aplicând Teorema 2 grafului G' și perechii x, z . \square

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

Propoziție

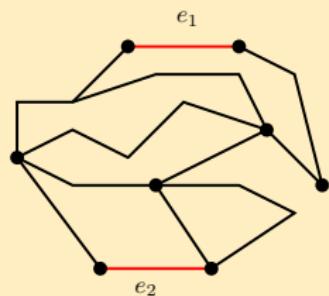
Fie $G = (V, E)$ un graf p -conex, $p \geq 2$. Atunci, pentru orice două muchii e_1 și e_2 ale lui G și pentru orice, x_1, \dots, x_{p-2} , $p - 2$ noduri ale lui G , există un circuit în G care conține toate aceste muchii și noduri.

Conexiune - Teorema lui Menger și aplicații

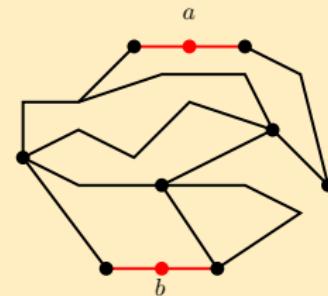
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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



G'

G' este 2-conex (orice graf de tipul $G' - v$ este conex). Astfel, există două drumuri intern-disjuncte de la a la b , care dă circuitul din G conținând e_1 și e_2 .

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 multime 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 multimii $\{x_1, x_2, \dots, x_{p-2}\}$ (altfel, aplicăm ipoteza inductivă și obținem că în graful $(p - 1)$ -conex, G , există un circuit C conținând e_1 , e_2 și multimea 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).

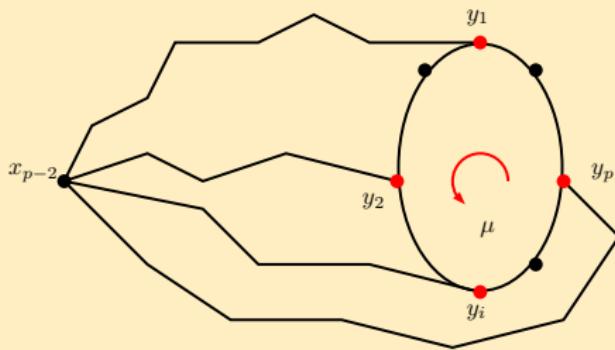
Graful $G - x_{p-2}$ este $(p - 1)$ -conex. Din ipoteza inductivă, există un circuit μ care conține x_1, x_2, \dots, x_{p-3} , e_1 și e_2 . Fie Y mulțimea nodurilor lui μ . Evident, $|Y| \geq p$ (mulțimii de $p - 3$ noduri x_1, x_2, \dots, x_{p-3} , îi adăugăm cel puțin trei extremități ale muchiilor 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} .

Conexiune - Teorema lui Menger și aplicații

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

Demonstrație (continuare). Fie $P_{x_{p-2}y_1}, P_{x_{p-2}y_2}, \dots, P_{x_{p-2}y_p}$ aceste drumuri, 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 drumurile de mai sus nu conține în interior niciun element din multimea $x_1, x_2, \dots, x_{p-3}, e_1$ 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

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

Teorema 5

(Dirac, 1953) Prin orice $p \geq 2$ noduri ale unui graf p -conex trece un circuit.

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

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

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

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.

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

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 $v \in V(G) \setminus V(C) \neq \emptyset$.

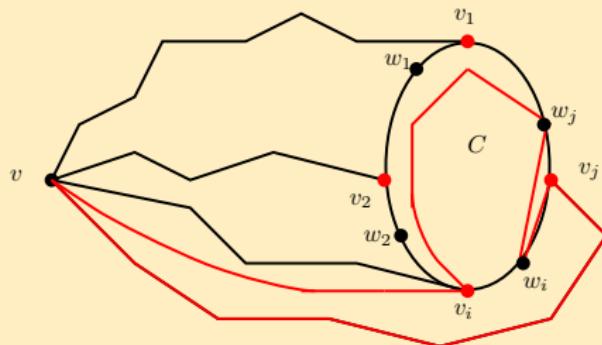
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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.



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
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 $\alpha(G) \leq p$, multimea $\{v, w_1, w_2, \dots, w_p\}$ 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). □

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

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

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:

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 Algorithms

(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.

Definiție

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 un drum P în T . Deoarece $E' \subseteq E$, P este un drum și în G , deci G este conex.

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

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 $T_G \neq \emptyset$) se bazează pe observația că există o muchie în cross între cele două clase ale oricărei bipartiții 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 $G = (V, E)$:

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

$k \leftarrow 1; T_1 \leftarrow (\{v\}, \emptyset); // v \in V$

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:

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

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 $e \in E$.

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

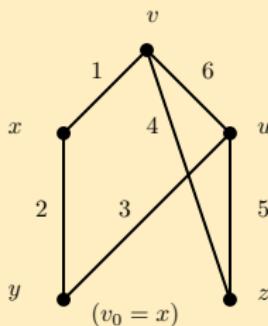
Demonstrație: Omisă. \square

Graph Algorithms * C. Croitoru - Graph Algorithms *

Arbore - Generarea \mathcal{T}_G

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

- Descriem o metodă simplă de tip **backtracking** pentru a genera toți arborii parțiali ai unui graf conex $G = (V, E)$, unde $V = \{1, \dots, n\}$, $|E| = m$.
- Mulțimea 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

Arbore - Generarea \mathcal{T}_G

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 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.

Arbore - Generarea T_G

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

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

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

- 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, $\{T[1], \dots, T[i-1]\} = \emptyset$).

- 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 păduri, 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
         $S \leftarrow S \cup \{u\}; root[u] \leftarrow root[w];$ 
```

Arbore - Generarea T_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.

- Cu alte cuvinte toate nodurile din arborele cu rădăcina x sunt adăugate arborelui cu rădăcina $\text{root}[w]$; aceste noduri sunt salvate în multimea S .
- După apelul $\text{All-ST-Gen}(i + 1)$, vector root trebuie setat din nou la valoarea dinaintea apelului, aceasta poate fi făcută **B** prin:

```
for ( $u \in S$ ) do  
     $\text{root}[u] = x;$ 
```

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

Arbore - Numărarea arborilor parțiali

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 matrcii $L[G]$ obținut prin stergerea liniei i și a coloanei j .

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:

$$L[K_n] = \begin{pmatrix} n-1 & -1 & \dots & -1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{pmatrix}.$$

Arbore - Numărarea arborilor parțiali

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

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

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

$$\begin{vmatrix} 1 & 1 & \dots & 1 \\ -1 & n-1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ -1 & -1 & \dots & n-1 \end{vmatrix} = n^{n-2}. \text{ (De ce?)}$$



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

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 $A = \{v\}$ satisfac 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 satisfac (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) Deduceți că K_n ($n \in \mathbb{N}^*$) sunt singurele grafuri regulate, triangulate și conexe.

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

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 inducă sau este muchia mediană a unui P_4 inducă.

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.

Exerciții pentru seminarul de săptămâna viitoare

Exercițiu 6. Arătați că un graf este 2-muchie conex dacă și numai dacă G poate fi orientat astfel ca graful orientat rezultat să fie tare conex.

Exercițiu 7.

- (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 8.

- (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

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

Exercițiul 9. Fie $G = (V, E)$ un graf conex și $u, v \in V$ două noduri distincte ale lui G . O submulțime de noduri X se numește **uv -separatoare minimală** dacă a u și v se află în componente conexe diferite ale lui $G - X$, dar pentru orice $X' \subsetneq X$, u și v sunt în aceeași componentă a lui $G - X'$.

- Dovediți că $X \subseteq V$ este muțime uv -separatoare minimală dacă și numai dacă u și v se află în componente diferite ale lui $G - X$, iar orice nod din X are vecini în ambele componente.
- Dacă X_1 și X_2 sunt două mulțimi uv -separatoare minimale din G astfel încât X_1 intersectează cel puțin două componente din $G - X_2$, atunci X_1 intersectează componentele lui $G - X_2$ care conțin pe u și v .

- 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

Exercițiul 10. Pentru un graf conex G dat aplicăm următorul algoritm:

$\mathcal{Q} \leftarrow \{G\}$; // \mathcal{Q} este o coadă;

while ($\mathcal{Q} \neq \emptyset$) {

$H \leftarrow \text{pop}(\mathcal{Q})$;

 fie $A \subseteq V(H)$ o mulțime de articulație minimală din H ;

 fie G_1, \dots, G_k componentele conexe ale lui $H - A$;

 pentru ($j = 1$ to k)

$\text{push}(\mathcal{Q}, [A \cup V(G_j)]_G)$;

}

Observăm că dacă G este un graf complet, atunci în \mathcal{Q} nu se mai adaugă vreun alt graf.

- Arătați că orice graf adăugat în \mathcal{Q} este conex.
- Dovediți că numărul total de grafuri adăugate la coada \mathcal{Q} este cel mult $|G|^2$.

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 11. Fie $G = (V, E)$ un graf conex și T_1, T_2 doi arbori parțiali ai lui G ($T_1, T_2 \in \mathcal{T}_G$).

- 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.
- 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.

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

Exercițiu 12. Demonstrați că mulțimea de muchii a unui graf complet K_n ($n \geq 2$) poate fi partionată în $\lceil n/2 \rceil$ submulțimi fiecare reprezentând mulțimea de muchii ale unui arbore (subgraf al lui K_n).

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

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

Exercițiu 13.

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 $T \cong [A]_{G_n}$.

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

Algoritmica grafurilor - Cursul 6

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

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

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 arborelui parțial 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
- 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 $c(T) = \sum_{e \in E(T)} c(e).$

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

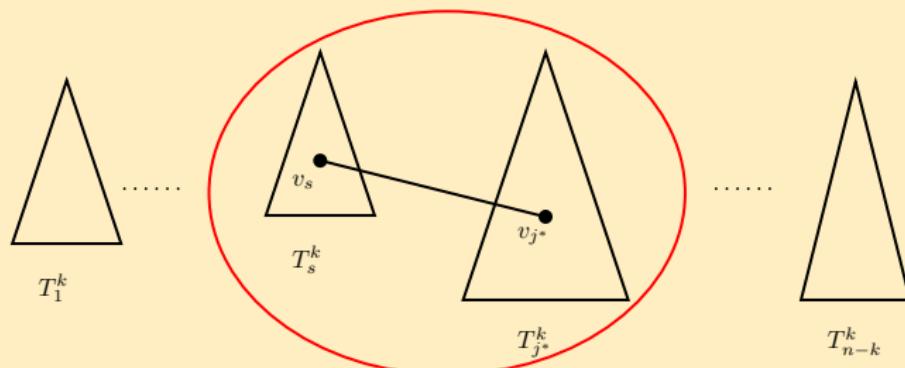
- Se pornește cu familia $\mathcal{T}^0 = (T_1^0, T_2^0, \dots, T_n^0)$ de arbori disjuncti: $T_i^0 = (\{i\}, \emptyset)$, $i = \overline{1, n}$.
- La fiecare pas k ($0 \leq k \leq n - 2$), din familia $\mathcal{T}^k = (T_1^k, T_2^k, \dots, T_{n-k}^k)$ de $n - k$ arbori disjuncti astfel încât $V = \bigcup_{i=1}^{n-k} V(T_i^k)$ 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^* .

Metoda generală MST

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

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} .

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 (roșie) este adevărată.

Metoda generală MST

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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). Dacă proprietatea **roșie** are loc pentru $0 \leq k \leq n - 2$, atunci există un MST al lui G , T_k^* , astfel încât $E(T^k) \subseteq E(T_k^*)$. Din construcție, $E(T^{k+1}) = E(T^k) \cup \{e^*\}$. Dacă $e^* \in E(T_k^*)$, atunci luăm $T_{k+1}^* = T_k^*$ și proprietatea are loc și pentru $k + 1$.

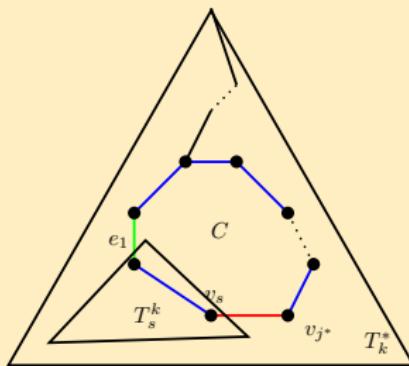
Să presupunem că $e^* \notin E(T_k^*)$. Atunci, $T_k^* + e^*$ are exact un circuit C , conținând $e^* = v_s v_{j^*}$. Deoarece $v_{j^*} \notin V(T_s^k)$, urmează că există o muchie $e_1 \neq e^*$ în C cu o extremitate în $V(T_s^k)$ și cealaltă în $V \setminus V(T_s^k)$. Din modul de alegere al e^* avem $c(e^*) \leq c(e_1)$ și $e_1 \in E(T_k^*) \setminus E(T^k)$. Fie $T^1 = T_k^* + e^* - e_1$; evident, $T^1 \in \mathcal{T}_G$ (fiind conex cu $n - 1$ muchii). Deoarece $e_1 \in E(T_k^*) \setminus E(T^k)$, avem $E(T^{k+1}) \subseteq E(T^1)$.

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 *

Metoda generală MST

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

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

- 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

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

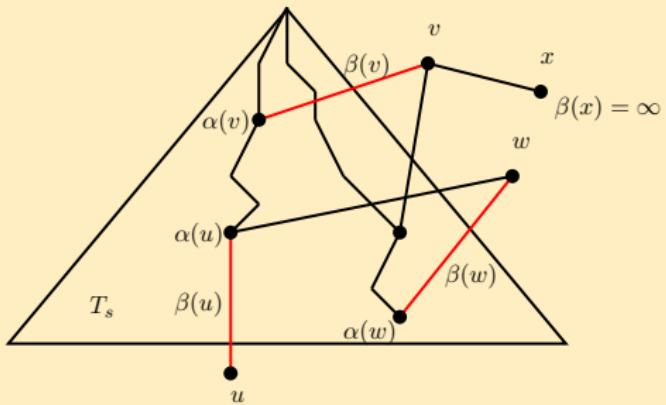
Algoritmul lui Prim

```
 $V_s \leftarrow \{s\}; E_s \leftarrow \emptyset; //$  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. i.  $\beta[j^*] = \min_{j \in V \setminus V_s} \beta[j];$   
   $V_s \leftarrow V_s \cup \{j^*\}; E_s \leftarrow E_s \cup \{\alpha[j^*]j^*\};$   
  for ( $j \in V \setminus V_s$ ) do  
    if ( $\beta[j] > c[j^*j]$ ) then  
       $\beta[j] \leftarrow c[j^*j]; \alpha[j] \leftarrow j^*;$ 
```

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

Remarci



- Se observă că (S) este satisfăcută după inițializare. 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 $\mathcal{O}(n^2)$.

Algoritmul lui Kruskal

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

- 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 diferenți ai familiei.
- Această alegere poate fi făcută prin sortarea inițială a muchiilor descrescător 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 ($(T \cup \{e_i\})_G$ nu are circuite) **then**

$T \leftarrow T \cup \{e_i\}$;

i ++;

Algoritmul lui Kruskal

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

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

- 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** .

Algoritmul lui Kruskal – Union-Find

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

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

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

- În metoda generală este necesară o reuniune (disjunctă) a mulțimilor de noduri a doi arbori (pentru a obține T^{k+1}).
- Folosim procedura $\text{union}(u, w)$ 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 ( $\text{find}(v) \neq \text{find}(w)$ ) then
         $\text{union}(v, w)$ ;
         $T \leftarrow T \cup \{e_i\}$ ;
     $i++$ ;
```

Algoritmul lui Kruskal – Union-Find – Prima soluție

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

*CCS în cursuri *CCS în cursuri *CCS în cursuri

- Tabloul $\text{root}[1..n]$ cu elemente din V are semnificația: $\text{root}[v] =$ rădăcina arborelui care reține mulțimea căreia îi aparține v .
- Adaugă la pasul de inițializare (correspunzând familiei \mathcal{T}^0):

```
for (v ∈ V) do
    root[v] ← v;
```
- Funcția find (cu complexitatea timp $\mathcal{O}(1)$):

```
function find(v : V);
    return root[v];
```
- Procedura union (cu complexitatea timp $\mathcal{O}(n)$):

```
procedure union(v, w : V);
    for (i ∈ V) do
        if (root[i] = root[v]) then
            root[i] = root[w];
```

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

Algoritmul lui Kruskal – Union-Find – Prima 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

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, timp 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.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 doua soluție

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

- 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 multimea căruia îi aparține v .
- Adaugă la pasul de inițializare (corresponzând familiei \mathcal{T}^0):

```
for (v ∈ V) do
    pred[v] ← 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 ← v; // o variabilă locală.
    while (pred[i] > 0) do
        i ← pred[i];
    return i;
```

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

Algoritmul lui Kruskal – Union-Find – A doua 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 -

- Procedura *union* (de complexitate timp $\mathcal{O}(1)$) este apelată doar pentru noduri rădăcină:

```
procedure union(root1, root2 : V)  
  pred[root1] ← root2;
```

- Bucla *while* a algoritmului este modificată astfel:

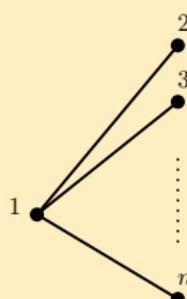
```
while (i ≤ m) do  
  let ei = vw; x ← find(v); y ← find(w);  
  if (x ≠ y) then  
    union(x, y);  
    T ← T ∪ {ei};  
  i ++;
```

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

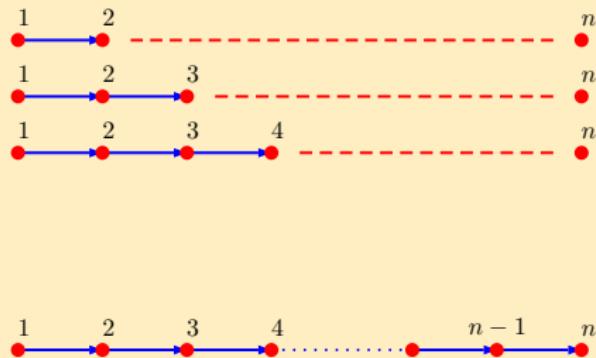
Algoritmul lui Kruskal – Union-Find – A doua soluție

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**):



$F(1), F(2), U(1, 2)$
 $F(1), F(3), U(2, 3)$
 $F(1), F(4), U(3, 4)$
 \vdots
 $F(1), F(n), U(n-2, n)$



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 Algorithms * C. Croitoru - Graph Algorithms *

Algoritmul lui Kruskal – Union-Find – A doua soluție

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

- 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

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

- 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  
         $pred[root_1] \leftarrow root_2$ ;  $pred[root_2] \leftarrow t$ ;
```

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 doua soluție

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

Afirmație. 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 iterări 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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 această iterație **while** avem un apel al procedurii **union**. Fie $\text{union}(x, y)$ 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. □

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 doua 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

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 datorată șirului 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.  
     $i \leftarrow v$ ;  
    while ( $pred[i] > 0$ ) do  
         $i \leftarrow pred[i]$ ;  
     $j \leftarrow v$ ;  
    while ( $pred[j] > 0$ ) do  
         $aux \leftarrow pred[j]$ ;  $pred[j] \leftarrow i$ ;  $j \leftarrow aux$ ;  
    return  $i$ ;
```

Algoritmul lui Kruskal – Union-Find – A treia soluție

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

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, și care are 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

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 (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 , $\langle A \rangle_G$.

- C. Croitoru - Grafiii Algoritmici * C. Croitoru - Grafiii Algoritmici * C. Croitoru - Grafi

Definiție

Un **cuplaj (mulțime independentă de muchii)** în G este o mulțime 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 :

$$\mathcal{M}_G = \{M : M \subseteq E, M \text{ cuplaj în } G\}.$$

- Grafiii Algoritmici * C. Croitoru - Grafiii Algoritmici * C. Croitoru - Grafi

Se observă că \mathcal{M}_G satisfacă:

- (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 Algorithms * C. Croitoru - Graph Algorithms *

Acoperire minimă cu muchii

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

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 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

$$|M^*| + |F^*| = n.$$

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

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. î.  $vv' \in E$ ;
     $F \leftarrow F \cup \{vv'\}$ ;
```

Cuplaje maxime – Acoperire minime cu muchii

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

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

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).

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

Cuplaje maxime – Acoperire minime cu muchii

Remarcă

Să observăm că tocmai am demonstrat că două probleme **P₁** și **P₂** sunt polinomial echivalente deoarece cuplajul M și acoperirea cu muchii F construite sunt soluții optime pentru cele două probleme, respectiv.

Exerciții pentru seminarul din săptămâna următoare

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 bipartiție (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 $c : R \rightarrow \mathbb{R}$. 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

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

$$s[T(H, A, w)] = \sum_{e \in E_T} w(e) = \\ = \min \left\{ \sum_{e \in E_{T'}} w(e) : T' = (V_{T'}, E_{T'}) \text{ arbore în } H, A \subseteq V_{T'} \right\}$$

- (a) Arătați că un arbore Steiner poate fi determinat în timp polinomial dacă $A = V(H)$ sau $|A| = 2$.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul din săptămâna următoare

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

Exercițiu 5 (continuare).

- (b) Fie $G = (V, E)$ un graf conex cu $V = \{1, 2, \dots, n\}$, și $A \subseteq V$; avem de asemenei și o funcție de cost $c : E \rightarrow \mathbb{R}_+$. Considerăm graful complet K_n (cu $V(K_n) = V$) și definim $\bar{c} : E(K_n) \rightarrow \mathbb{R}_+$:

$$\bar{c}(ij) = \min \left\{ c(P) = \sum_{e \in E(P)} c(e) : P \text{ este } ij\text{-drum în } G \right\}$$

Demonstrați că $s[T(G, A, c)] = s[T(K_n, A, \bar{c})]$ și arătați cum se poate construi un arbore Steiner $T(K_n, A, \bar{c})$ dintr-un arbore Steiner $T(G, A, c)$.

- (c) Arătați că există un arbore Steiner $T(K_n, A, \bar{c})$ astfel încât toate nodurile sale din afara lui A au gradul cel puțin 3. Folosind această proprietate arătați că există un arbore Steiner $T(K_n, A, \bar{c})$ cu cel mult $2|A| - 2$ noduri.

Exerciții pentru seminarul din săptămâna următoare

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

Exercițiu 6. 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 : $\mathbf{x}_i^A = 1 \Leftrightarrow e_i \in A$. 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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 7. Fie $G = (V, E)$ un graf conex și $c : E \rightarrow \mathbb{R}$ o funcție de cost pe muchiile sale.

a) Fie T^* un arbore parțial de cost c -minim al lui G și $\epsilon > 0$. Arătați că T^* este singurul arbore parțial de cost \bar{c} -minim al lui G , unde

$$\bar{c}(e) = \begin{cases} c(e) - \epsilon, & \text{dacă } e \in E(T^*) \\ c(e), & \text{altfel} \end{cases}$$

b) Deducreți de aici că pentru orice arbore parțial de cost minim, T^* , al lui G există o ordonare a muchiilor lui G astfel încât algoritmul lui Kruskal returnează T^* .

Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul din săptămâna următoare

Exercițiul 8. 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 .

Exercițiul 9. Determinați numărul de cuplaje maxime ale următorului graf:



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

Exercițiu 10. Doi copii se joacă pe un graf dat, G , astfel: fiecare alege alternativ un nod nou v_0, v_1, \dots aşa încât, pentru orice $i > 0$, v_i este adiacent 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.

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. 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 $r_{\mathcal{A}}, r_{\mathcal{B}} : \{1, 2, \dots, k\} \rightarrow S$ aşa 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 *

Algoritmica Grafurilor - Cursul 7

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 Cuplaje

- Formularea analitică a problemei cuplajului maxim
- Cuplaje perfecte - Teorema lui Tutte
- Cuplaje de cardinal 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 din săptămâna a 11-a

Formularea analitică problemei cuplajului maxim

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 \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 câte 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 $\mathbf{1}_p$ este vectorul p -dimensional cu toate componente 1, atunci problema **P₁** poate fi descrisă echivalent astfel:

Formularea analitică a problemei cuplajului maxim

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

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

$$\mathbf{P}'_1 \max \left\{ \mathbf{1}_m^T \mathbf{x} : B\mathbf{x} \leqslant \mathbf{1}_n, \mathbf{x} \geqslant 0, x_i \in \{0, 1\}, i = \overline{1, m} \right\}.$$

\mathbf{P}'_1 este o problemă ILP (Integer Linear Programming), care este în general NP-hard. Vom încerca să rezolvăm \mathbf{P}'_1 , folosind problema (relaxată) asociată LP (Linear Programming)

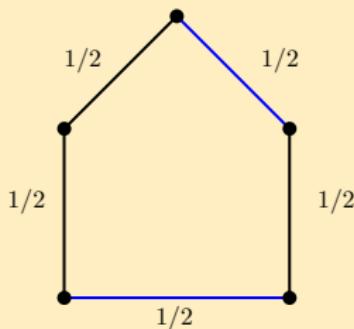
$$\mathbf{LP}'_1 \max \left\{ \mathbf{1}_m^T \mathbf{x} : B\mathbf{x} \leqslant \mathbf{1}_n, \mathbf{x} \geqslant 0 \right\}.$$

Soluțiile optime ale lui \mathbf{LP}'_1 pot avea componente ne-întregi și de asemenea 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 \mathbf{LP}'_1 cu valoarea optimă $n + 1/2 > n$.

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

Formularea analitică a problemei cuplajului maxim

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



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 componentele din $\{0, 1/2, 1\}$. Componentele 1/2 apar dacă și numai dacă G are circuite impare.

Formularea analitică a problemei cuplajului maxim

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

- 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 ungără" 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:

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

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

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 , $\nu(G) = n - \alpha(G)$.

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

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 .

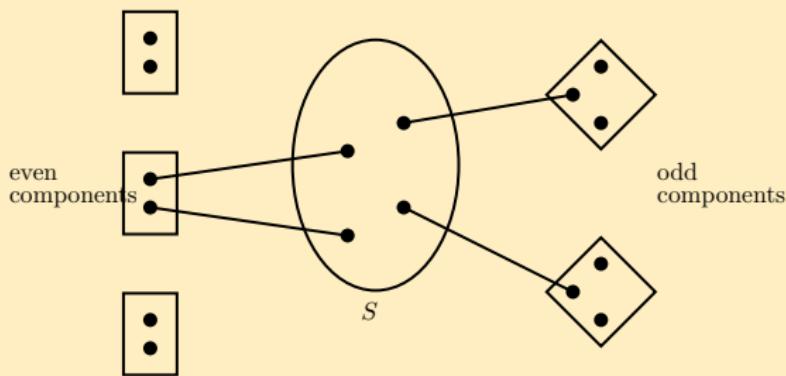
Remarcă

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 muchiilor diferite sunt distințte, urmează că $|S| \geq q(G - S)$.

Pentru $S = \emptyset$ în (T), obținem $q(G - \emptyset) \leq 0$, deci $q(G) = 0$.

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



Teorema 4

(Tutte, 1947) Un graf G admite cuplaj perfect dacă și numai dacă

$$(T) \quad q(G - S) \leq |S|, \forall S \subseteq V(G).$$

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

Cuplaje perfecte - Teorema lui Tutte

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

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.

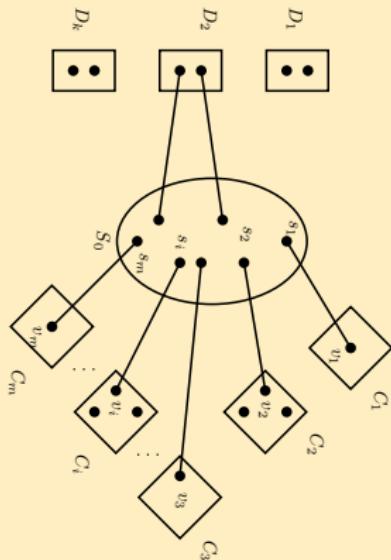
Pentru $n = 1, 2$ teorema are loc evident. În pasul inductiv, fie G un graf cu $n \geq 3$ noduri care satisfacă (T) și să presupunem că orice graf G' cu $|G'| < n$ și care satisfacă (T) are un cuplaj perfect.

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, 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).

Cuplaje perfecte - Teorema lui Tutte

Fie $m = |S_0| > 0$, C_1, C_2, \dots, C_m componentele impare ale lui $G - S_0$ și D_1, D_2, \dots, D_k componentele 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 $s_i \in S$ și celălalt $v_i \in C_i$ ($i = 1, \overline{1, m}$);
 - c) cuplaj perfect în fiecare subgraph $C_i - v_i$ ($i = 1, \overline{1, m}$).
- a) Pentru orice $1 \leq i \leq k$ graful $[D_i]_G$ admite cuplaj perfect. Într-adevăr, deoarece $m > 0$, urmează că $|D_i| < n$ și din ipoteza inductivă 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:

$$q(G - (S_0 \cup S')) = q(G - S_0) + q(G' - S') = |S_0| + q(G' - S') > |S_0 \cup S'|.$$

Astfel, $q(G' - S') \leq |S'|$, $\forall S' \subseteq D_i$, i. e., G' satisfacă (T).

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

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 graf 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 $|N_H(A)| = |B| \geq |A|$.

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 inductivă, este suficient să dovedim că G' satisfacă (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

$$\begin{aligned} |S''| &\geq q(G - S'') = q(G - S_0) - 1 + q(G' - S') = |S_0| - 1 + q(G' - S') \geq \\ &\geq |S_0| - 1 + |S'| + 2 = |S''|, \end{aligned}$$

i. e., $q(G - S'') = |S''|$, în contradicție cu alegerea lui S_0 (deoarece $S_0 \subsetneq S''$).

Cuplaje perfecte - Teorema lui Tutte

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.

Cuplaje de cardinal maxim - Teorema lui Berge

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

$$P : v_0, v_0v_1, v_1, \dots, v_{k-1}, v_{k-1}v_k, v_k$$

astfel încât $\{v_{i-1}v_i, v_iv_{i+1}\} \cap M \neq \emptyset, \forall i = \overline{1, k-1}$.

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

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 (muchiiile 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.

Cuplaje de cardinal maxim - Teorema lui Berge

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

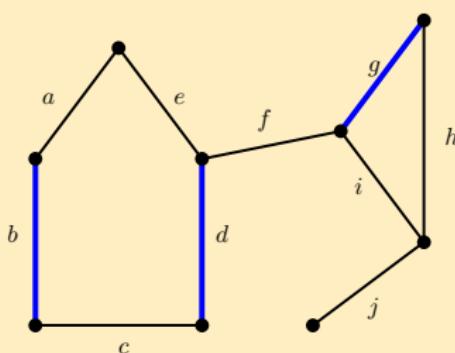
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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 .

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

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 - augmenting path

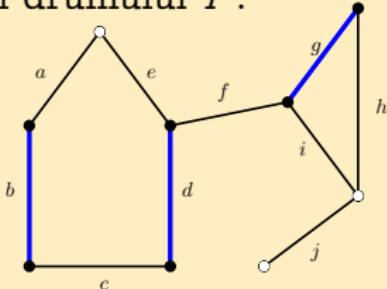
Cuplaje de cardinal maxim - Teorema lui Berge

Teorema 5

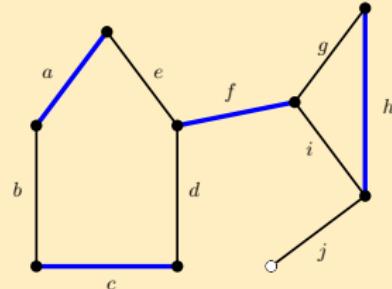
(Berge, 1959) M este un cuplaj de cardinal maxim în graful G dacă și numai dacă nu există drum de creștere în G relativ la M .

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



$M \Delta P$

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

Mai mult, $|M'| = |P \cap M| + 1 + |M \setminus P| = |M| + 1$, în contradicție cu alegerea lui M .

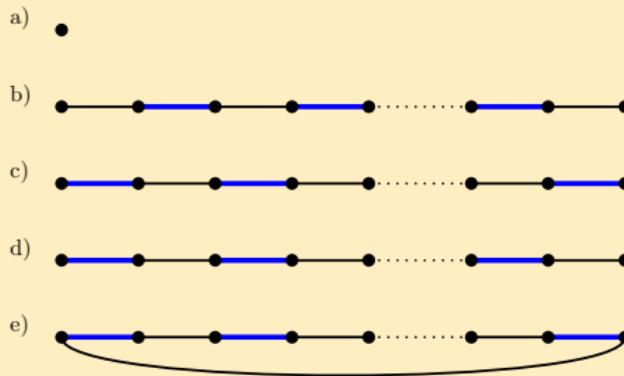
" \Leftarrow " Fie M un cuplaj în G cu proprietatea că nu există drumuri de creștere în G relativ la M .

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 .

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,

$$\begin{aligned}|M \setminus M^*| &= \sum_{C \text{ comp. a lui } G'} m_M(C) = \\&= \sum_{C \text{ comp. a lui } G'} m_{M^*}(C) = |M^* \setminus M|\end{aligned}$$

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

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 iterare **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)**.

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

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

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

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

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.

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

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 .

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

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

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 $|N| = s = \nu(G)$.

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

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

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

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

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** .

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

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 $M \Delta P$. Atunci $|P'| \geq |P| + 2|P \cap P'|$.

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

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**

let 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

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

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.

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

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

Graph Algorithms * C. Croitoru - Graph Algorithms *

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

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

$$|P_r| \leq 2\lceil r/(s-r) \rceil + 1 = 2\lceil \lceil s - \sqrt{s} \rceil / (s - \lceil s - \sqrt{s} \rceil) \rceil + 1 < 2\lfloor \sqrt{s} \rfloor + 1.$$

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

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 faza următoare va descrește (altfel, mulțimea drumurilor de creștere minime construite în faza 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 :

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

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

$M \leftarrow \emptyset;$

repeat

determină \mathcal{P} o familie maximală de drumuri de creștere minime relativ la M disjuncte pe noduri;

for ($P \in \mathcal{P}$) **do**

$M \leftarrow M \Delta P;$

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

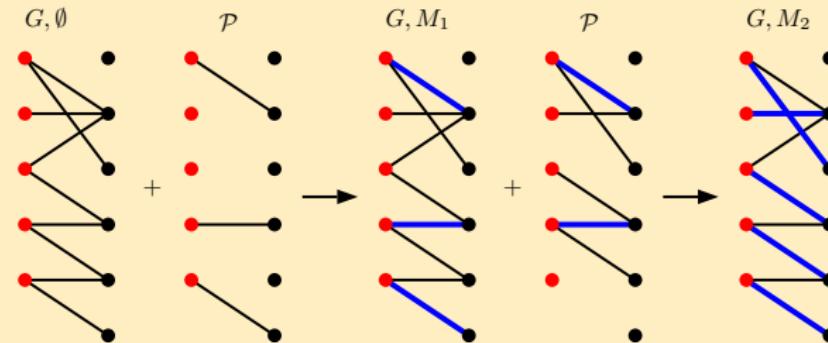
Acest rezultat a fost extins la grafuri arbitrară de către **Micali și Vazirani (1980)** folosind o structură de date elaborată pentru a întreține etichetele asociate nodurilor pentru construcția drumurilor de creștere minime.

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 liste de adiacență în $\mathcal{O}(n + m)$.

Detaliile sunt omise; un exemplu este dat pe slide-ul următor.

Cuplaje de cardinal maxim - Algoritmul Hopcroft-Karp

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 $c((i, j)) = c_{ij}$.

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

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 care ieșe)**.

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 $V \times V$, constrângerile (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 constrângerile (ii) (pentru $i \in V \setminus \{s, t\}$) obținem

$$\begin{aligned} 0 &= \sum_{i \neq s, t} \left(\sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} \right) = \sum_{i \neq s, t} \sum_{j \neq s, t} x_{ji} - \sum_{i \neq s, t} \sum_{j \neq s, t} x_{ij} + \\ &\quad + \sum_{i \neq s, t} x_{si} + \sum_{i \neq s, t} x_{ti} - \sum_{i \neq s, t} x_{is} - \sum_{i \neq s, t} x_{it} = \end{aligned}$$

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

Valoarea fluxului

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

Exerciții pentru seminarul din săptămâna a 11-a

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

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 $Y_i \subseteq X_i$, $|Y_i| = d_i$, $\forall i = \overline{1, n}$ dacă și numai dacă

$$\left| \bigcup_{i \in I} X_i \right| \geq \sum_{i \in I} d_i,$$

pentru orice $I \subseteq \{1, \dots, n\}$.

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

Exercițiu 2. Orice graf p -regular bipartit admite cuplaj perfect.

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

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 din săptămâna a 11-a

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-independentă* 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-independentă* au același cardinal).

Exercițiu 6. Fie $T = (V, E)$ un arbore cu rădăcină; îi notăm cu r rădăcina și cu $\text{parent}(v)$ 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 $w \in \text{parent}(v) \cap M$.

- (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 .

Algoritmica grafurilor - Cursul 8

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

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 a 11-a

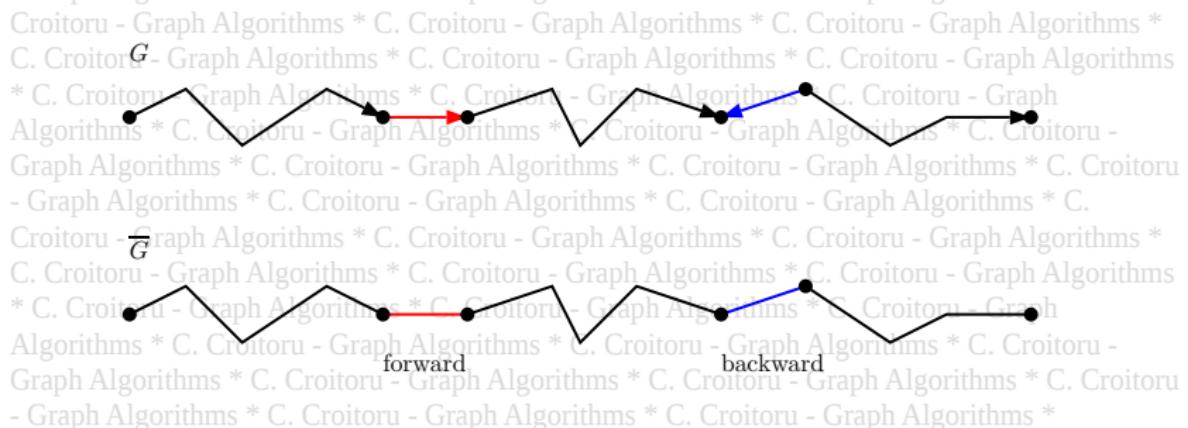
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru
- Graph Algorithms * C.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

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

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)** al lui P , altfel (e corespunde arcului $v_j v_i$) e este un **arc înapoi (backward)** al lui P .



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

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 \widetilde{G} astfel încât $\forall ij \in E(P)$:

- dacă ij este un arc forward, atunci $x_{ij} < c_{ij}$,
- dacă ij este un arc backward, 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

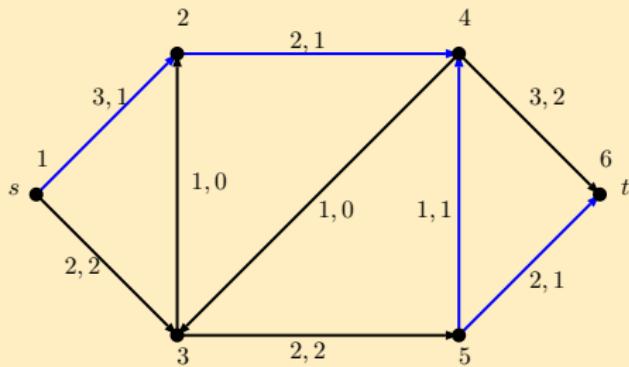
$$r(ij) = \begin{cases} c_{ij} - x_{ij}, & \text{dacă } ij \text{ este un arc forward al lui } P \\ x_{ji}, & \text{dacă } ij \text{ este un arc backward al lui } P \end{cases}$$

Capacitatea reziduală a drumului P este $r(P) = \min_{e \in E(P)} r(e)$.

Problema fluxului maxim - Drumuri de creștere

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 reiduală 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

$$x_{ij}^1 = \begin{cases} x_{ij}, & \text{dacă } ij \notin E(P) \\ x_{ij} + r(P), & \text{dacă } ij \in E(P), ij \text{ este arc forward în } P \\ x_{ij} - r(P), & \text{dacă } ij \in E(P), ij \text{ este arc backward în } P \end{cases}$$

este un flux în R cu $v(x^1) = v(x) + r(P)$.

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

Problema fluxului maxim - Drumuri de creștere

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

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 forward:

$$\begin{aligned} \sum_j x_{ji}^1 - \sum_j x_{ij}^1 &= \sum_{j \neq l} x_{ji} - \sum_{j \neq k} x_{ij} + x_{li}^1 - x_{ik}^1 \\ &= \sum_{j \neq l} x_{ji} - \sum_{j \neq k} x_{ij} + x_{li} + r(P) - x_{ik} - r(P) = \sum_j x_{ji} - \sum_j x_{ij} = 0. \end{aligned}$$

b) li este arc forward și ik arc backward:

$$\begin{aligned} \sum_j x_{ji}^1 - \sum_j x_{ij}^1 &= \sum_{j \neq l, k} x_{ji} - \sum_j x_{ij} + x_{li}^1 + x_{ki}^1 \\ &= \sum_{j \neq l, k} x_{ji} - \sum_j x_{ij} + x_{li} + r(P) + x_{ki} - r(P) = \sum_j x_{ji} - \sum_j x_{ij} = 0. \end{aligned}$$

Problema fluxului maxim - Drumuri de creștere

- c) li arc backward și ik arc forward: similar cu b).
- d) li și ik arce backward: similar cu a).

$v(x^1)$ diferă de $v(x)$ datorită fluxului de pe arcul lt din P :

lt arc forward:

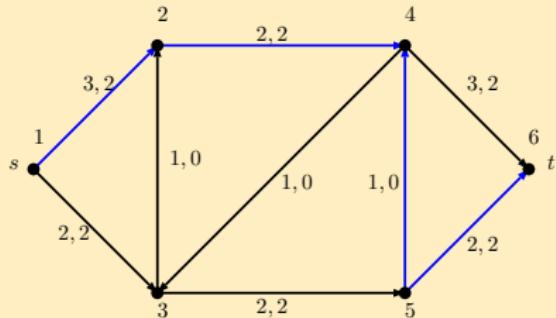
$$\begin{aligned}v(x^1) &= \sum_j x_{jt}^1 - \sum_j x_{tj}^1 = \sum_{j \neq l} x_{jt} - \sum_j x_{tj} + x_{lt}^1 = \\&= \sum_{j \neq l} x_{jt} - \sum_j x_{tj} + x_{lt} + r(P) = v(x) + r(P).\end{aligned}$$

lt arc backward:

$$\begin{aligned}v(x^1) &= \sum_j x_{jt}^1 - \sum_j x_{tj}^1 = \sum_j x_{jt} - \sum_{j \neq l} x_{tj} - x_{tl}^1 = \\&= \sum_j x_{jt} - \sum_{j \neq l} x_{tj} - (x_{tl} - r(P)) = v(x) + r(P). \quad \square\end{aligned}$$

Problema fluxului maxim - Drumuri de creștere

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 partitie (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

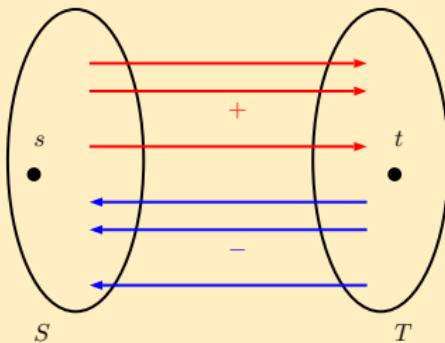
Dacă x este un flux în $R = (G, s, t, c)$ și (S, T) este o secțiune în această rețea, atunci

$$v(x) = \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji})$$

(valoarea fluxului este fluxul net care trece prin secțiune).

Problema fluxului maxim - Secțiuni

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



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

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

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

$$v(x) \leq c(S, T).$$

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 Algorithms

Demonstrație. Din Lema 2

$$v(x) = \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}) \stackrel{x_{ij} \leq c_{ij}}{\leq} \sum_{i \in S} \sum_{j \in T} (c_{ij} - x_{ji}) \stackrel{x_{ji} \geq 0}{\leq}$$

$$\stackrel{x_{ji} \geq 0}{\leq} \sum_{i \in S} \sum_{j \in T} c_{ij} = c(S, T). \square$$

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

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 .

Problema fluxului maxim - Teorema drumului 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
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). Urmează 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 remarca 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).

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
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 iterăție **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

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

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

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 .

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

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 capacitate raționale, algoritmul considerat în demonstrația Teoremei 2 satisface această condiție. Pentru capacitate arbitrară reale vom prezenta mai târziu un astfel de algoritm datorat lui **Edmonds și Karp (1972)**.

Problema fluxului maxim - Algoritmul 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 -

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.

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

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

Eticheta 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, $e_2 \in \{\text{forward}, \text{backward}\}$ 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:

Problema fluxului maxim - Algoritmul 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

```
procedure scan( $i$ )
for ( $j \in V$ , neetichetat) do
    if ( $ij \in E$  și  $x_{ij} < c_{ij}$ ) then
        etichetează  $j$  cu  $e = (i, forward, \min\{e_3[i], c_{ij} - x_{ij}\})$ ;
    end if
    if ( $ji \in E$  și  $x_{ji} > 0$ ) then
        etichetează  $j$  cu  $e = (i, backward, \min\{e_3[i], x_{ji}\})$ ;
    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

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

$r(P)$ 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 .

* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 - Algoritmul lui Ford & Fulkerson

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

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;

$\text{scan}(i)$;

if (t a fost etichetat) **then**

modifică fluxul pe drum dat de etichete;

sterge toate etichetele; $e(s) \leftarrow (\cdot, \cdot, \infty)$;

end if

end while

$S \leftarrow \{i : i \in V, i \text{ este etichetat}\}$;

$T \leftarrow V \setminus S$;

x este un flux de valoare maximă, (S, T) este o secțiune de capacitate minimă.

Problema fluxului maxim - Algoritmul lui Ford & Fulkerson

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

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 $\mathcal{O}(nmU)$.

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

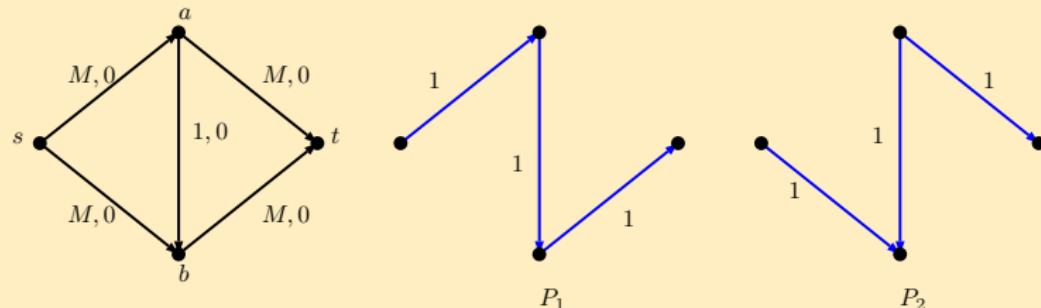
Remarcă

E posibil ca algoritmul să nu se termine pentru capacitați iraționale. Această situație nu apare în implementările practice, dar neajunsul acestei descrierii a algoritmului este dat de faptul că numărul de creșteri ale fluxului curent depinde de capacitați (și nu de dimensiunile rețelei).

Problema fluxului maxim - Algoritmul 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

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.

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

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

Aceste deficiențe ale algoritmului pot fi evitate dacă alegerea nodurilor etichetate în vederea scanării nu se face aleatoriu (**Dinic (1970), Edmonds & Karp (1972)**).

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

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 ;

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

Pentru a arăta că această secvență este finită, fie $\forall i \in V$ și $\forall k \in \mathbb{N}^*$:

σ_i^k = lungimea minimă a unui A -drum de la s la i relativ la x^k .

τ_i^k = lungimea minimă a unui A -drum de la i la t relativ la x^k .

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

Lema 4

$\forall i \in V$ și $\forall k \in \mathbb{N}^*$ avem

$$\sigma_i^{k+1} \geq \sigma_i^k \text{ și } \tau_i^{k+1} \geq \tau_i^k.$$

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

Demonstrație. Omisă.

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).

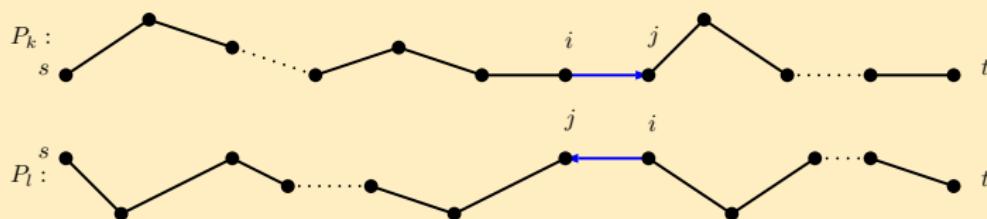
Demonstrație. Dacă P este un drum de creștere relativ la un flux x în R , cu capacitatea reziduală $r(P)$, un **arc critic** în P este un arc $e \in E(P)$ cu $r(e) = r(P)$. În $x \otimes r(P)$, fluxul de pe arce critice devine fie egal cu capacitatea (pe arcele forward), sau nul (pe arcele backward). 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 $\sigma_i^k + \tau_i^k = \sigma_j^k + \tau_j^k$

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

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

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:

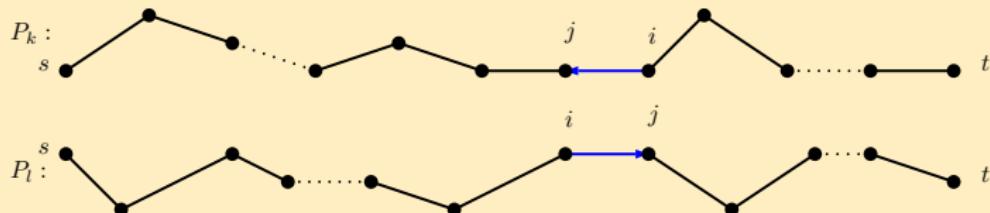
ij este un arc forward în P_k . Atunci $\sigma_j^k = \sigma_i^k + 1$; în P_l ij va fi arc backward, deci $\sigma_i^l = \sigma_j^l + 1$.



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 $length(P_l) \geq length(P_k) + 2$.

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

ij este un arc backward în P_k . Atunci $\sigma_i^k = \sigma_j^k + 1$; în P_l ij va fi arc forward, 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 $length(P_l) \geq length(P_k) + 2$.

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).

Modificarea lui Edmonds & Karp a algoritmului lui Ford & Fulkerson

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

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

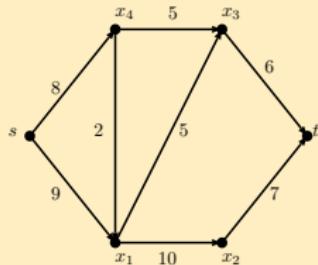
Demonstrație. Există $\mathcal{O}(mn)$ creșteri de flux (din Teorema 4), fiecare de complexitate $\mathcal{O}(m)$. □

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 Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul din săptămâna a 11-a

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

Exercițiu 1. Considerăm rețeaua de transport de mai jos:



Determinați un flux maxim în rețeaua de mai jos utilizând algoritmul lui Edmonds-Karp pentru următoarele ordonări ale listelor de adiacență:

- (a) $A(s) = (x_1, x_4)$, $A(x_1) = (s, x_3, x_2, x_4)$, $A(x_2) = (x_1, t)$,
 $A(x_3) = (x_1, x_4, t)$, $A(x_4) = (x_3, x_1)$.
- (b) $A(s) = (x_4, x_1)$, $A(x_1) = (s, x_2, x_3, x_4)$, $A(x_2) = (x_1, t)$,
 $A(x_3) = (x_1, x_4, t)$, $A(x_4) = (x_3, x_1)$.

Exerciții pentru seminarul din săptămâna a 11-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.

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 $\delta(P)$ cel puțin K - dacă un asemenea drum există - în $\mathcal{O}(m)$ time complexity.

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

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 $\delta(P) \geq K$)

$x \leftarrow x \otimes \delta(P);$

$K \leftarrow K/2;$

}

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. Digaful $G = (V, E)$ 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 încărcaării (static load balancing strategy) în G : indicați pentru fiecare procesor încărcarea ce trebuie trimisă și căruia procesor astă încât toate procesoarele să aibă aceeași încărcare.

Exerciții pentru seminarul din săptămâna a 11-a

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 $c : E \rightarrow \mathbb{Z}_+$, $n = |V|$, $m = |E|$.

- 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^* .
- 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.

Exerciții pentru seminarul din săptămâna a 11-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.

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.

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

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 *

Algoritmica Grafurilor - Cursul 9

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

1 **Fluxuri în rețele** * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **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 **Exerciții pentru seminarul din săptămâna a 12-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. Croitoru - Graph Algorithms *

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

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

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) $\forall i \neq s, e_i = \sum_{ji \in E} x_{ji} - \sum_{ij \in E} x_{ij} \geq 0.$

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

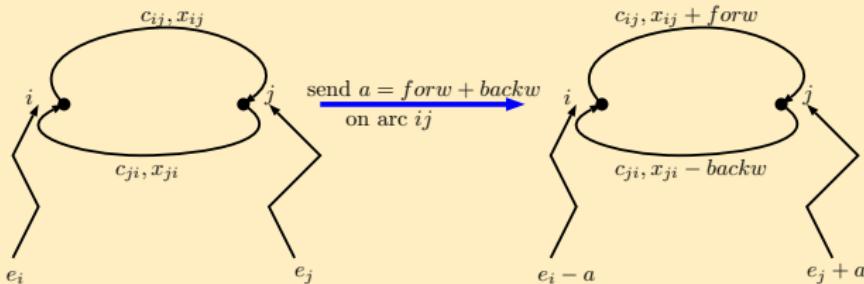
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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).



Prefluxuri

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

În cele de urmă, **a trimite 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. i.

$$(D_1) \quad d(t) = 0,$$

$$(D_2) \quad \forall ij \in E, r_{ij} > 0 \Rightarrow d(i) \leq d(j) + 1.$$

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

Remarci

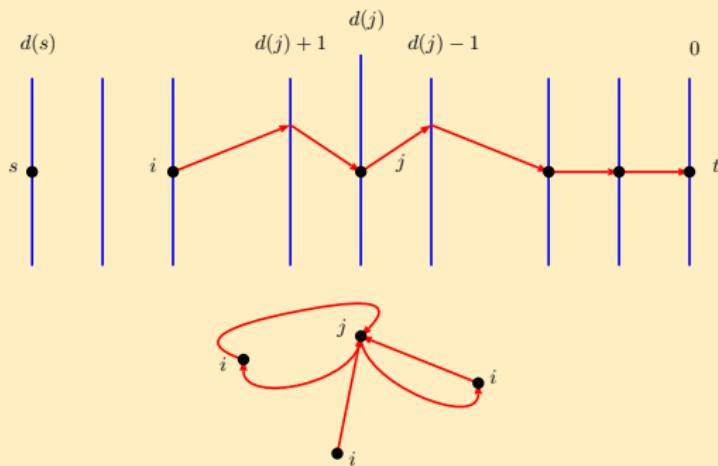
- Dacă P este un A -drum relativ la prefluxul x în R de la i la t , atunci $d(i) \leq \text{length}(P)$ (arcele lui P au capacitați reziduale pozitive și apoi se aplică (D_2) în mod repetat). Urmează că (τ_i este lungimea minimă a unui A -drum de la i la t): $d(i) \leq \tau_i$.

Prefluxuri

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

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 $d(i) = d(j) + 1$.

Prefluxuri

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

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 $r_{sj} = 0$, $\forall sj \in A(s)$.

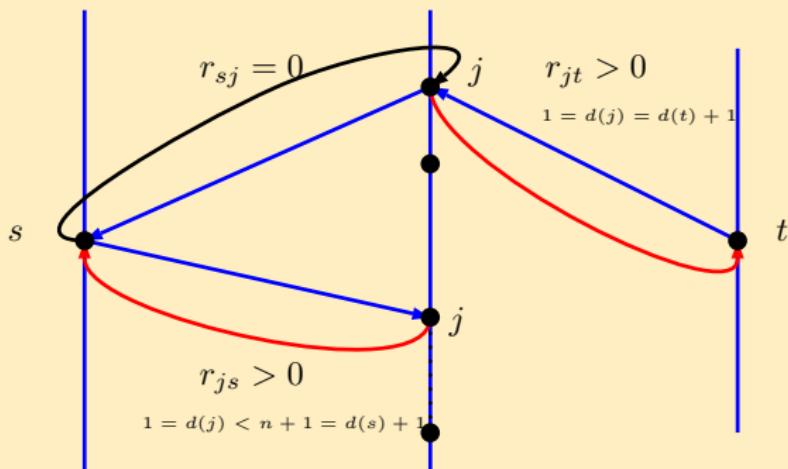
Astfel condiția (D_2) nu este nătărișată de $d(s) = n$.

Pentru toate arcele ij , (D_2) este îndeplinită:

$$d(s) = n$$

$$d(j) = 1$$

$$d(t) = 0$$



Prefluxuri

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

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

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

$relabel(i)$;

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

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.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Prefluxuri - Schema generală a unui algoritm de tip preflux

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

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 iterații **while**, atunci după această iterație cele două condiții sunt de asemenei satisfăcute. Avem două cazuri, în funcție de care dintre procedurile **push** sau **relabel** este executată în iterația **while** curentă:

este apelată $\text{push}(i)$: singura pereche care poate viola (D_2) este $d(i)$ și $d(j)$. Deoarece ij este admisibil, la apelul $\text{push}(i)$ avem $d(i) = d(j) + 1$. După execuția $\text{push}(i)$, 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 $\text{relabel}(i)$: actualizarea lui $d(i)$ este astfel încât (D_2) este satisfăcută pentru fiecare arc ij cu $r_{ij} > 0$. Deoarece $\text{relabel}(i)$ 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

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

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.

Prefluxuri - Schema generală a unui algoritm de tip preflux

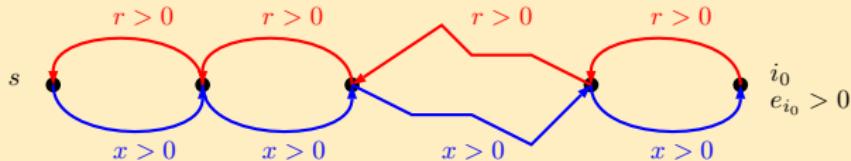
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

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. □

Graph Algorithms * C. Croitoru - Graph Algorithms *

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

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

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

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

Lema 3

(Goldberg și Tarjan, 1986). Numărul total de pompări nesaturate nu este mai mare decât $2n^2m$.

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

Demonstrație. Omisă.

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

Lema 4

Algoritmul de tip preflux returnează un flux de valoare maximă în R .

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

Demonstrație. Din Lemele 1 și 3 și din Corolarul 3 al Lemei 2, algoritmul se oprește în cel mult $2n^2m$ 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

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

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 $\mathcal{O}(n^2 \log U)$.

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

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 Algorithms

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:

- la începutul etapei p , $e_i \leq 2^p, \forall i \in V \setminus \{s, t\}$.
- 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 $d(i) \in \{1, 2, \dots, n-1\}$).
- nodul activ selectat pentru **push-relabel** în timpul etapei p va fi **primul nod din $L(p)$** . Urmează că, dacă se face o pompă (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 **pompă nesaturată** trimite cel puțin 2^{p-1} unități de flux.

După apelul $\text{push}(i)$ 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 înlesni parcurgerea.

Toate aceste liste sunt construite în $O(m)$, înaintea apelului procedurii **initialization**.

Prefluxuri - Algoritmul lui Ahuja & Orlin

```
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  $d[i] = \min \{ d[j] + 1 : ij \in A(i) \text{ și } r_{ij} > 0\}$   
    repozitionează  $i$  în  $L(p)$ ;  
    setează poziția curentă (a pointerului) la începutul listei  $A(i)$ ;
```

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 , $F(p) < \sum_{i \in V} \frac{2^p \cdot (2n)}{2^p} = 2n^2$.

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

Dacă, în etapa p , se execută **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 **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 $8n^2$ (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

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

În concluzie:

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

Teoremă

(Ahuja-Orlin, 1988) Algoritm de tip preflux cu scalarea exceselor are
complexitatea timp $\mathcal{O}(nm + n^2 \log U)$.

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

A. Determinarea unui cuplaj de cardinal maxim și a unei mulțimi stabile de cardinal maxim într-un 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

$$E_1 = \{sv_1 : v_1 \in V_1\}, E_2 = \{v_2t : v_2 \in V_2\},$$

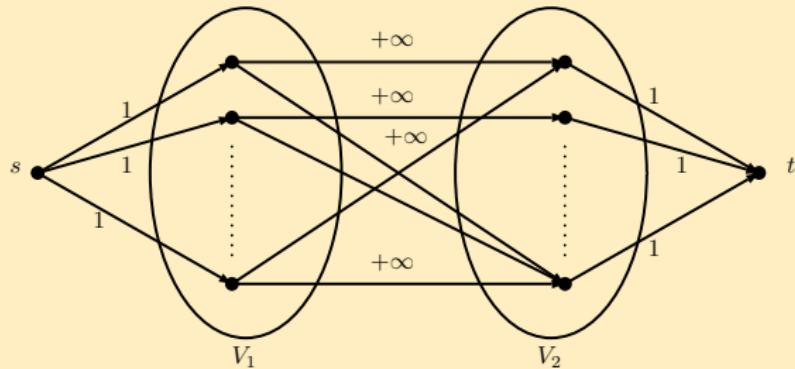
$$E_3 = \{v_1v_2 : v_1 \in V_1, v_2 \in V_2\},$$

- $c : E(G_1) \rightarrow \mathbb{N}$ 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

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



Dacă $x = (x_{ij})$ este un flux întreg în R , atunci mulțimea $\{ij : i \in V_1, j \in V_2 \text{ și } x_{ij} = 1\}$ corespunde unui cuplaj M^x în graful bipartit G , cu $|M^x| = v(x)$.

Reciproc, orice cuplaj $M \in \mathcal{M}_G$ corespunde unei multimi 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 satisface $v(x^M) = |M|$.

Aplicații combinatoriale - Cuplaje în grafuri bipartite

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

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).

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

Aplicații combinatoriale - Recunoașterea secvențelor digrafice

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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. 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 $R = (G_1, s, t, c)$.

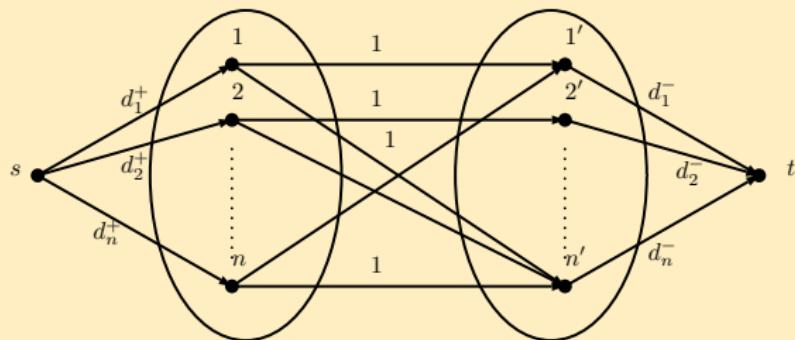
Graph Algorithms * C. Croitoru - Graph Algorithms *

Aplicații combinatoriale - Recunoașterea secvențelor digrafice

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

G_1 este obținut din graful complet bipartit $K_{n,n}$ cu bipartiția $(\{1, 2, \dots, n\}, \{1', 2', \dots, n'\})$, prin stergerea multimii 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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 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_{ij}^- arce, ij' , pe care $x_{ij'} = 1$.

Dograful 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 $\mathcal{O}(nm + n^2 \log n) = \mathcal{O}(n^3)$.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 combinatoriale - Conexiunea pe muchii

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

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 mulțimi de muchii astfel încât nu există drum de la s la t în graful obținut prin stergerea ei din G .

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

Teoremă

$$p_e(s, t) = c_e(s, t).$$

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

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algoritma Grafurilor - Cursul 9 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algoritma Grafurilor - Cursul 9 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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.

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

Proof (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 to 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

$$\min_{s, t \in V(G), s \neq t} c_e(s, t) (*)$$

Aplicații combinatoriale - Conexiunea pe muchii

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

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

$$\forall v \in S \text{ și } \forall w \in T \ c_e(v, w) \leq c(S, T) \ (**)$$

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 $\mathcal{O}(n \cdot (nm + n^2c)) = \mathcal{O}(n^2m)$.

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

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

$$k(G) = \begin{cases} n - 1, & \text{dacă } G = K_n \\ \min_{s, t \in V(G), s \neq t} c(s, t), & \text{dacă } G \neq K_n \end{cases} (****)$$

Aplicații combinatoriale - Conexiunea pe noduri

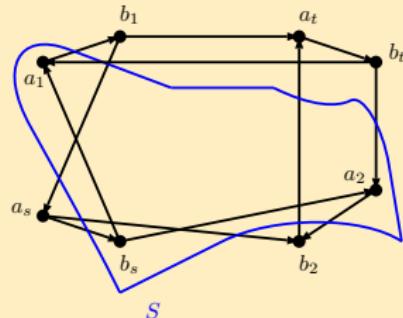
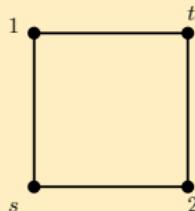
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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



Aplicații combinatoriale - Conexiunea pe noduri

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

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 multime 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

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 (S, T) be 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 multimi 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 .

Aplicații combinatoriale - Conexiunea pe noduri

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

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 $\forall v' \in V'$ și $\forall v'' \in V''$ avem $p(v', v'') = k(G)$.

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

Aplicații combinatoriale - Conexiunea pe noduri

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

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 rezolvăm toate problemele de flux maxim $p(v, w)$, cu $vw \notin E$. Numărul acestor probleme este $\mathcal{O}(nl) = \mathcal{O}\left(n\left(\frac{2m}{n} + 1\right)\right) = \mathcal{O}(m)$.

Astfel complexitatea timp a determinării lui $k(G)$ este $\mathcal{O}(m(nm + n^2 \log n))$.

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

Exerciții pentru seminarul din săptămâna a 12-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 -

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 a 12-a

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

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.

Exerciții pentru seminarul din săptămâna a 12-a

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

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

Exercițiu 5. O companie IT are n angajați P_1, P_2, \dots, P_n care trebuie să lucreze la m proiecte L_1, L_2, \dots, L_m . Pentru orice angajat P_i avem o listă \mathcal{L}_i a proiectelor la care poate lucra și s_i numărul de proiecte din \mathcal{L}_i pe care le poate termina într-o săptămână ($s_i \leq |\mathcal{L}_i|$). Fiecare proiect necesită cel puțin un angajat.

Cum se poate determina numărul minim de săptămâni necesare terminării tuturor proiectelor folosind fluxuri în rețele.

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

Exercițiu 6. Planul de evacuare de urgență al unei clădiri este descris ca un grid $n \times n$; frontierele celulelor sunt rutile de evacuare către exteriorul clădirii (grid-ului). O instanță a problemei evacuării conține dimensiunea n a grid-ului și m puncte de plecare (colțurile celulelor).

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

Exerciții pentru seminarul din săptămâna a 12-a

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

Exercițiu 6 (continuare). Instanța primește un răspuns afirmativ dacă există m drumuri disjuncte către frontieră grid-ului plecând din cele m puncte de mai sus. Dacă drumurile acestea nu există, atunci instanța primește un răspun negativ.

Determinați o reprezentare a acestei **problem a evacuării** ca o problemă de flux într-o rețea. Descrieți un algoritm eficient pentru a recunoaște instanțele pozitive ale problemei (care este complexitatea timp a algoritmului?).

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

Exercițiu 7. Fie $G = (V, E)$ un graf cu n noduri $\{v_1, v_2, \dots, v_n\}$ și $c : E \rightarrow \mathbb{R}_+$ o funcție de capacitate pe muchiile lui G . O **tăietură** în G este o bipartiție (S, T) a lui V . Capacitatea unei tăieturi (S, T) este $c(S, T) = \sum_{e \in E, |e \cap S|=1} c(e)$.

Exerciții pentru seminarul din săptămâna a 12-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 -

Exercițiul 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 induș de ordin $n \geq 3$) cu toate capacitatele 1, arătați că există $\frac{n(n-1)}{2}$ tăieturi minime.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciții pentru seminarul din săptămâna a 12-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 -

Exercițiu 8. Fie $G = (V; E)$ un digraf și $w : V \rightarrow \mathbb{R}$ astfel ca $w(V) \cap \mathbb{R}_+$ 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. 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.

Graph Algorithms * C. Croitoru - Graph Algorithms *

Algoritmica Grafurilor - Cursul 10

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

1 **Fluxuri în rețele**
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

• **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 -

2 **Reduceri în timp polinomial pentru probleme pe 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 -

• **Multimii stabile de cardinal maxim**
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

• **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 -

• **Probleme Hamiltoniene**
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

3 **Exerciții pentru seminarul din săptămâna a 13-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. Croitoru -

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

Fluxuri de cost minim

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

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

$$a(x^0) = \min \{a(x) : x \text{ flux în } R, v(x) = v\}.$$

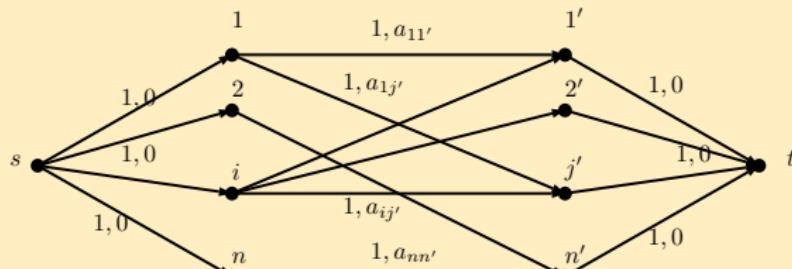
Să observăm că, dacă v nu este mai mare decât valoarea fluxului maxim în R , atunci problema are întotdeauna soluții ($a(x)$ este o funcție liniară definită pe mulțimea nevidă și compactă din \mathbb{R}^{m+1} a tuturor fluxurilor de valoare v).

Fluxuri de cost minim - Exemple

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algoritmică Grafurilor - Cursul 10 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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 $a_{j't} = 0$, $\forall i, j \in \{1, 2, \dots, n\}$.



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

Fluxuri de cost minim - Exemple

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 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 clientii 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 , $\forall i \in \{1, \dots, n\}, \forall j \in \{1, \dots, m\}$.

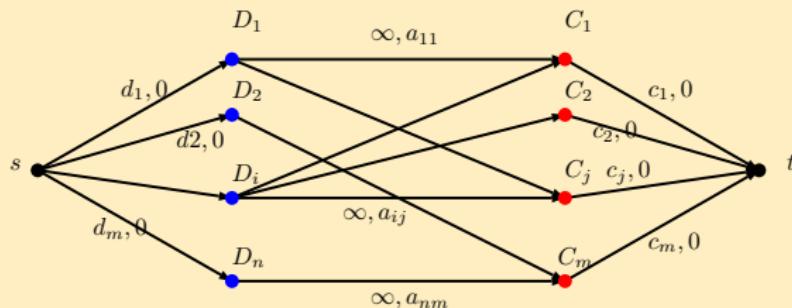
Să se determine o schemă de transport care să satisfacă cererea clientilor cu un cost total al transportului minim.

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

Fluxuri de cost minim - 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

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.



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

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 posibile 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 $a(x^1) = a(x) + r(P) \cdot a(P)$.

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 $c(x^1) < c(x)$.

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

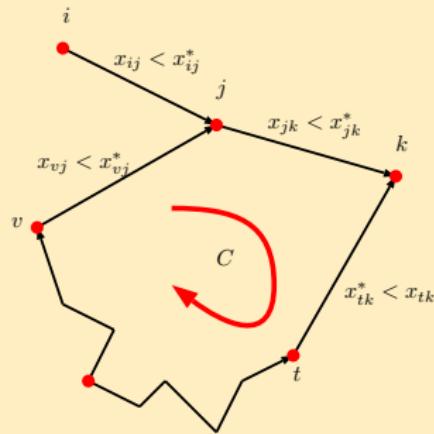
$$\Delta(x, x^*) = \min \{ \Delta(x, x') : x' \text{ flux de cost minim de valoare } v \},$$

unde $\Delta(x, x') = |\{ij \in E : x_{ij} \neq x'_{ij}\}|$.

Fluxuri de cost minim

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 încis, 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

$$\delta(C') = \min \left\{ \min_{kj \text{ forward in } C'} (x_{kj} - x_{kj}^*), \min_{kj \text{ backward in } C'} (x_{jk}^* - x_{jk}) \right\},$$

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

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

Fluxuri de cost minim

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 $ij \in E$ (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 .

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 Algorithms * C. Croitoru - Graph Algorithms *

Fluxuri de cost minim - Un algoritm generic (Klein, Busacker, Gowan etc.)

* 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 $\mathcal{O}(nm^2 \log n)$ iterării.

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).

Exemplu

SAT \leqslant_P **3SAT**

SAT

Instanță: $U = \{u_1, u_2, \dots, u_n\}$ o mulțime finită de variabile booleene;

$C = C_1 \wedge C_2 \dots \wedge C_m$ o formulă CNF peste U :

$$C_i = v_{i1} \vee v_{i2} \vee \dots \vee v_{ik_i}, i = \overline{1, m}, \text{ unde}$$

$$\forall i_j, \exists \alpha \in \{1, 2, \dots, n\} \text{ a. i. } v_{ij} = u_\alpha \text{ sau } v_{ij} = \bar{u}_\alpha.$$

Întrebare: Există o asignare $t : U \rightarrow \{\text{true}, \text{false}\}$ a. i. $t(C) = \text{true}$?

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

3SAT este restricția problemei **SAT** la mulțimea instanțelor în care fiecare clauză C_i are exact 3 literali ($k_i = 3$), unde un literal, $v_{i,j}$, este o variabilă sau o variabilă negată.

Problema SAT este faimoasă deoarece este prima problemă despre care s-a arăta ca este NP-completă (Cook, 1971).

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

$$\text{NP} \neq \text{NP} \cap \text{coNP} = \text{P}$$

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 Algorithms * C. Croitoru - Graph Algorithms *

Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

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

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 $C_i = v_{i1} \vee v_{i2} \vee v_{i3}$, $i = \overline{1, m}$, (unde $\forall i_j, \exists \alpha \in \{1, 2, \dots, n\}$ a. i. $v_{ij} = u_\alpha$ sau $v_{ij} = \bar{u}_\alpha$) 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 $|S| \geq k$.

Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

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

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 .

$$V(G) = \left(\bigcup_{i=1}^n V(T_i) \right) \cup \left(\bigcup_{j=1}^m V(Z_j) \right)$$

$$E(G) = \left(\bigcup_{i=1}^n E(T_i) \right) \cup \left(\bigcup_{j=1}^m E(Z_j) \cup E_j \right).$$

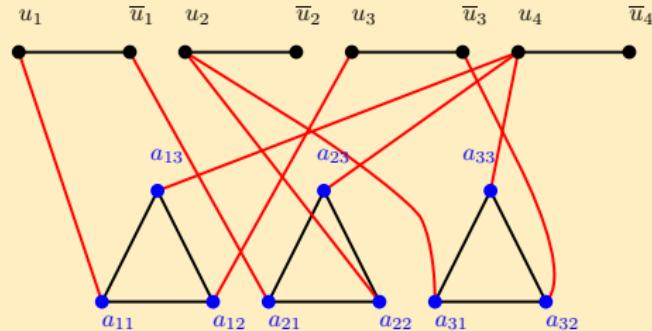
Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

Evident, construcția lui G se poate face în timp polinomial relativ la dimensiunea $n + m$ a instanței 3SAT. Fie $k = n + m$.

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

Exemplu

$$U = \{u_1, u_2, u_3, u_4\}; C = (u_1 \vee u_3 \vee u_4) \wedge (\overline{u}_1 \vee u_2 \vee u_4) \wedge (u_2 \vee \overline{u}_3 \vee u_4); k = 4 + 3 = 7.$$



Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

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

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 $C_j = v_{j1} \vee v_{j2} \vee v_{j3}$ și $S \cap V(Z_j) = \{a_{jk}\}$ ($k \in \{1, 2, 3\}$), atunci (deoarece $a_{jk}, v_{jk} \in E$) urmează că $v_{jk} \notin S$.

Reduceri în timp polinomial - Problema mulțimii stabile de cardinal maxim

* 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 \overline{S}_1 mulțimea stabilă $\overline{S}_1 = \bigcup_{i=1}^n V'_i$ 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 $t(C_j) = \text{true}$, $\forall j = 1, m$, urmează că există $k_j \in \{1, 2, 3\}$ astfel încât $t(v_{jk}) = \text{true}$. Fie $\overline{S}_2 = \bigcup_{j=1}^m \{a_{jk}\}$. Evident, \overline{S}_2 este o mulțime stabilă în G având m noduri.

Fie $\overline{S} = \overline{S}_1 \cup \overline{S}_2$. Evident, $|\overline{S}| = n + m = k$ (astfel $|\overline{S}| \geq k$). Dacă arătăm că \overline{S} este o mulțime stabilă, atunci **răspunsul la SM pentru instantă (G, k) este da**.

Să presupunem că $\exists v, w \in \overline{S}$ astfel încât $e = vw \in E(G)$. Atunci o extremitate a lui e este în \overline{S}_1 , iar cealaltă în \overline{S}_2 . Dacă $v \in \overline{S}_1$, 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 $v = u_\alpha \notin \overline{S}_1$, contradicție.

Reduceri în timp polinomial - Problema multimii stabile de cardinal maxim

- $v = \bar{u}_\alpha$, $w = a_{jk_j}$, $\alpha \in \{1, 2, \dots, n\}$, $j \in \{1, 2, \dots, m\}$, $k_j \in \{1, 2, 3\}$
şı $v_{jk_j} = \bar{u}_\alpha$.

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.

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

COL

Instanță: $G = (V, E)$ graf și $p \in \mathbb{N}^*$.

Întrebare: Există o p -colorare (a nodurilor) lui G ?

Teorema 4

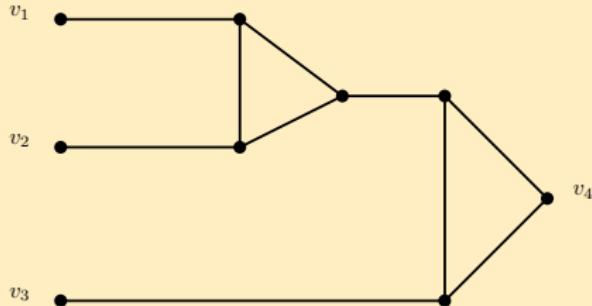
3SAT \leqslant_P COL.

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.

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

Lema 1

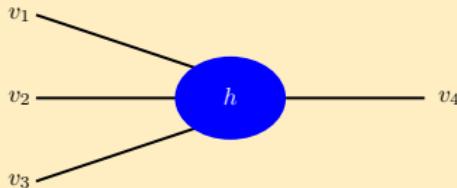
Fie H graful



- 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 $c(v_4) \neq a$.

Demonstrație. Se examinează lista 3-colorărilor lui H .

Vom utiliza reprezentarea simplificată a grafului H :



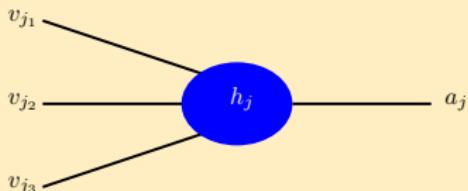
Demonstrația Teoremei 4. Considerăm datele unei instanțe a 3SAT:

$U = \{u_1, \dots, u_n\}$, ($n \in \mathbb{N}^*$), 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_{j_k} ($k = \overline{1, 3}$) sunt nodurile corespunzătoare literalilor v_{j_k} , grafurile h_j sunt disjuncte, și a_j sunt noduri distințte.

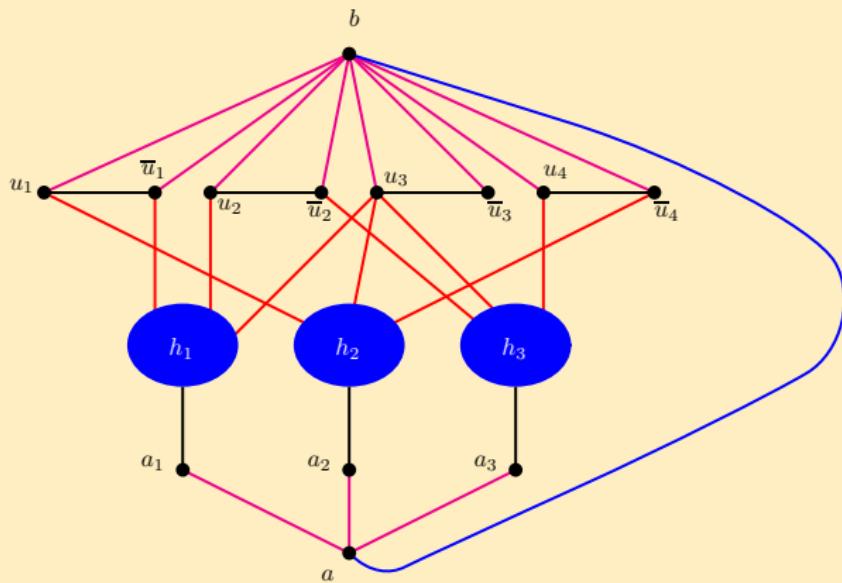
- Considerăm un nod nou a , și toate muchiile aa_j , $\forall j = \overline{1, m}$.
- Considerăm un nod nou b , toate muchiile bu_i , $b\bar{u}_i$, $\forall i = \overline{1, n}$ și ba .

Graful G are un număr de liniar de noduri relativ la $n + m$.

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



Reduceri în timp polinomial - Colorări ale nodurilor

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
Traveling salesman problem C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Să presupunem că răspunsul la 3SAT pentru instanță considerată este da

Astfel $\exists t : U \rightarrow \{\text{true}, \text{false}\}$ a. i. $t(C) = \text{true}$, 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 Lemă 1 b), putem extinde c la o 3-colorare, în fiecare graf h_j , astfel încât $c(a_j) \neq 2$, adică $c(a_j) \in \{1, 3\}$.

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 instanța dată este da.

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

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

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 are 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

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

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

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 $P_2 \leqslant_P P_1$.

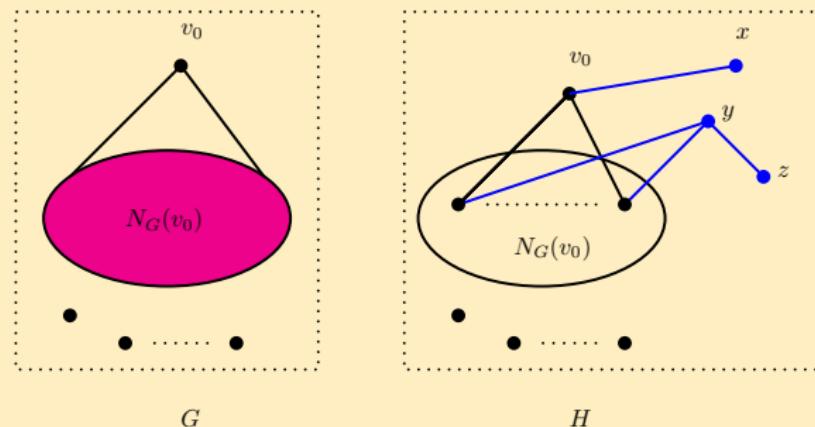
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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 $E(H) = E(G) \cup \{xv_0, yz\} \cup \{wy : w \in V(G)$ și $wv_0 \in V(G)\}$.

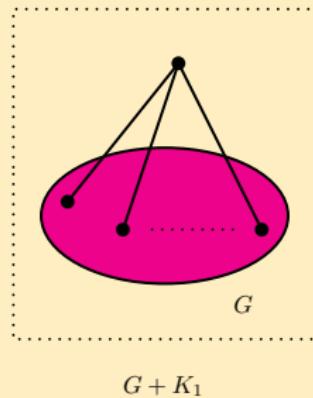
Demonstrația Teoremei 5 (continuare).



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.

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.

Reduceri în timp polinomial - Probleme Hamiltoniene

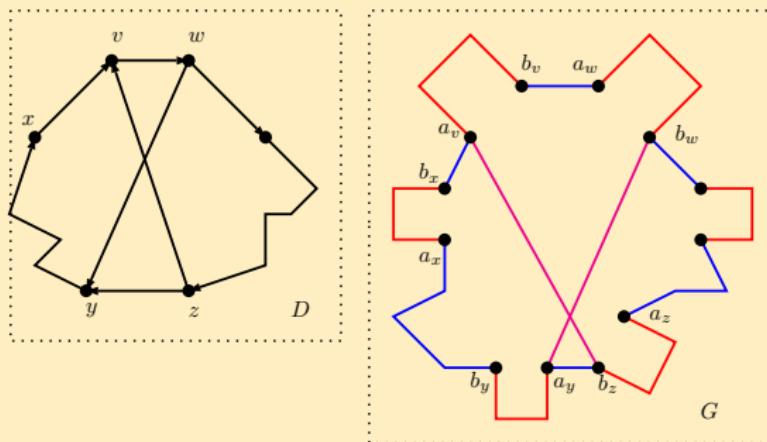
Evident fiecare circuit în G dă un circuit în D și reciproc, fiecare circuit în D dă un circuit în G .

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 :

$$P_3(v) = (\{a_v, b_v, c_v, d_v\}, \{a_v c_v, c_v d_v, d_v b_v\}).$$

Fiecare 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

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

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 ($\mathbf{DCH} \leqslant_P \mathbf{CH}$) este de fapt $\mathbf{DCH} \leqslant_P \mathbf{BCH}$.

Deoarece $\mathbf{BCH} \leqslant_P \mathbf{CH}$ este evidentă, Teorema 5 este complet demon-
strată. □

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 a 13-a

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. 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. î. $Ax \leq b$?

(Inegalitatea \leq dintre doi vectori este pe componente.) Hint: You can try $\text{SM} \leq_P \text{INT}$.

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

Exercițiu 2. Considerăm următoarea problemă de decizie P

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 $\text{SM} \leq_P \text{P}$.

Exerciții pentru seminarul din săptămâna a 13-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

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 $E \subseteq \mathbb{P}_k(V) = \{A \subseteq E : |A| = k\}$. Este ușor de văzut că un hipergraf 2-uniform este un graf simplu.

Spunem că un hipergraf k -uniform $H = (V, E)$ este *simplu* 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?

- (a) Arătați că problema 3-SIMPLE este NP-completă.
- (b) Arătați că problema 2-SIMPLE este în P.

Exerciții pentru seminarul din săptămâna a 13-a

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

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** ∈ **P**.

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

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 $v \in V \setminus S$ sibil din S pe un drum de lungime cel mult 2.

- Arătați că un quasi-kernel poate fi construit în $\mathcal{O}(n + m)$, unde $n = |V|$ și $m = |E|$.
- 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.

Exerciții pentru seminarul din săptămâna a 13-a

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. Considerăm următoarea problemă de decizie: **LPL**

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

Întrebare: Are G un drum P astfel încât $\text{length}(P) \geq k$?

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:

NUCLEU

Instanță: G un digraf.

Întrebare: Are G un kernel?

Arătați că următoarea construcție conduce la o reducere polinomială a lui **SAT** la **NUCLEU** (i.e. $\text{SAT} \leq_P \text{NUCLEU}$):

Exerciții pentru seminarul din săptămâna a 13-a

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

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

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 a 13-a

Exercise 8. Considerăm următoarea problemă de decizie 2SAT

Instanță: \mathcal{C} o familie de clauze fiecare cu doi literali.

Întrebare: Există o asignare a valorilor de adevăr către variabile astfel ca toate clauzele din \mathcal{C} să fie satisfăcute?

Define G digraful (implicațiilor): $V(G)$ = mulțimea literalilor folosiți în \mathcal{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 \mathcal{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 $\mathcal{O}(n + m)$.

Exercise 9. Arătați că următoarea problemă este NP-completă.

MAX-2SAT

Instanță: \mathcal{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 variabile astfel ca cel puțin k dintre clauze să fie satisfăcute?

Exerciții pentru seminarul din săptămâna a 13-a

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

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

Exercise 10. Considerăm următoarea problemă de decizie NAE-3SAT

Instanță: \mathcal{C} o familie de clauze fiecare cu doi 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:

$$C_j^1 = v_{j_1} \vee v_{j_2} \vee y_j, \quad C_j^2 = v_{j_3} \vee x \vee \overline{y}_j.$$

Exercitii pentru seminarul din săptămâna a 13-a

Exercițiu 11. Arătați că următoarea problemă este NP-completă.

P

Instanță: G un graf, $k \in \mathbb{N}$, $k \leq |G|$.

Întrebare: Are G un arbore parțial, T , cu $\Delta(T) \leq k$?

Algoritmica Grafurilor - Cursul 11

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

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

Teorema 1

(Karp, 1972) $\text{SM} \leqslant_P \text{CH}$.

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.

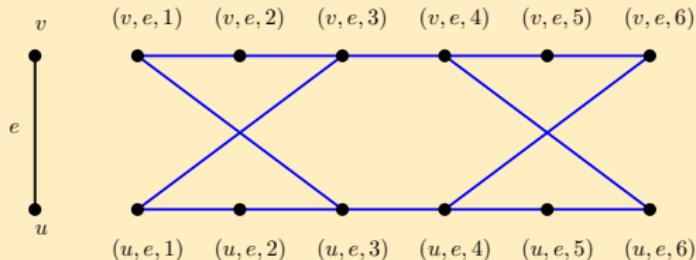
Fie $k = |V| - j$. Să presupunem că $k > 0$ (pentru a evita cazurile banale)

- (i) Fie $A = \{a_1, a_2, \dots, a_k\}$ o mulțime de k noduri distințe.
- (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).



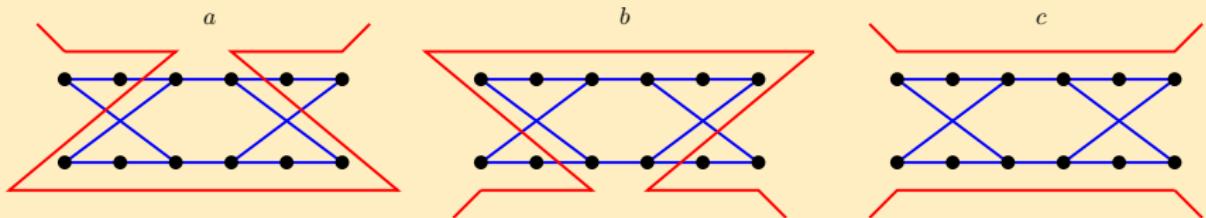
Graful are proprietatea că, dacă este un subgraf inducător al unui graf Hamiltonian, H , și nici unul dintre nodurile (w, e, i) cu $w \in \{u, v\}$ și $i = \overline{2, 5}$ nu are alți vecini în H , atunci singurele posibilități de parcursere a lui G'_e pe un circuit Hamiltonian sunt:

Graph Algorithms * C. Croitoru - Graph Algorithms *

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

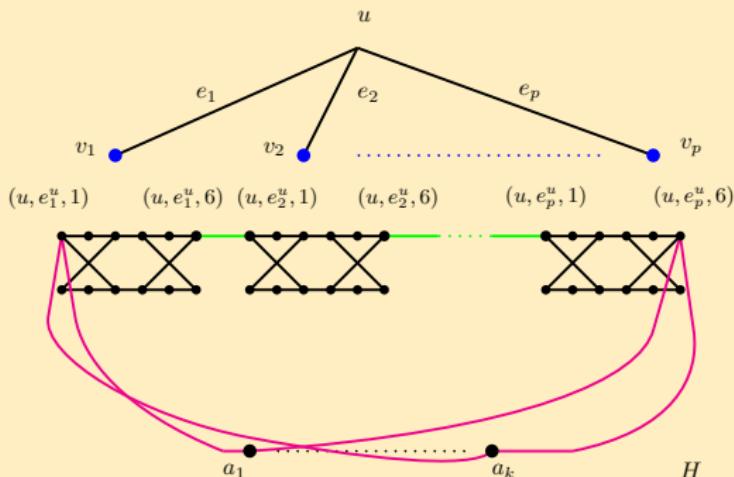


Demonstrație (continuare). Astfel, dacă circuitul Hamiltonian “intră” în G'_e printr-un nod corespunzând lui u ($(u, e, 1)$ sau $(u, e, 6)$) atunci “părăsește” graful G'_e de asemenei printr-un nod corespunzând lui u .

- (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 $E_u'''' = \{a_i(u, e_1^u, 1), a_i(u, e_p^u, 6) : i = \overline{1, k}\}$

*C. Crețanu - Graph Algorithms * C. Crețanu - Graph Algorithms * C. Crețanu - Graph Algorithms*

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).



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

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} .

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

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

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

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

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 , 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.

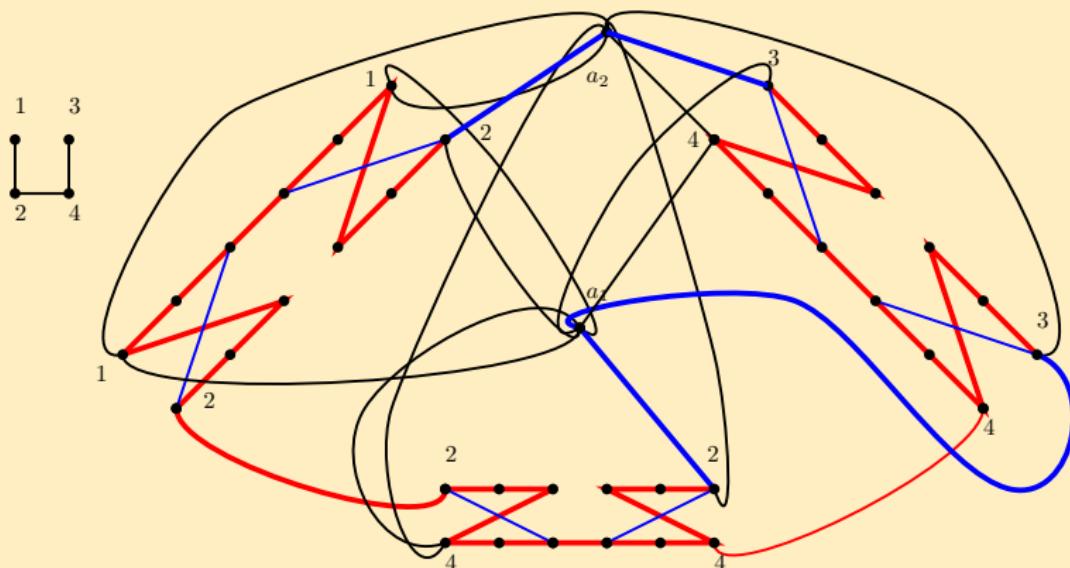
Algoritmi * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

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

Demonstrație (continuare).



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

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 G . \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

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_0 , a. i. suma ponderilor pe muchiile lui H_0 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_0 reprezintă cel mai scurt traseu care vizitează toate orașele exact odată (cu excepția lui v_0) pe care îl poate parcurge un comis-voiajor plecând din v_0 și întorcându-se în v_0 .

Aceasta este probabil, cea mai studiată problemă de optimizare NP-hard!

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

Reduceri în timp polinomial - Problema comis-voiajorului (TSP)

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

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_0 , în K_n , cu $d(H_0)$ minim printre toate circuitele Hamiltoniene ale lui K_n , unde

$$d(H_0) = \sum_{e \in E(H_0)} d(e).$$

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 .

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

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_0 , în K_n , a. î. $d(H_0) \leq B$?

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

Teorema 2

CH \leq_P DTSP.

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

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.

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

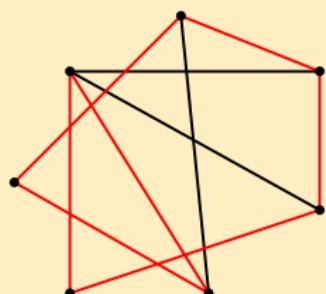
Reduceri în timp polinomial - Problema comis-voiajorului (TSP)

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

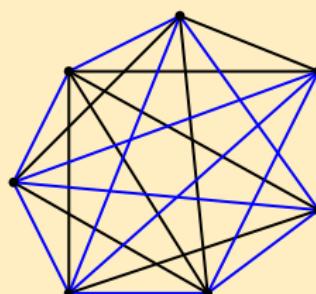
Fie

$$d(vw) = \begin{cases} 1, & \text{dacă } vw \in E(G) \\ 2, & \text{dacă } vw \in E(\overline{G}) \end{cases} \text{ și } B = n.$$

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:



G



weight 2
weight 1

Urmează că TSP este o problemă NP-hard. \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

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:

Teorema 3

Dacă există un algoritm de aproximare în timp polinomial A pentru TSP a. î. $R_A < \infty$, atunci problema CH poate fi rezolvată în timp polinomial.

Demonstrație. Fie A un algoritm de aproximare în timp polinomial cu $R_A < \infty$. Există $k \in \mathbb{N}$ astfel încât $R_A \leq k$.

Fie $G = (V, E)$ un graf arbitrar, instanță a CH. Dacă $n = |V|$, atunci considerăm $d : E(K_n) \rightarrow \mathbb{N}$ definită prin

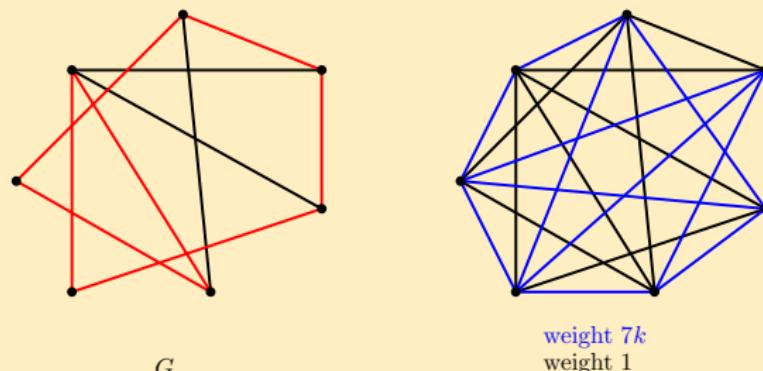
$$d(uv) = \begin{cases} 1, & \text{dacă } uv \in E(G) \\ kn, & \text{dacă } uv \notin E(G) \end{cases} .$$

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 $H_{\mathcal{A}} = H_0$.
- Dacă $d(H_{\mathcal{A}}) > kn$, atunci $d(H_0) > n$. Într-adevăr, presupunând că $d(H_0) = n$, avem $\frac{d(H_{\mathcal{A}})}{d(H_0)} \leq k$, astfel $d(H_{\mathcal{A}}) \leq kd(H_0) = kn$, 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 Algorithms * 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 de aproximare în timp polinomial \mathcal{A} cu $R_{\mathcal{A}} = 3/2$.

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).

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, $(v_{i_1}, v_{i_2}, \dots, v_{i_1})$. Eliminăm toate aparițiile multiple ale nodurilor interne pentru a obține un circuit Hamiltonian H_A în K_n cu mulțimea de muchii $H_A = \{v_{j_1}v_{j_2}, v_{j_2}v_{j_3}, \dots, v_{j_n}v_{j_1}\}$ (amândouă construcțiile se fac în $\mathcal{O}(n^2)$).

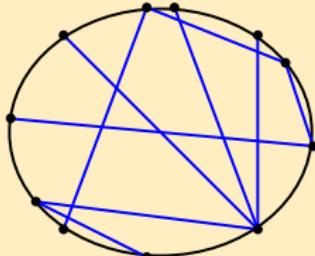
Fie H_A soluția aproximativă a TSP dată de algoritmul lui Christofides.

Fie $m = \lfloor n/2 \rfloor$ și H_o o soluție optimă. Arătăm (după Cornuejols & Nemhauser) că

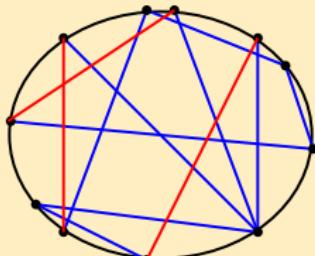
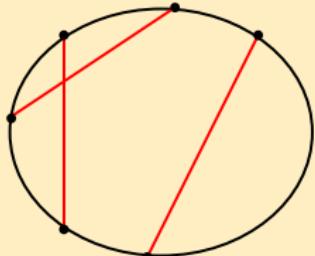
$$\forall n \geq 3, d(H_{\mathcal{A}}) \leq \frac{3m-1}{2m} d(H_o).$$

Abordări ale problemelor NP-hard - Algoritmul lui Christofides

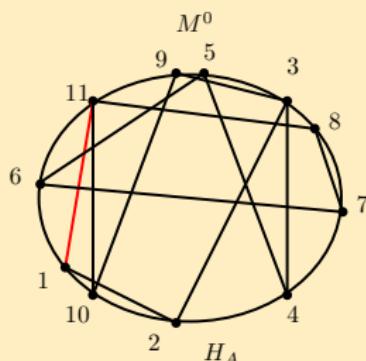
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru



T^0



Eulerian graph



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

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

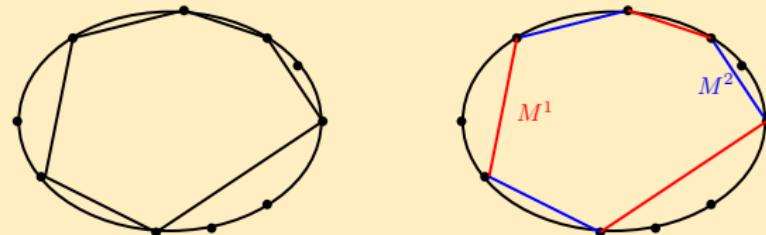
Demonstrație (continuare) Fie $H_o = \{v_1 v_2, v_2 v_3, \dots, v_n v_1\}$ (dacă e necesar, redenumim nodurile).

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 $v_{i_j} v_{i_{j+1}}$).

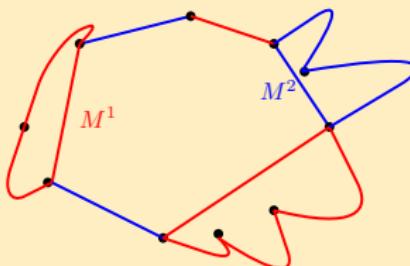
Deoarece H este un circuit par, poate fi scris ca reuniunea a două cuplaje perfecte în $[W]_{K_n}$, $M^1 \cup M^2$. Să presupunem că $d(M^1) \leq d(M^2)$.

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 : $(v_{i_j} v_{i_j+1} \in M^i \Rightarrow v_{i_j} v_{i_j+1}, \dots, v_{i_{j+1}-1} v_{i_{j+1}} \in H^i)$.



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 de cost minim obținut din H_o prin stergerea 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

$$\begin{aligned} d(H_A) &\leq d(T^0) + d(M^0) \leq d(H_o) \left(1 - \frac{\alpha}{m}\right) + \alpha d(H_o) = \\ &= \left(1 + \frac{\alpha(m-1)}{m}\right) d(H_o) \stackrel{\alpha \leq 1/2}{\leq} \frac{3m-1}{2m} d(H_o), \forall n \geq 3. \quad \square \end{aligned}$$

Abordări ale problemelor NP-hard - Algoritmul de colorare greedy

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
         $j \leftarrow j + 1;$ 
         $v \leftarrow$  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
```

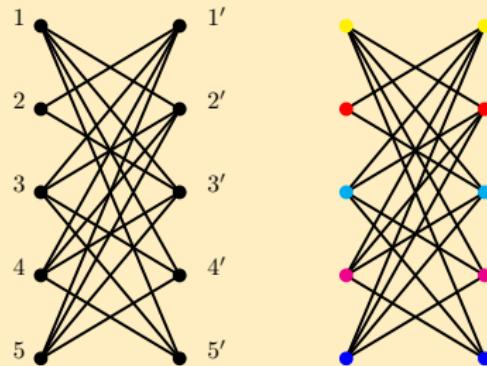
Abordări ale problemelor NP-hard - Algoritmul de colorare greedy

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

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

Să observă 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 $11', 22', \dots, nn'$.



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

Dacă $\pi = (1, 1', 2, 2', \dots, n, n')$, atunci algoritmul de colorare greedy returnează $c(1) = c(1') = 1$, $c(2) = c(2') = 2$, $c(n) = c(n') = n$. 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:



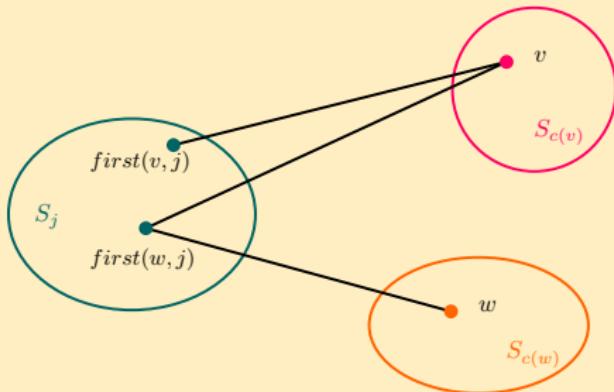
Abordări ale problemelor NP-hard - Algoritmul de colorare greedy

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, $\forall vw \in E$ și $\forall j < \min\{c(v), c(w)\}$ astfel încât $first(v, j) < first(w, j)$ în ordonarea dată de π , avem $first(w, j)v \in E$, atunci $\chi(G, \pi) = \chi(G)$.

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



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

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
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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, este 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 *

Algoritmica Grafurilor - Cursul 12

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

1

Grafuri planare

- Proprietăți elementare
- Desenarea grafurilor planare
- Separatori mici

Grafuri planare - Proprietăți elementare - Definiție

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 S o suprafață (e.g., plan, sferă) din \mathbb{R}^3 . O reprezentare a lui G pe S este un graf $G' = (V', E')$ astfel încât:

- a) $G \cong G'$;
- b) V' este o mulțime puncte distințe ale lui S ;
- c) Orice muchie $e' \in E'$ este o curbă simplă (curbă Jordan) conținută în S unindu-i extremitățile;
- d) Orice punct din S este fie un nod al lui G' fie este conținut în cel mult o muchie a lui G' .

Dacă S este un plan, atunci G este un **graf planar** și G' este o **reprezentare planară** a lui G .

Dacă S este un plan și G' este un graf satisfăcând constrângerile b), c) și d) de mai sus, atunci G' se numește **graf plan**.

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 S tangentă la π în x . Fie y punctul diametral opus lui x în S . Considerăm $\varphi : \pi \rightarrow S$ dată prin $\varphi(M) =$ punctul diferit de y în care dreapta My intersectează sfera, $\forall M \in \pi$. φ este o bijecție și astfel $\varphi(G')$ este o reprezentare a lui G pe S .

Reciproc, dacă G are o reprezentare pe o sferă S : luăm un punct y în S , considerăm x , punctul diametral opus lui y pe S , construim un plan tangent π la S în x , și definim $\psi : S \rightarrow \pi$ by $\psi(M) =$ punctul în care dreapta yM intersectează planul π , pentru orice $M \in S$. Imaginea prin ψ a reprezentării lui G pe sferă, $\psi(G)$, este reprezentarea planară dorită a lui G . □

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 -

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 exteroară.

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 exterioară a celei de-a doua.

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

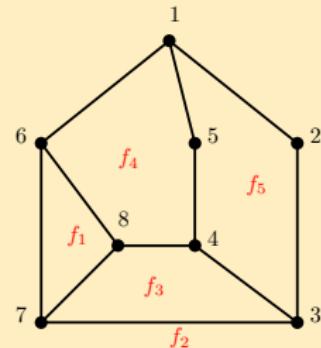
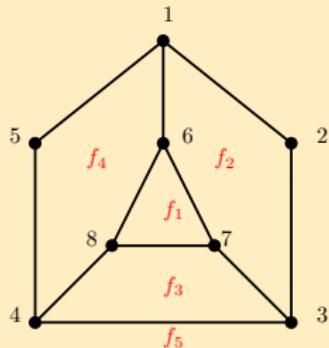
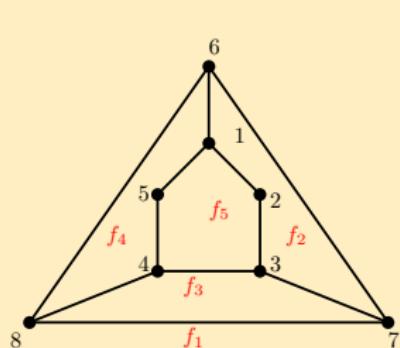
Demonstrație. Fie G' o reprezentare planară a lui G și F o față a lui G' . Fie G^0 o reprezentare a lui G' pe o sferă și F^0 față a lui G^0 corespunzând lui F . Alegem un punct y în interiorul lui F^0 , x punctul său diametral opus pe sferă, și π planul tangent în x la sferă.

$G'' = \psi(G^0)$ este o reprezentare a lui G în planul π având ca față exterioară $\psi(F^0)$. \square

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

Grafuri planare - Proprietăți elementare - Euler's formula

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



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

$$f = m - n + 2.$$

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

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,

$$m \leq 3n - 6.$$

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

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

$$\sum_{i=1}^{n-1} i \cdot n_i = 2m \leq 2(3n - 6) = 6 \left(\sum_i n_i \right) - 12 \Rightarrow \sum_i (i - 6)n_i + 12 \leq 0.$$

Pentru $i \geq 6$ toți termenii din această sumă sunt ≥ 0 , deci există $i_0 \leq 5$ astfel încât $n_{i_0} > 0$. \square

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.

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

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

Teoremă

(Fary, 1948, independent Wagner & Stein) Orice graf planar are o reprezentare planară cu toate muchiile segmente de dreaptă (reprezentare Fary).

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

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

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.

Demonstrație algoritmică. Vom schița o desenare în $\mathcal{O}(n \log n)$.

Fără a restrângе 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

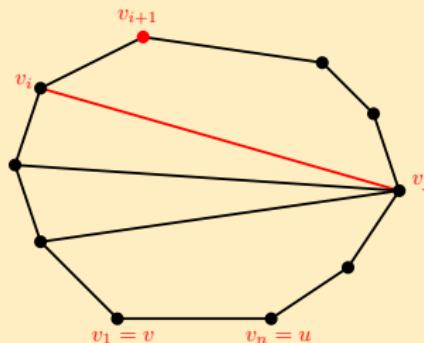
Fie G un graf planar și G' o reprezentare planară a lui G . Dacă C' este un circuit al lui G' care trece prin muchia $uv \in E(G')$, atunci există $w \in V(C')$ astfel încât $w \neq u, v$ și nu există nicio coardă interioară a lui C' cu o extremitate în w .

Demonstrație. Fie v_1, v_2, \dots, v_n nodurile lui C' întâlnite într-o parcursare de la u la v ($v = v_1$, $u = v_n$).

Demonstrație (continuare). Dacă C' nu are corzi interioare, atunci leme 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

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

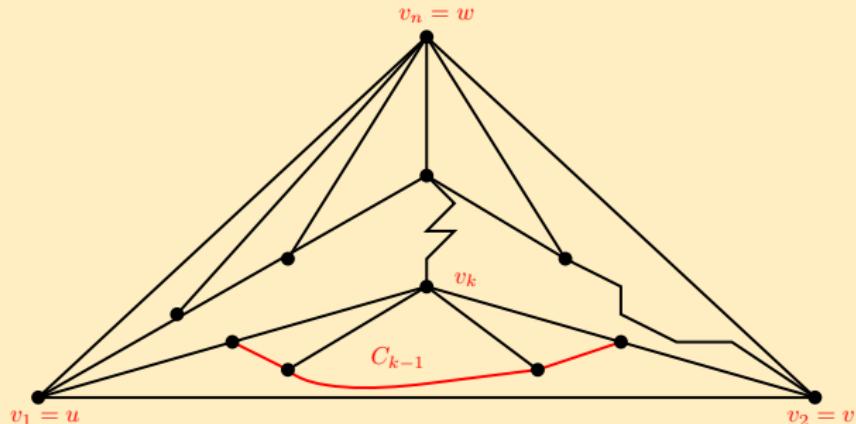
Fie G un graf maximal planar cu $n \geq 4$ noduri și G' o reprezentare planară a lui G având față exterioară triunghiul u, v, w . Atunci, există o etichetare v_1, v_2, \dots, v_n a nodurilor lui G' astfel încât $v_1 = u$, $v_2 = v$, $v_n = w$ și, pentru fiecare $k \in \{4, \dots, n\}$, avem:

- (i) Subgraful induz $G'_{k-1} = [\{v_1, \dots, v_{k-1}\}]_G$ este 2-conex și față sa exterioară este determinată de circuitul C'_{k-1} conținând uv .
- (ii) În subgraful induz G'_k nodul v_k este în față 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 $C'_{k-1} - uv$.

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

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} .

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
Algoritmică Grafurilor - Cursul 12 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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:

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

Demonstrație (continuare).

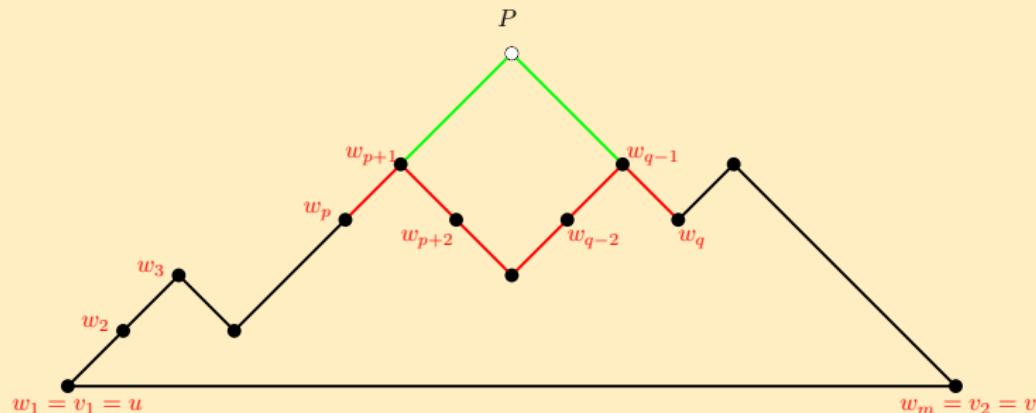
- (1) v_1 are coordonatele $(0, 0)$; v_2 are coordonatele $(i, 0)$, $i \leq 2k - 4$.
- (2) Dacă w_1, w_2, \dots, w_m sunt nodurile circuitului care corespunde feței exterioare a lui G_k , într-o parcurgere de la v_1 la v_2 ($w_1 = v_1$, $w_m = v_2$), atunci
$$x_{w_1} < x_{w_2} < \dots < x_{w_m}.$$
- (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} ($1 \leq p < q \leq m$).

Desenarea grafurilor planare

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



Demonstrație (continuare) Paralela prin w_p la prima bisectoare intersectează paralela prin w_q la cea de-a doua 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 distințe, 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 doua 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 Algorithms * C. Croitoru - Graph Algorithms *

Teorema

(Tarjan & Lipton, 1979) Fie G un graf planar cu n noduri. Există o partiție (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 partiție poate fi găsită în $\mathcal{O}(n)$.

Ideea demonstrației. Fie G un graf conex plan. Executăm o parcursere 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

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). 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.

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

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 $t_2 - t_0 \leq \sqrt{n}$.

Grafuri planare - Separatori mici

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

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

Demonstrație. Considerăm t_0 cel mai mare întreg care satisfacă $t \leq t_1$ și $|L(t)| \leq \sqrt{n}$ (există un astfel de nivel deoarece $|L(0)| = 1$). Există t_2 un cel mai mic întreg care satisfacă $t > t_1$ 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. □

O aplicație a Teoremei Separatorilor

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

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 asemenea la o 3-colorare a subgrafului induș de $B \cup C$ în G (recurivs).

Graph Algorithms * C. Croitoru - Graph Algorithms *

O aplicație a Teoremei Separatorilor

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

Timpul de execuție, $T(n)$, al acestui algoritm, satisface recursia

$$T(n) = \begin{cases} \mathcal{O}(1), & \text{dacă } n \leq c, \\ \mathcal{O}(n) + 2^{\mathcal{O}(\sqrt{n})} (\mathcal{O}(\sqrt{n}) + 2T(2n/3)), & \text{dacă } n > c. \end{cases}$$

Urmează că $T(n) = 2^{\mathcal{O}(\sqrt{n})}$ (este posibil ca acele constante din spatele
notației $\mathcal{O}(\cdot)$ să fie foarte mari).

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