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

Cuprins

1. Descrierea cursului
   - Pagina cursului şi alte informaţii
2. Aplicaţii ale teoriei grafurilor
3. Vocabularul teoriei grafurilor
   - Definiţia grafului
4. Exerciţii pentru seminarul din săptămâna viitoare

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

Pagina cursului şi alte informaţii

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 noţiunile de bază în Teoria algoritmică a grafurilor. Cunoştinţele acumulate vor fi aplicate în proiectarea unor algoritmi eficienţi pentru rezolvarea problemelor de optimizare combinatorială.

Metode de predare:
Prezentarea video a notelor de curs, care vor fi postate ca fişiere .pdf înaintea fiecărui curs. Aceste fişiere vor conţine şi exerciţiile pentru seminarii.

Pagina cursului şi alte informaţii

Seminarii: E. F. Olariu, A. C. Frăsinaru, A. Policiuc, V. Motroi
Fiecare seminar este dedicat unui număr de exerciţii (postate în avans în notele de curs) cu scopul de a aprofunda conceptele introduse la curs. Studenţii sunt încurajaţi să propună soluţii originale.

Notarea:

- Activitatea de la seminar: prezenţă (max. 13p), teste (max. 12p) şi participare (max. 10p) - max. 35 puncte.
- Teme pentru acasă: trei seturi de exerciţii, max. 15p fiecare - max. 45 puncte.
- Examen final scris: max. 70 puncte

Pentru a susţine teza scrisă în sesiune sunt necesare cel puţin 35 de puncte din cele maximum 80p din activitatea de seminar şi temele pentru acasă.
Din maximum 150p limita de promovare este de 70 puncte.

Pagina cursului şi alte informaţii

Cuprinsul cursului:

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

Aplicaţii ale teoriei grafurilor

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

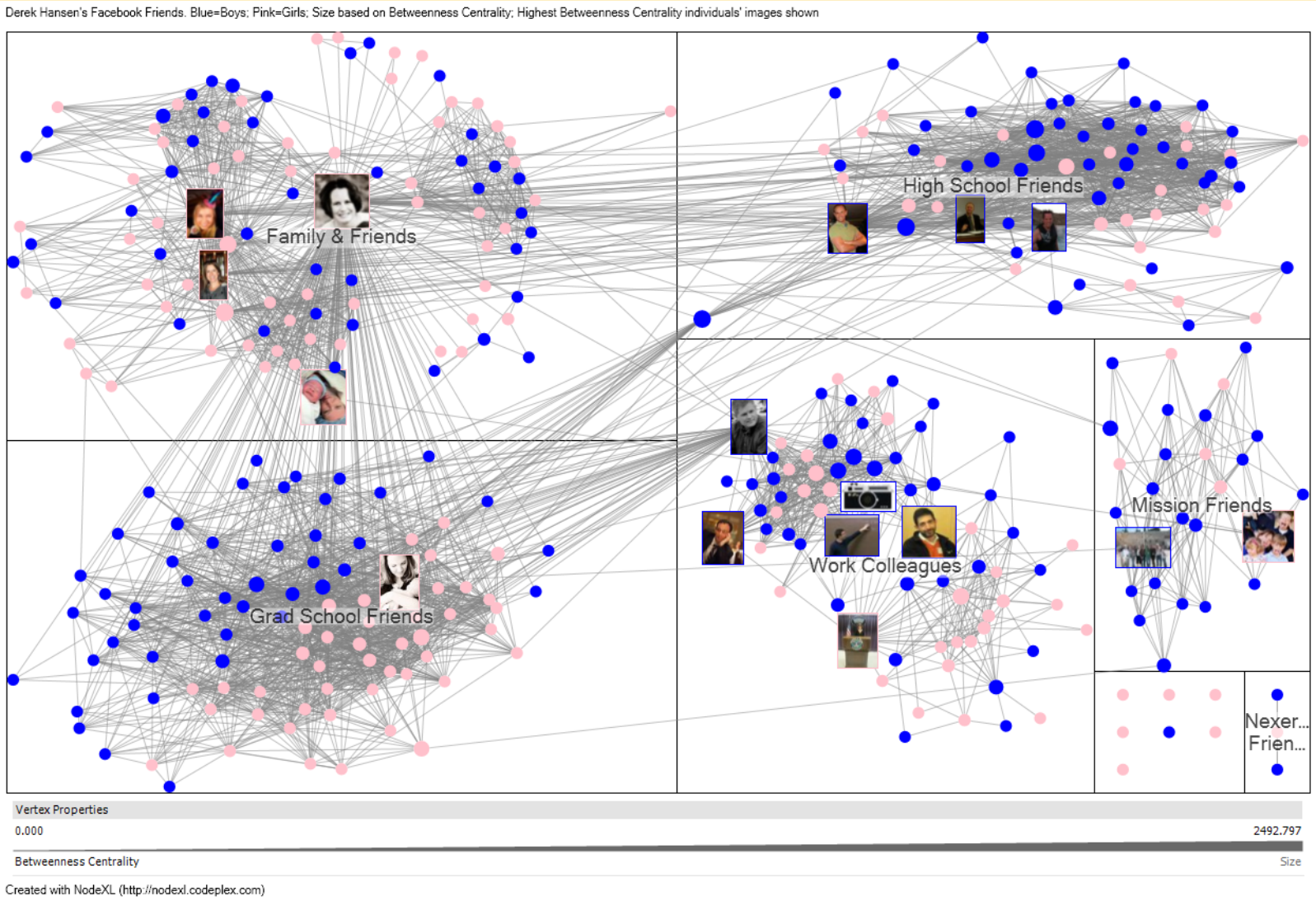


Figure: Facebook/Twitter

Aplicaţii ale teoriei grafurilor

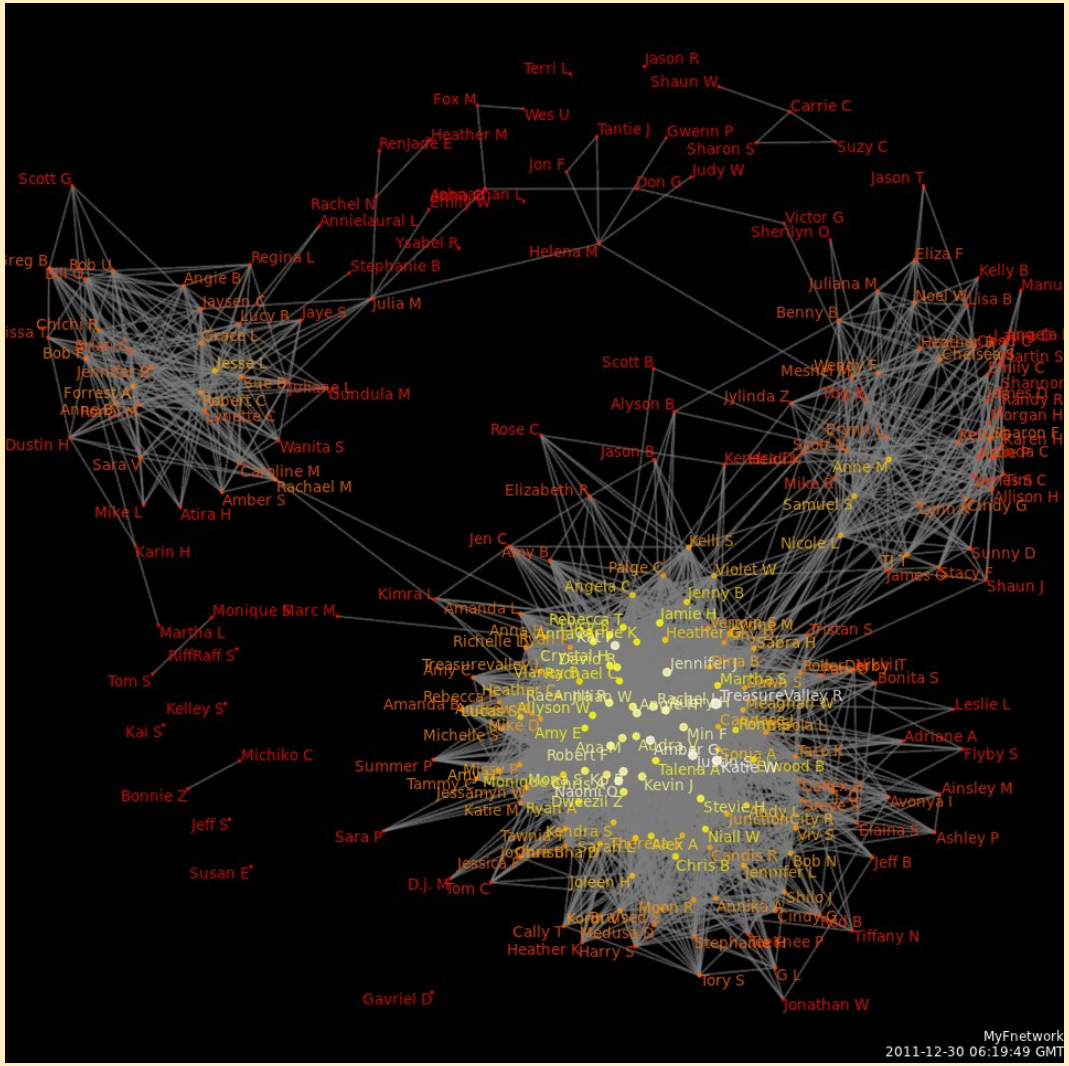


Figure: A Facebook network of a few users

Aplicaţii ale teoriei grafurilor

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

- nivelul de conexiune, densitatea;
- influenţa utilizatorilor asupra reţelei (centralitatea, potenţialul reţelei sociale (social networking potential);
- nivelul de segmentare: măsura clusterizării;
- robusteţea sau stabilitatea structurală a reţelei.

Analiza reţelelor este folosită pentru

- data mining şi agregare, marketing;
- analiza comportamentului şi predicţie în reţea;
- colectarea de informaţii şi analiza securităţii (supravegherea terorismului şi a crimei organizate) etc.

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

Aplicaţii 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

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

- **Arbori parţiali de cost minim**: pentru conectarea eficientă a unor puncte de comunicaţie (e. g. în IT&C);
- **Drumuri şi circuite Euleriene**: Problema poştasului 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 poştei sau a unor servicii, colectarea deşeurilor etc.);
- **Drumuri şi circuite Hamiltoniene**: vizitarea eficientă a unui număr de puncte (dintr-un oraş, dintr-o ţară etc); Problema comis-voiajorului, Problema rutării vehiculelor;

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

- **Colorarea grafurilor**: colorarea hărţilor (a feţelor unui graf planar), planificarea cursurilor/seminariilor (problema orarului), planificarea unei sesiuni, alocarea frecvenţelor radio mobile, alocarea regiştrilor de memorie.

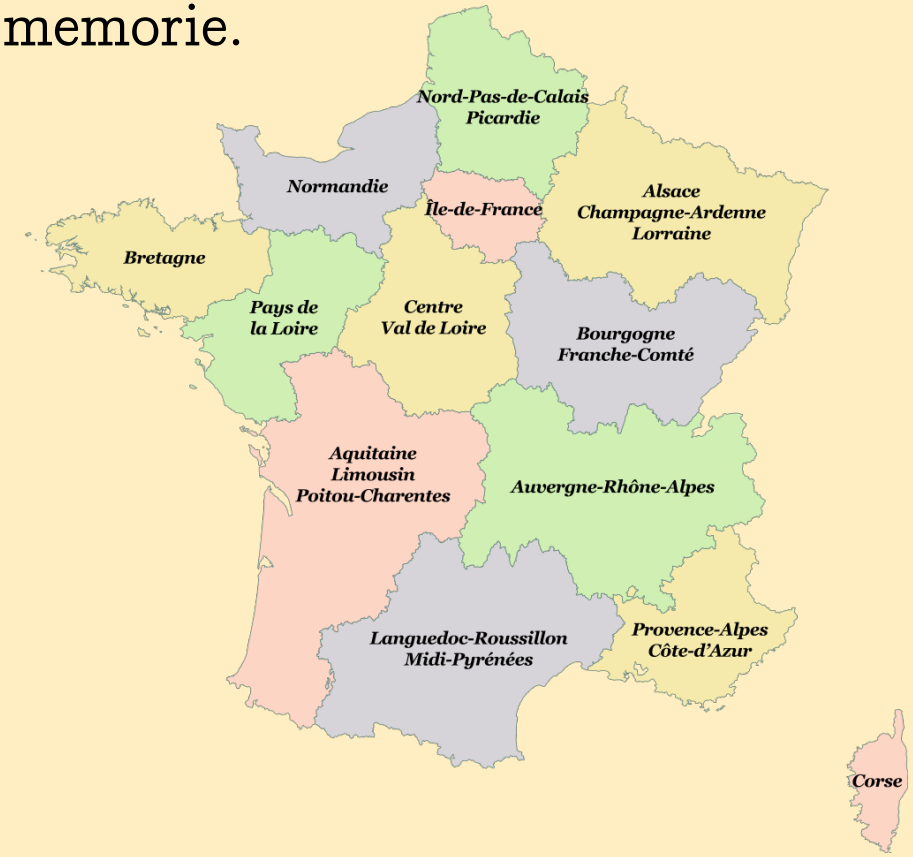


Figure: Regiunile Franţei începând cu 2016

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

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

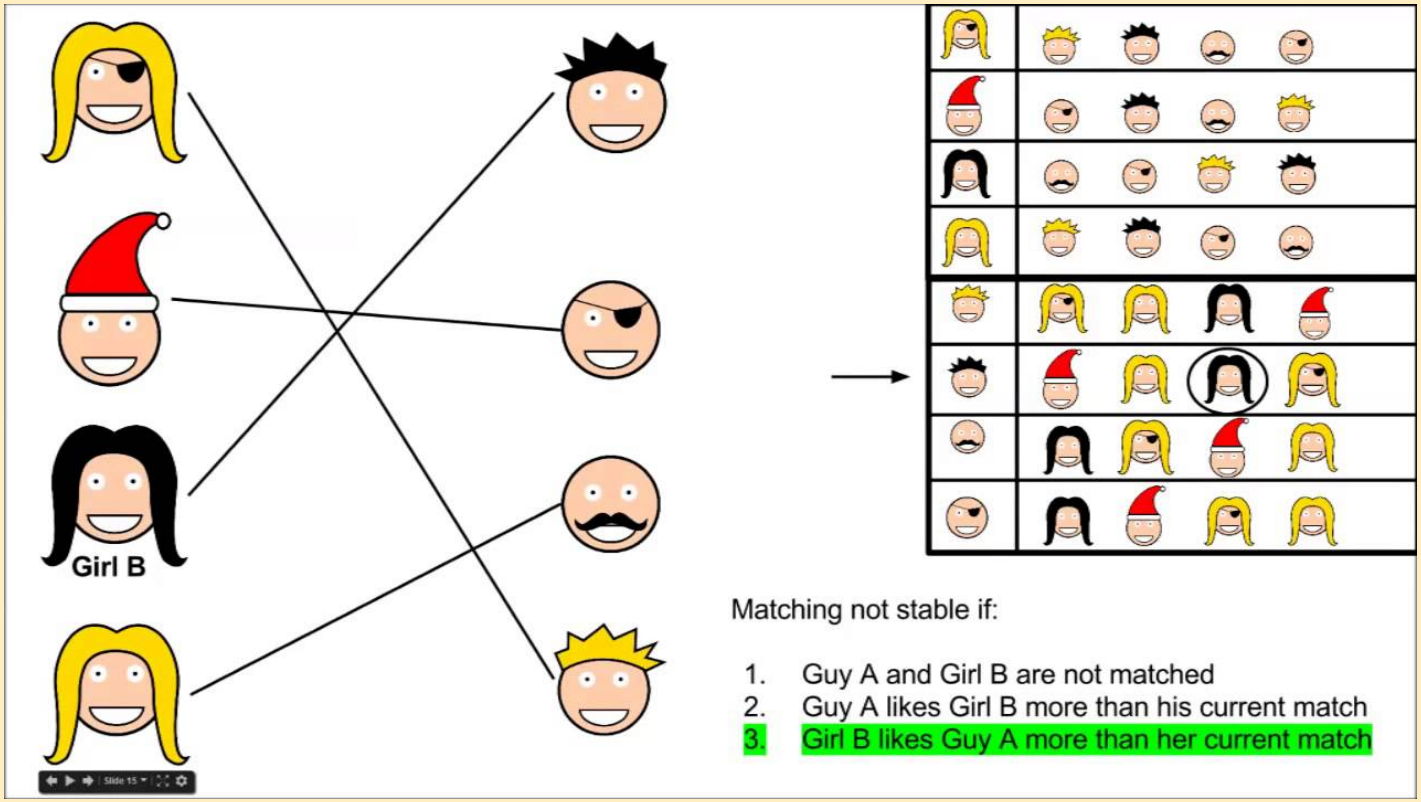


Figure: Gale Shapley algorithm

Aplicaţii 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 rezidenţilor în spitale, proceduri de admitere în învăţământul superior, identificarea repartizării optime a organelor donate pentru transplant (e. g. rinichi), problema alocării proiectelor către studenţi etc.
- **Fluxuri de valoare maximă**: determinarea nivelului de încărcare în reţelele de transport pentru îmbunătăţirea condiţiilor de trafic, reconstrucţia imaginilor din proiecţia 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Aplicaţii în informatică

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

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

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

Notaţii

Pentru o mulţime finită X:

- $|X| = card(X) \in \mathbb{N}$ este cardinalul lui X;
- Dacă $|X| = k$, atunci X este o k-mulţime;
- $2^X = \mathcal{P}(X)$ este mulţimea părţilor 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 *

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

Definiţia 1

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

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

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

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

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

Definiţia 2

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

- *e* **conectează** *(sau* **leagă***) nodurile u şi v;*
- *nodurile u şi v sunt* **adiacente** *sau u şi v sunt* **vecine***;*
- *e este* **incidentă** *cu u şi v;*
- *u şi v sunt* **capetele** *(***extremităţile***) 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 şi f sunt **adiacente** dacă au un capăt în comun: $|e \cap f| = 1$.

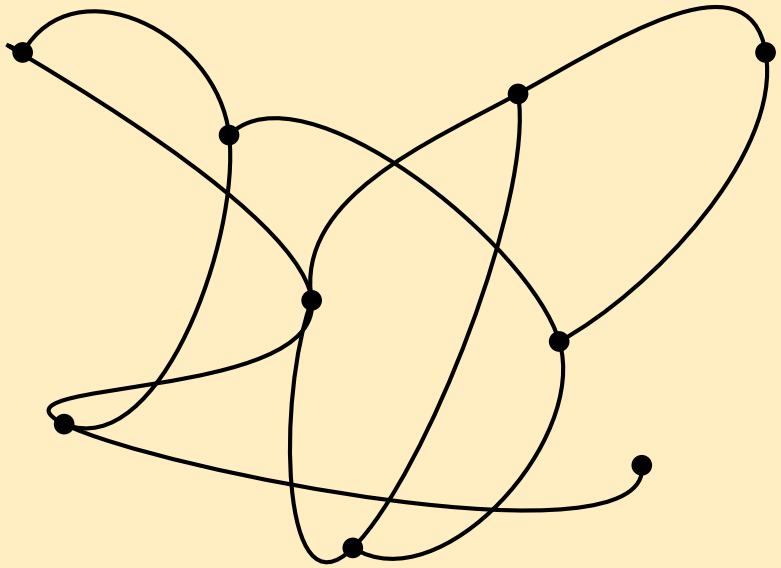
Definiţia grafului

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 mulţime de puncte (forme geometrice mici: puncte, cercuri, pătrate etc) corespunzând vârfurilor sale şi curbe care conectează vârfurile corespunzătoare muchiilor din graf.

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

Un exemplu:



Definiţia grafului

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

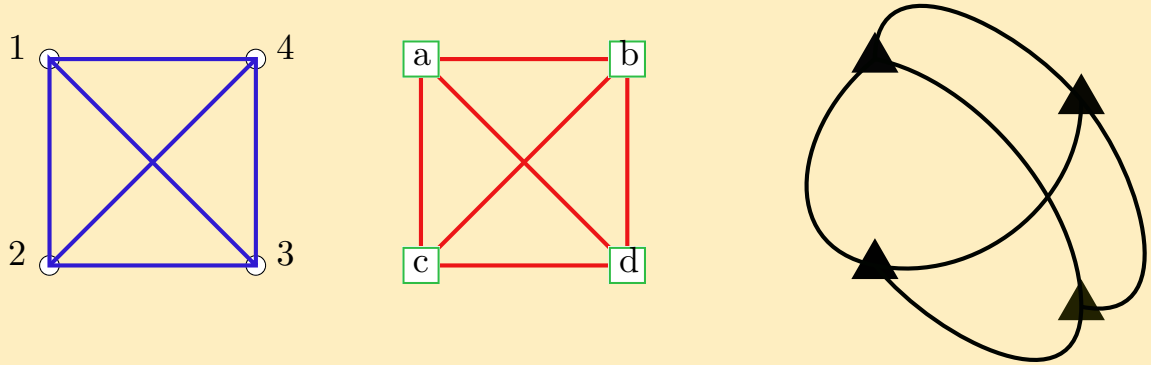
Definiţia grafului

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) şi culori nodurilor şi muchiilor obţinând reprezentări vizuale mai bune.

Mai jos sunt trei reprezentări vizuale ale aceluiaşi graf:

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



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

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

Definiţia 3

Mulţime stabilă *(sau* **mulţime independentă de noduri***) în* $G = (V, E)$*:* $S \subseteq V$ *astfel încât* $\binom{S}{2} \cap E = \varnothing$.

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

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

Notaţie

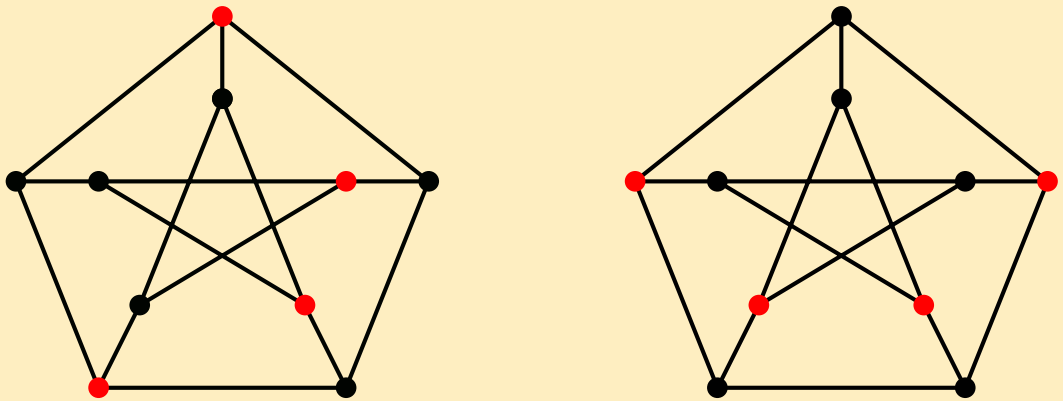
Cardinalul maxim al unei mulţimi stabile a lui G este **numărul de stabilitate** (sau **numărul de independenţă**) al lui G şi se notează cu $\alpha(G)$.

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

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

Exemplu

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



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

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

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

Definiţia 4

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

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 în comun.

Notaţie

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

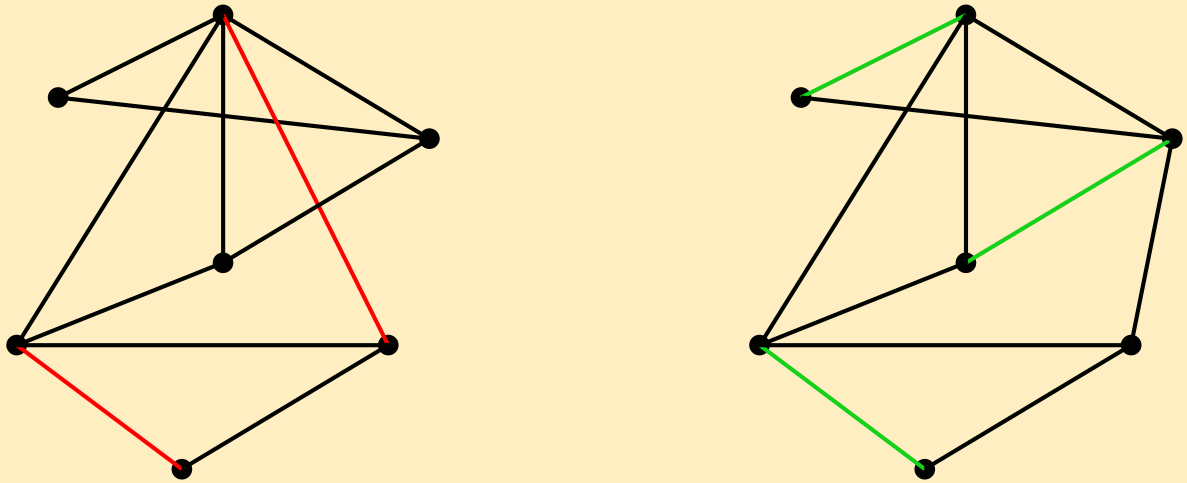
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

Pentr următorul graf sunt marcate două cuplaje cu roşu şi verde (al doilea fiind de cardinal maxim - de ce?):

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



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

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

Cuplaj maxim: **Problemă de optimizare**

P2 Input: G graf.
Output: $\nu(G)$ şi 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 $P1$ şi $P2$ sunt similare: în amândouă se cere să se determine un membru de cardinal maxim al unei familii de mulţimi relativ la un graf dat. Ce face ca ele să fie diferite?

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

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

Definiţia 5

Pentru $p \in \mathbb{N}$, o **p-colorare** *a grafului $G = (V, E)$ este o funcţie $c : V \to \{1, \ldots p\}$ 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 mulţimea tuturor nodurilor cu aceeaşi culoare este o mulţime stabilă (se mai numeşte clasă de colorare). Deoarece noduri adiacente au culori diferite, o p-colorare corespunde unei partiţii a lui V cu cel mult p mulţimi stabile (sau clase de colorare).

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

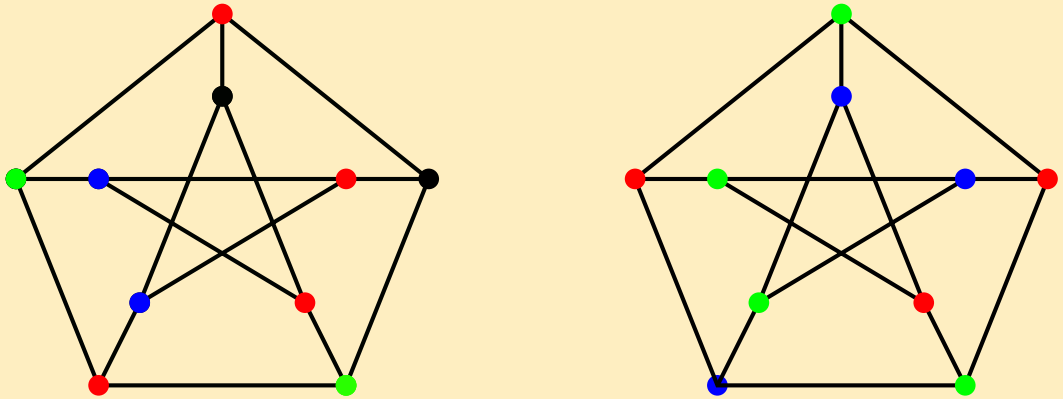
Notaţie

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 $\chi(G)$. ($\chi(G) \leqslant |G|$ - de ce?)

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

Exemplu

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



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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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: $\chi(G)$ şi o $\chi(G)$-colorare.

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

Colorarea nodurilor: **Problemă de decizie**

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

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

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

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

Definiţia 6

Pentru $p \in \mathbb{N}$, *a* **p-muchie colorare** *a grafului* $G = (V, E)$ *este o funcţie* $c : E \to \{1, \ldots p\}$ *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 mulţime de muchii cu aceeaşi culoare este un cuplaj. Astfel, o p-muchie colorare corespunde unei partiţii a lui E cu cel mult p cuplaje.

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

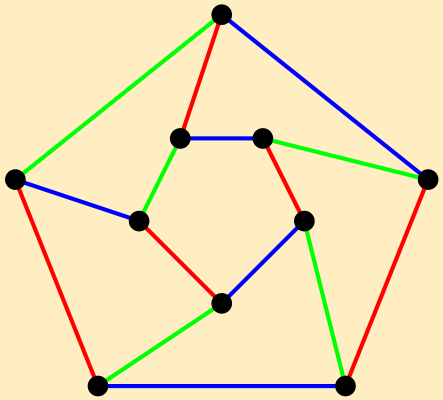
Notaţie

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 $\chi'(G)$. ($\chi'(G) \leqslant |E(G)|$ - de ce?)

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

Exemplu

Pentru următorul graf 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 *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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: $\chi'(G)$ şi o $\chi'(G)$-colorare.

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

Colorarea muchiilor: **Problemă de decizie**

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

NP-completă pentru $p \geqslant 3$ (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 *

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

Definiţia 7

Două grafuri $G_1 = (V_1, E_1)$ şi $G_2 = (V_2, E_2)$ sunt **izomorfe** *dacă există o bijecţie $\varphi : V_1 \to V_2$ astfel încât pentru orice două noduri $u_1, v_1 \in V_1$, u_1 şi v_1 sunt adiacente în G_1 (i. e., $u_1 v_1 \in E_1$) dacă şi numai dacă $\varphi(u_1)$ şi $\varphi(v_2)$ sunt adiacente in G_2 (i. e., $\varphi(u_1)\varphi(u_2) \in E_2$).*

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

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

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

Notaţie

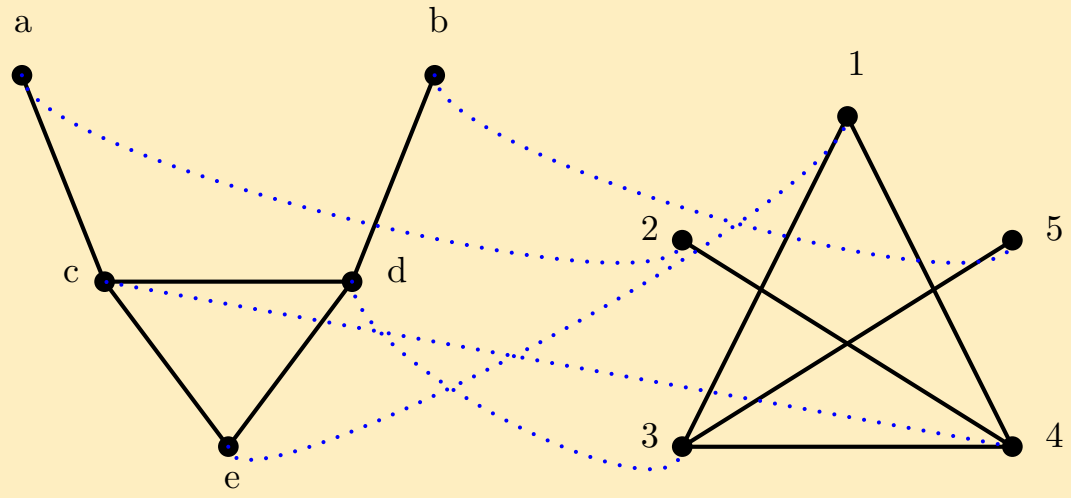
$G_1 \cong G_2$.

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

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

Exemplu

Grafurile de mai jos sunt izomorfe.



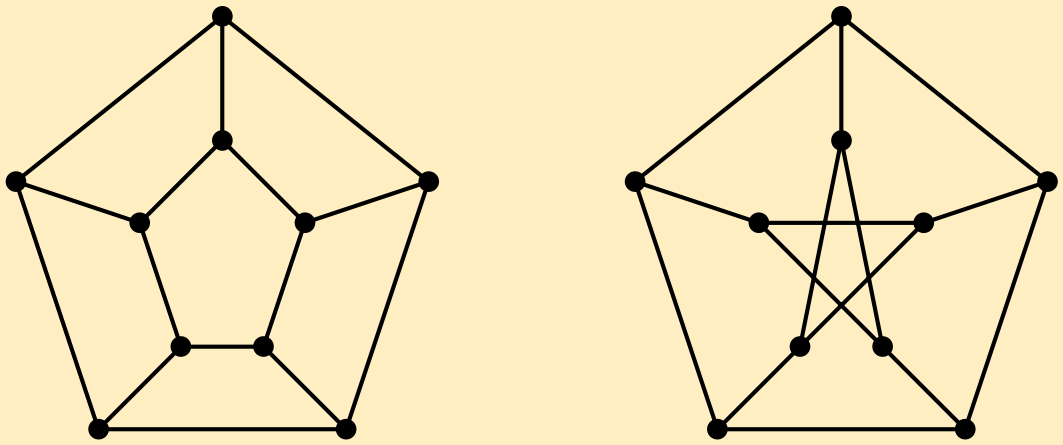
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.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

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

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

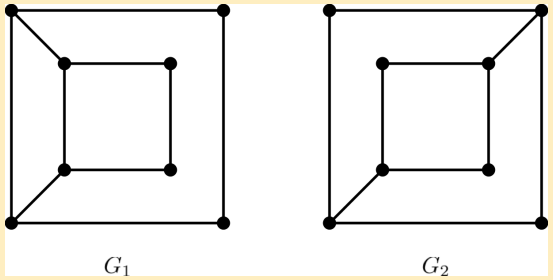
Următoarele două grafuri au aceleaşi ordine, dimensiuni şi secvenţe ale gradelor, dar nu sunt izomorfe (de ce?).



* 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 *

Exerciţii pentru seminarul din săptămâna viitoare

Exercise 1'. Grafurile de mai jos sunt izomorfe?



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

(a) Determinaţi ordinul şi dimensiunea lui G_k

(b) Arătaţi că G_k este bipartit şi determinaţi $\alpha(G_k)$.

Exerciţiul 2.

(a) Există un graf cu gradele $3, 3, 3, 3, 5, 6, 6, 6, 6, 6, 6$?

(b) Există un graf bipartit cu gradele $3, 3, 3, 3, 3, 5, 6, 6, 6, 6, 6, 6, 6, 6$?

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

Exerciţiul 3.

Doi studenţi, L(azy) şi 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, şi o soluţie lazy este să genereze (cu backtracking) toate aceste drumuri şi apoi să reţină drumul dorit. T nu este de acord şi dă următorul exemplu: fie $H = K_2 \times P_{n-1}$ ($n \geqslant 3$); adăugăm la H două noduri noi x şi y fiecare unite prin câte două muchii cu câte una dintre cele două perechi de noduri adiacente de grad 2 din H.

Graful obţinut, G, este rar dar numărul de drumuri dintre x şi y este foarte mare. Explicaţi lui L acest exemplu desenând G, arătând că este rar şi determinând numărul de drumuri dintre x şi y.

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

Exerciţii pentru seminarul 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 -

Exerciţiul 4. O sesiune de examene trebuie planificată folosind următoarele specificaţii: mulţimea examenelor este cunoscută; fiecare student trimite o listă a examenelor la care doreşte să fie examinat; fiecare examen are loc cu toţi studenţii înscrişi la examen (care este scris); fiecare student poate participa la cel mult un examen într-o aceeaşi zi.
Construiţi un graf cu ajutorul căruia să raspundeţi la următoarele întrebări (prin determinarea unor parametri corespunzători):

(a) Care este numărul maxim de examene care pot fi organizate într-o aceeaşi 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 * C. Croitoru - Graph Algorithms *

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

Exerciţiul 5. (exerciţiul 4 - continuare)

Un programator isteţ şi îndemânatic se întreabă dacă cele două probleme **NP**-hard din exerciţiul anterior nu ar putea fi rezolvate în timp polinomial deoarece graful construit pare să aparţină unei clase speciale de grafuri.

(a) Arătaţi 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 exerciţiul 4: începând cu prima zi, se planifică zilnic un număr maxim de examene (din mulţimea examenelor neplanificate încă), până când toate examenele vor fi planificate.

b) Arătaţi că această abordare greedy este greşită printr-un contraexemplu.

Exerciţii 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

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

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

(a) Care este numărul maxim de variabile de care nu este nevoie în acelaşi timp?

(b) Care este numărul minim de regiştri necesari acestor variabile?

Indicaţie: avem două tipuri de obiecte: variabilele (cu valorile lor) şi operaţiile CPU (sau operatorii) care folosesc una sau mai multe variabile.

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

Exerciţii pentru seminarul din săptămâna viitoare

Exerciţiul 7. (exerciţiul 6 - continuare) Un student se întreabă dacă la întrebările de mai sus (care corespond unor probleme **NP**-hard) nu s-ar putea da un răspuns în timp polinomial deoarece graful construit pare să aparţină unei clase speciale de grafuri.

(a) Arătaţi că pentru orice graf dat, G, există un input pentru problema de alocare a regiştrilor astfel încât graful construit (ca mai sus) să fie tocmai G.

Studentul sugerează următoarea "abordare greedy" pentru a răspunde la a doua întrebare din exerciţiul 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) Arătaţi că această abordare greedy este greşită printr-un contraexemplu.

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

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

(a) Arătaţi că un graf *autocomplementar* este conex şi că $|G| \equiv 0$ sau 1 *mod* 4.

(b) Determinaţi toate grafurile *autocomplementare* cu cel mult 7 noduri.

(c) Arătaţi că pentru orice graf G există un graf *autocomplementar* H astfel încât G este subgraf indus al lui H.

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

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

Cuprins

1. Vocabularul teoriei grafurilor
   - Variaţii în definiţia unui graf
   - Grade
   - Subgrafuri
   - Operaţii cu grafuri
   - Clase de grafuri
   - Drumuri şi circuite
2. Exerciţii pentru seminarul de săptămâna viitoare

C. Croitoru - Graph Algorithms *

Variaţii în definiţia 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 mulţime nevidă (de noduri), şi E este un multiset (de muchii) pe V, i. e., există o funcţie $m : \binom{V}{2} \to \mathbb{N}$.

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 obţinut din G prin înlocuirea fiecărei muchii multiple printr-una simplă.

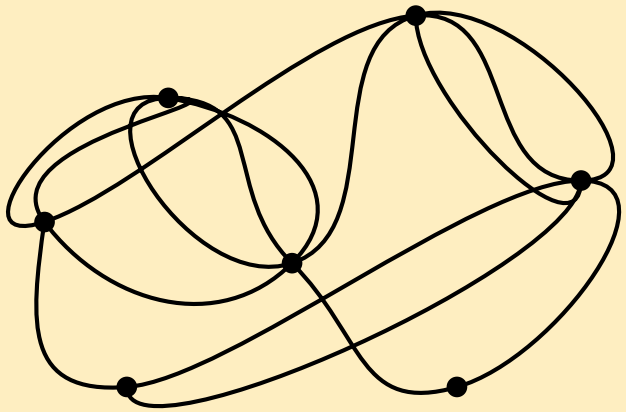
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

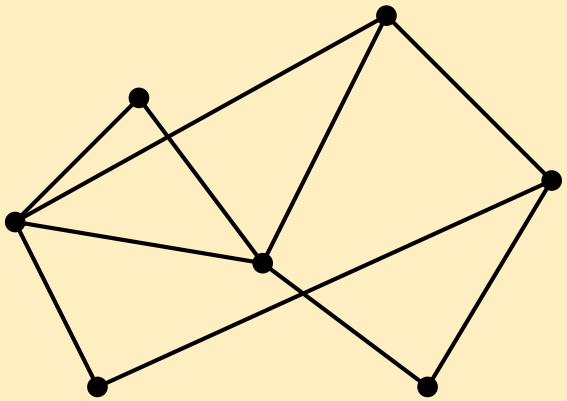
Variaţii în definiţia unui graf

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 şi 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 *

Variaţii în definiţia 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 -

Pseudograf (**graf general**): $G = (V, E)$, unde V este o mulţime (de noduri), şi E este un multiset (de muchii) peste $V \cup \binom{V}{2}$, i. e., există o funcţie $m : V \cup \binom{V}{2} \to \mathbb{N}$.

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

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

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

Graful suport al unui pseudograf G este graful obţinut din G prin înlocuirea fiecărei muchii multiple printr-una simplă şi prin ştergerea buclelor.

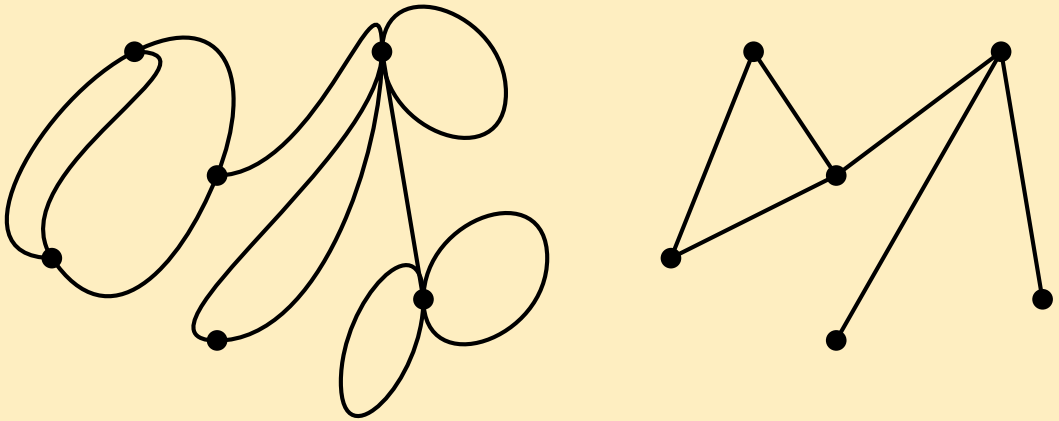
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Variaţii în definiţia 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 pseudograf şi 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 *

Variaţii în definiţia 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 mulţime (de noduri), şi $E(D) \subseteq V(D) \times V(D)$ este o mulţime 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 şi spunem că:

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

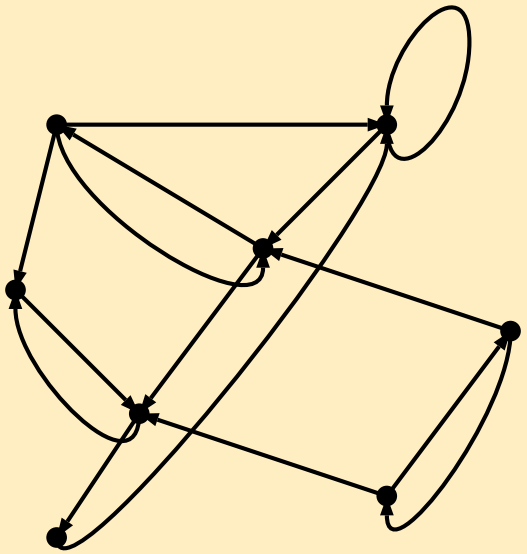
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Variaţii în definiţia 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 *

Variaţii în definiţia 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

- Pereche simetrică de arce: (uv, vu). uv este numit inversul lui vu.
- Inversul unui digraf D: se înlocuieşte fiecare arc din D cu inversul său.
- Graful suport al unui digraf D, $M(D)$, se înlocuieşte fiecare arc din D cu mulţimea 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 (distincte) sunt unite printr-o pereche simetrică de arce.
- Turneu: un graf orientat complet (orice două noduri (distincte) sunt unite exact printr-un arc).

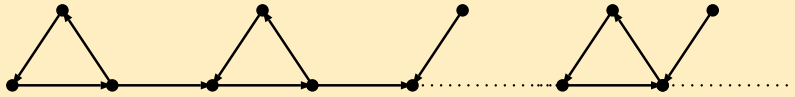
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Variaţii în definiţia unui graf

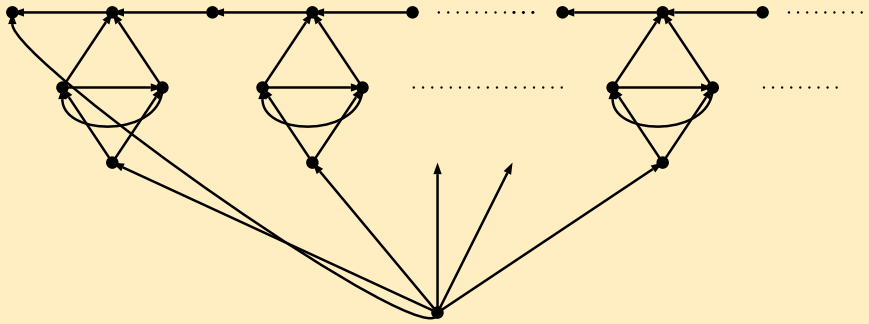
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

(Di)grafuri infinite: mulţimea nodurilor şi/sau mulţimea muchiilor (arcelor) este numărabil infinită.

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



The Danaids barrel



The sisyphus graph

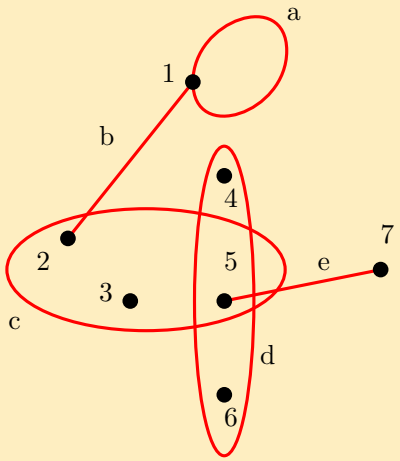
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Hipergrafuri (Sisteme de mulţimi finite)

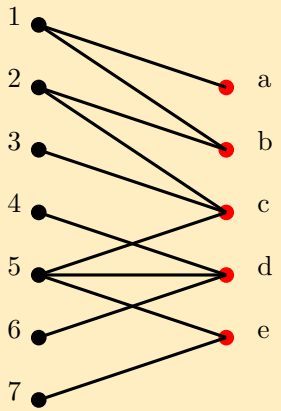
- Muchiile, numite acum hipermuchii, nu mai sunt restricţionate să fie submulţimi cu două elemente ale mulţimii de noduri. O hipermuchie este submulţime a mulţimii de noduri.
- Hipergrafuri k-uniforme: fiecare muchie are cardinalul k.

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

Fiecare hypergraf poate fi representat ca un graf bipartit:



Hypergraph H



Bipartite graph associated with H

Grade

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

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

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

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

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

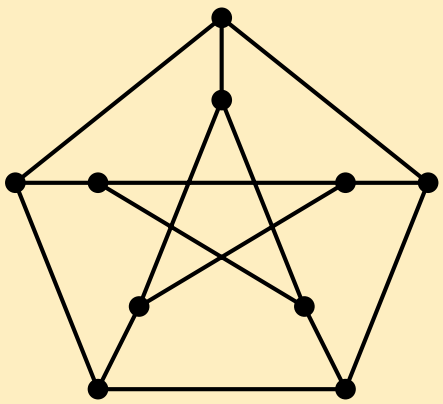
$$\Delta(G) = \max_{v \in V} d_G(v), \ \ \delta(G) = \min_{v \in V} d_G(v).$$

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

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

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

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.

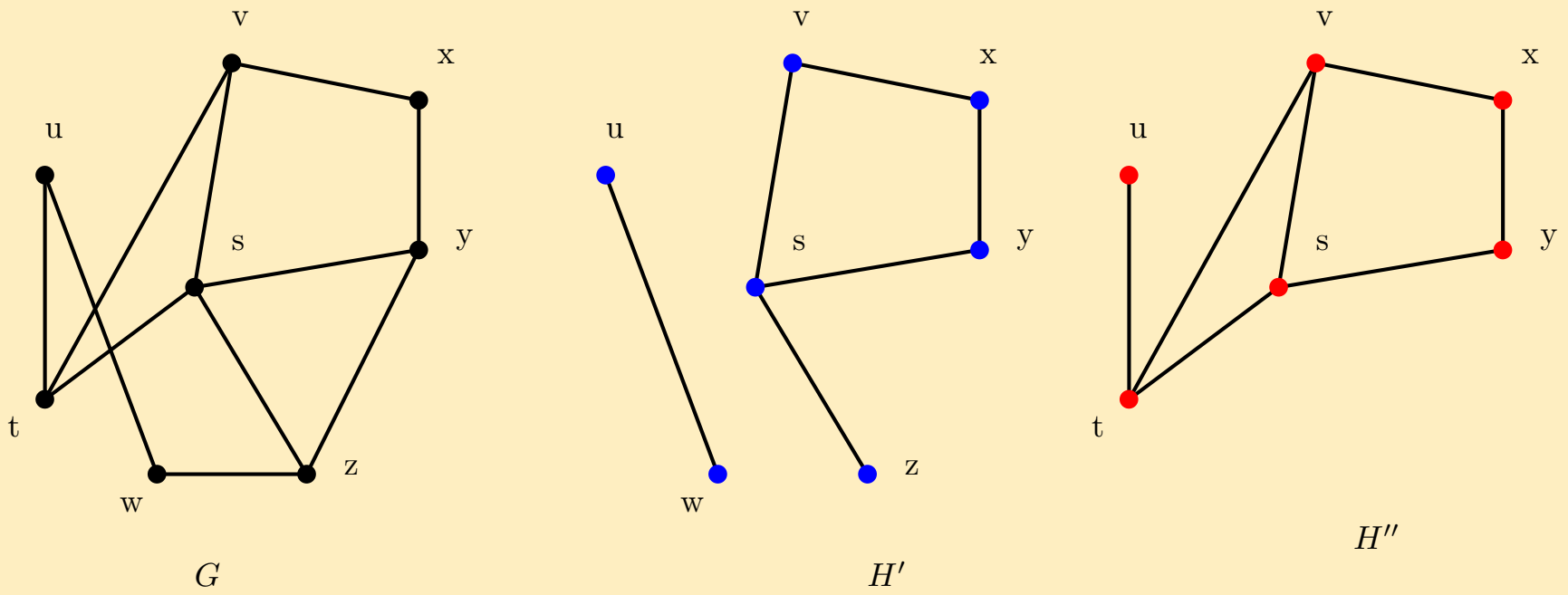
- Subgraf al lui G: un graf $H = (V(H), E(H))$ aşa încât $V(H) \subseteq V(G)$ şi $E(H) \subseteq E(G)$.
- Graf parţial al lui G: un subgraf H al lui G astfel ca $V(H) = V(G)$.
- Subgraf generat de $B \subseteq E(G)$ în G: un subgraf al lui G, $H = (V(H), E(H))$, astfel că $E(H) = B$ şi $V(H) = \cup_{uv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

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

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



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

Subgrafuri

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

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

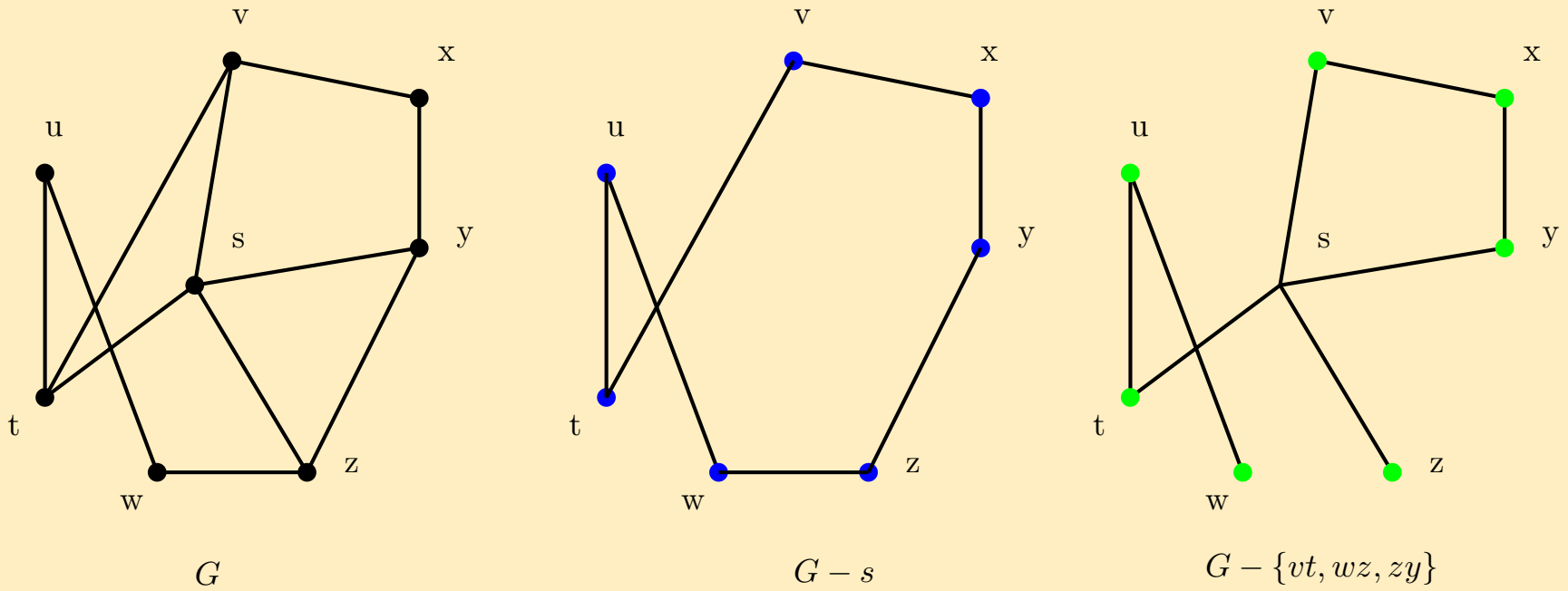
- Dacă $A \subseteq V(G)$, atunci subgraful $[V(G) \setminus A]_G$, notat prin $G - A$, este subgraful obţinut din G prin ştergerea nodurilor lui A. $G - \{u\}$ este numit subgraf de ştergere şi se notează, simplu cu $G - u$.
- Dacă $B \subseteq E(G)$, atunci subgraful $\langle E(G) \setminus B \rangle_G$, notat prin $G - B$, este subgraful obţinut din G prin ştergerea tuturor muchiilor lui B. $G - \{e\}$ se notează $G - e$.
- Definiţiii şi notaţii similare pentru digrafuri, multigrafuri 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 * 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, $G - s$, şi $G - \{vt, wz, zy\}$.



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

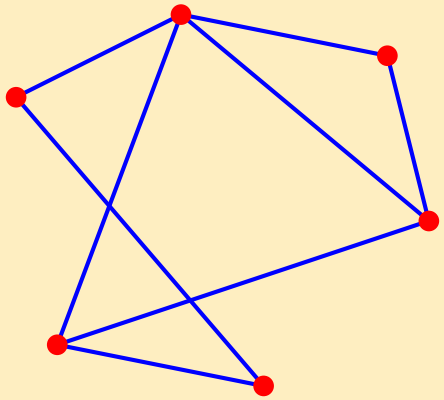
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Operaţie unară: $G = (V(G), E(G))$

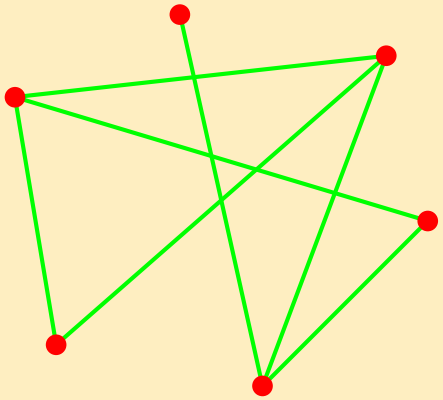
- **Complementul** unui graf G: un graf $\overline{G}$, cu $V(\overline{G}) = V(G)$ şi

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

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



G



$\overline{G}$

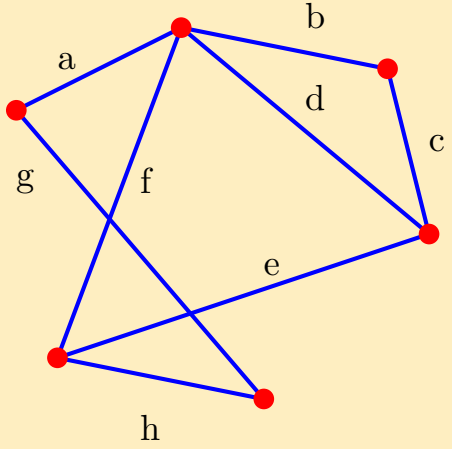
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

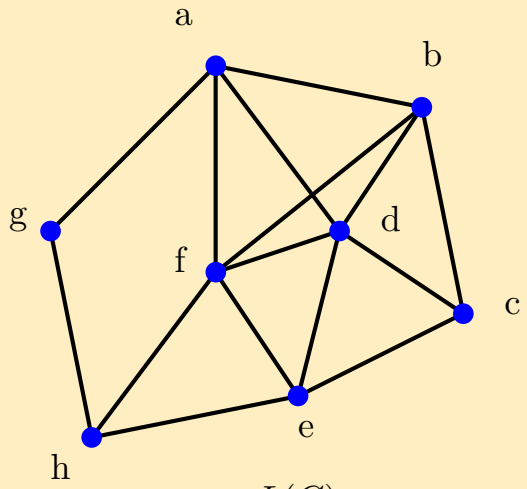
Operaţie unară: $G = (V(G), E(G))$

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

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



G



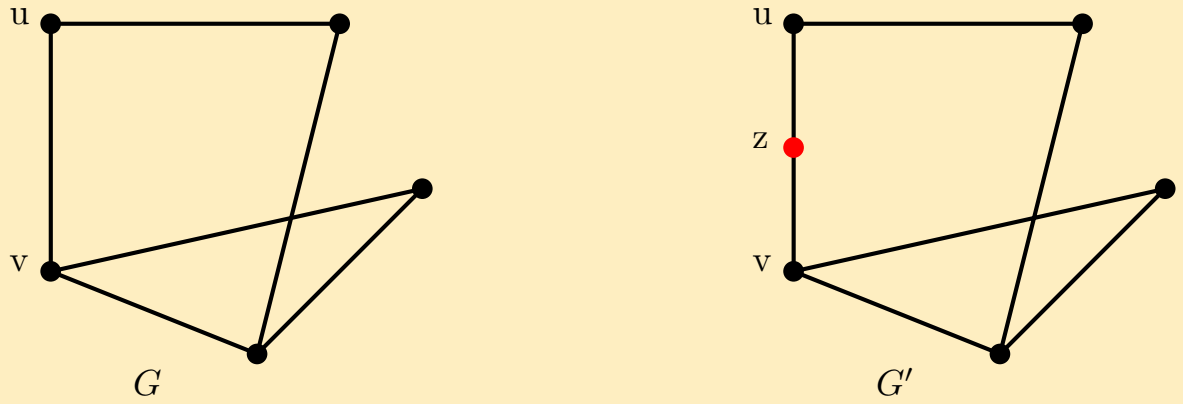
$L(G)$

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 -

Operaţie unară: $G = (V(G), E(G))$

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

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



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

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

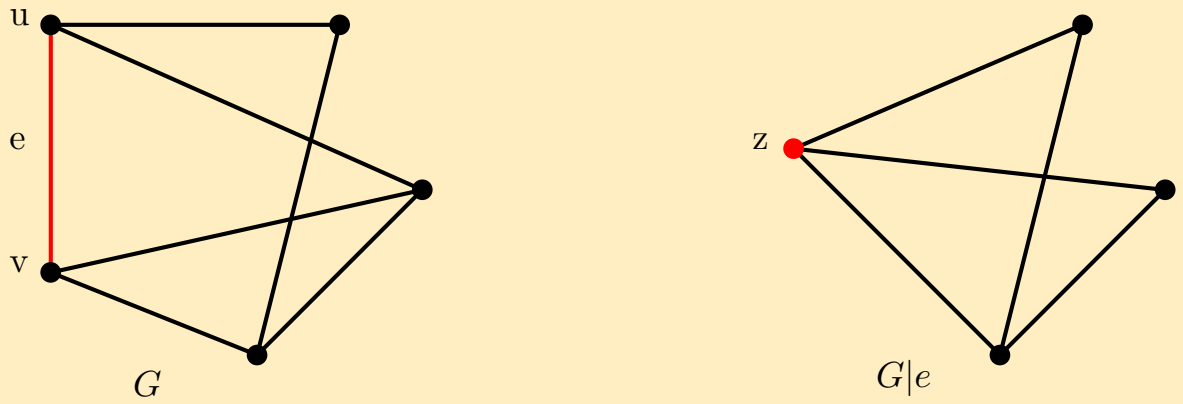
Operaţie unară: $G = (V(G), E(G))$

- **Graful obţinut din G prin contracţia 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



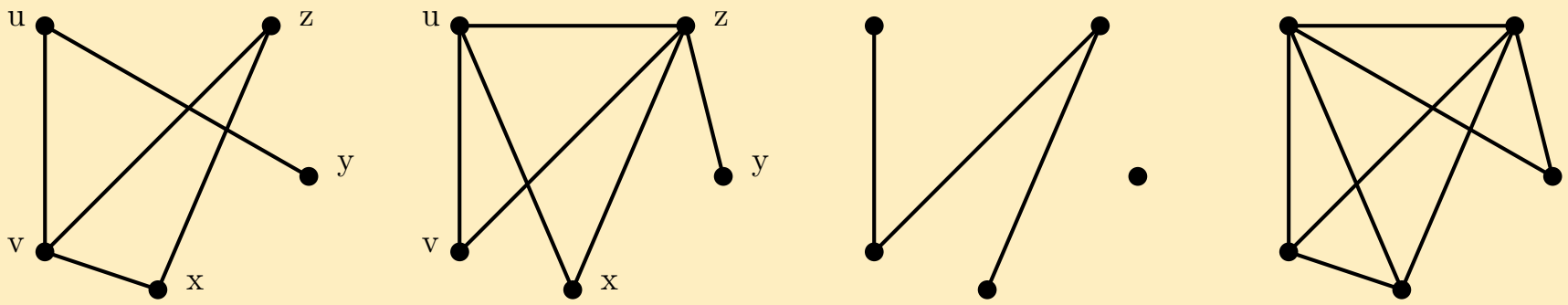
Operaţii cu grafuri

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

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

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

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



G $\quad$ G' $\quad$ $G \cap G'$ $\quad$ $G \cup G'$

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

Operaţii cu grafuri

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

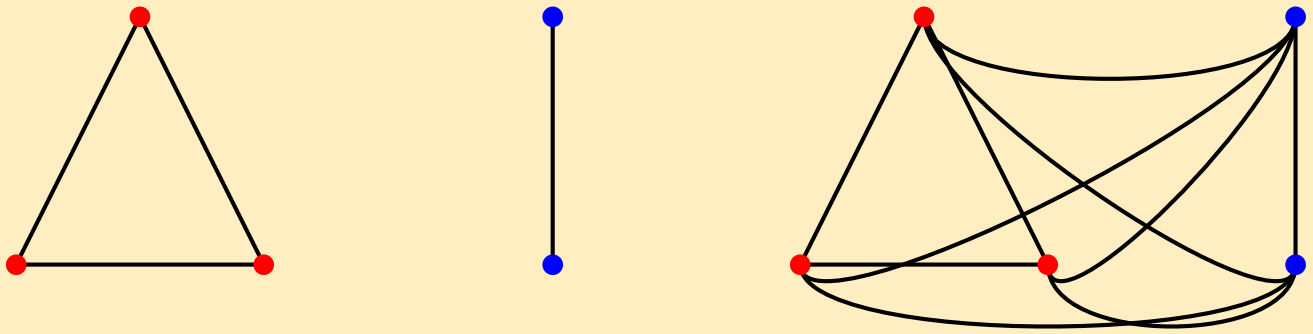
Operaţii cu grafuri

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

Operaţie binară: G, G' cu $V(G) \cap V(G') = \varnothing$

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

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



G G' $G + G'$

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

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

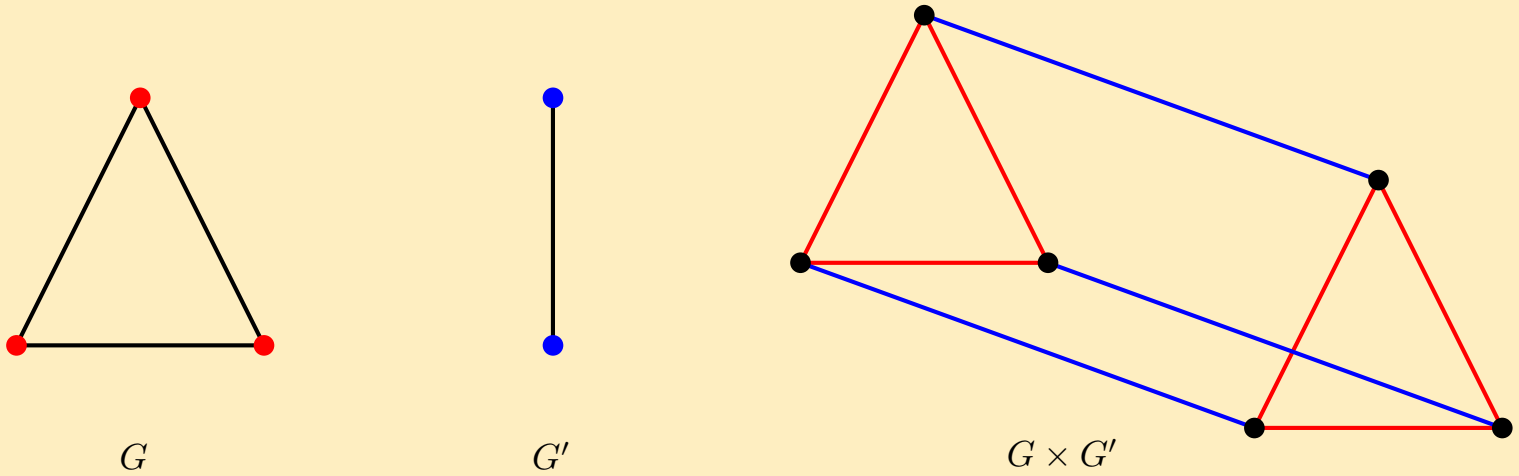
Operaţie binară: G, G' cu $V(G) \cap V(G') = \varnothing$

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

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

$$E(G \times G') = \{(u, u')(v, v') \,:\, u, v \in V(G), u', v' \in V(G'),$$
$$u = v \text{ şi } u'v' \in E(G') \text{ sau } u' = v' \text{ şi } uv \in E(G)\}.$$

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

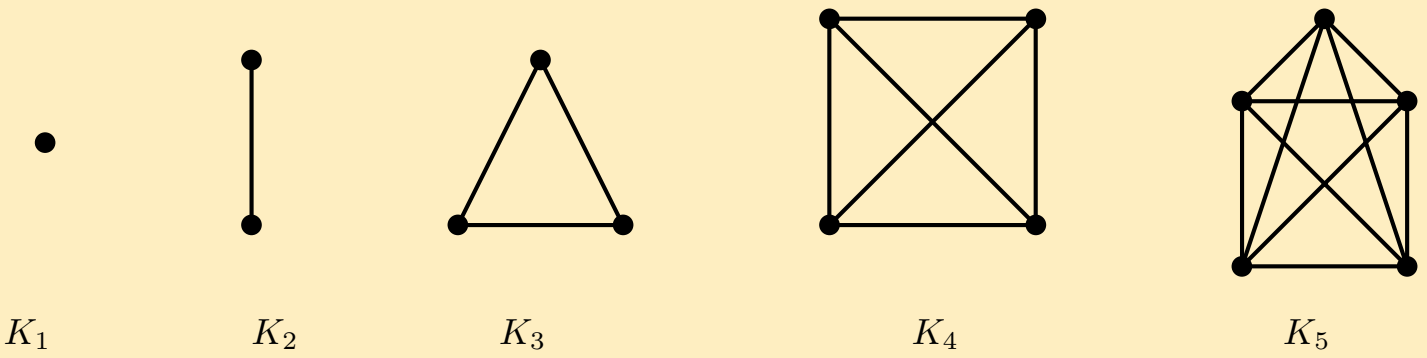


Clase de grafuri - Grafurile complete

C. Croitoru - Graph Algorithms * C.

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

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



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

Clase de grafuri - Grafurile nule

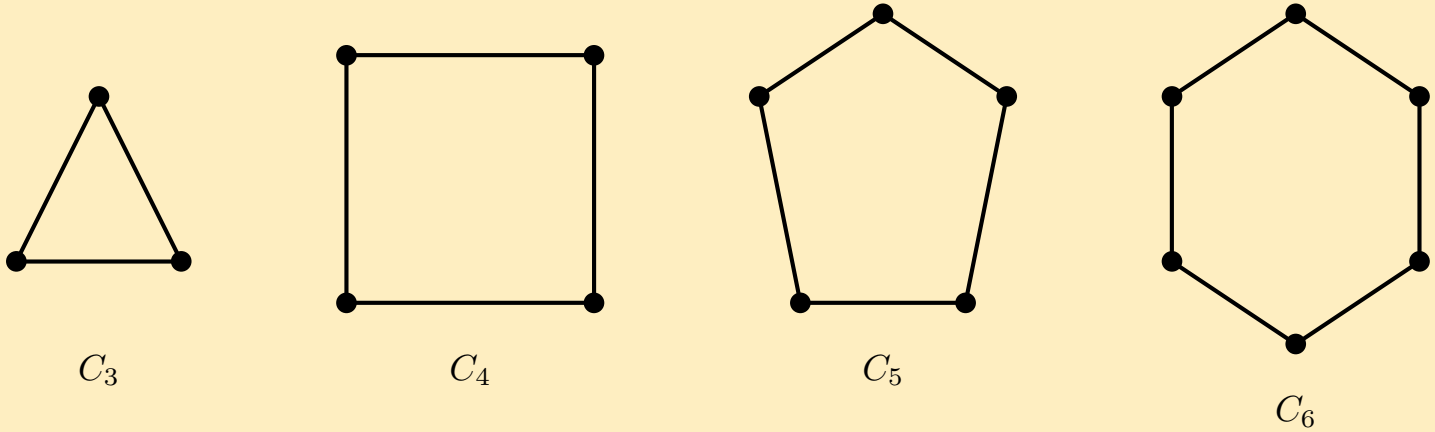
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Clase de grafuri - Circuitele C_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

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

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



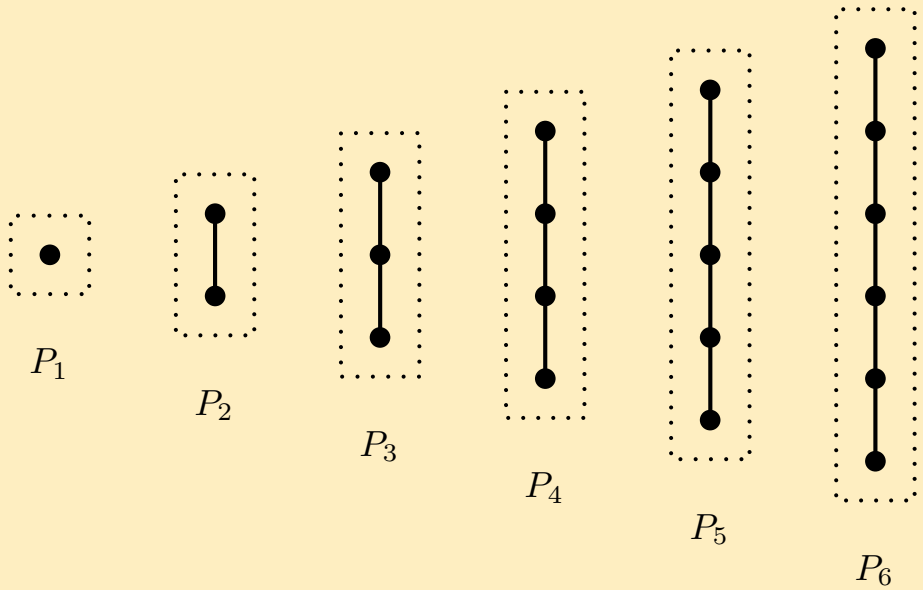
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.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

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: $V(P_n) = \{1, 2, \ldots, n\}$ şi $E(P_n) = \{12, 23, \ldots, n-1n\}$.

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



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

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

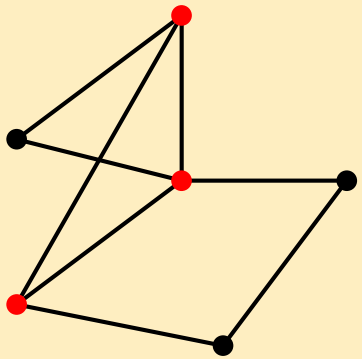
O submulţime de k noduri a graf G care induce un graf complet este numită o **k-clică**.

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

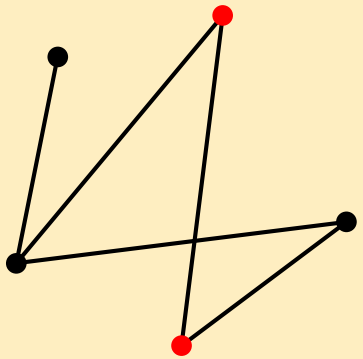
Remarcăm că $\omega(G) = \alpha(\overline{G})$.

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

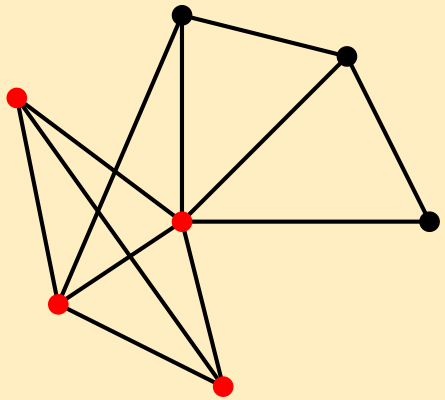
Exemplu



$\omega(G) = 3$



$\omega(G) = 2$



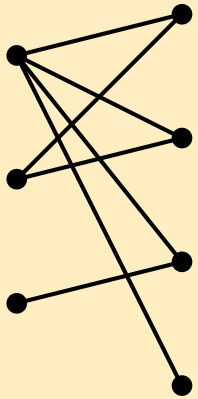
$\omega(G) = 4$

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

Graf bipartit: un graf G cu proprietatea că $V(G)$ poate fi partiţionat în două clase care sunt mulţimi stabile.
Dacă $V(G) = S \cup T$, $S \cap T = \varnothing$, $S, T \neq \varnothing$, cu S şi T mulţimi stabile în G, atunci G este notat $G = (S, T; E(G))$.
Graf bipartit complet: $G = (S, T; E(G))$, cu $uv \in E(G)$, $\forall u \in S$ şi $\forall v \in T$; se notează cu $K_{s,t}$, unde $s = |S|$, $t = |T|$.

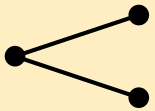
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exemplu

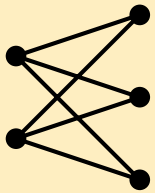


A bipartite graph

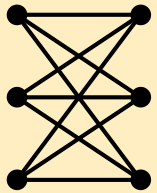
$K_{1,1}$



$K_{1,2}$



$K_{2,3}$



$K_{3,3}$

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 fiecărui nod să îi corespundă un punct al acelui plan şi fiecărei muchii să îi corespundă o curbă simplă care uneşte punctele corespunzătoare extremităţilor şi aceste curbe se intersectează doar în extremităţile lor. Un graf care nu este planar este numit **graf ne-planar**.

Grafuri planare: **Problemă de decizie**

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

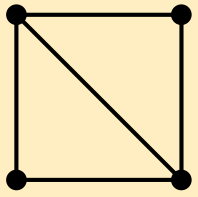
PLAN aparţine clase de complexitate **P** (Hopcroft, Tarjan, 1972, $\mathcal{O}(n+m)$).

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

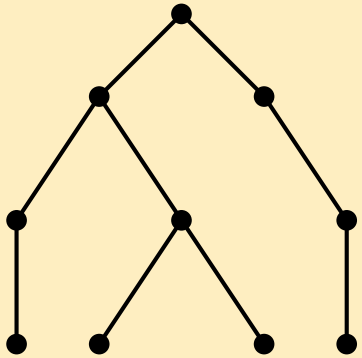
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru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

Exemplu

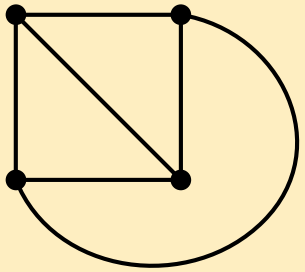
Grafuri planare.



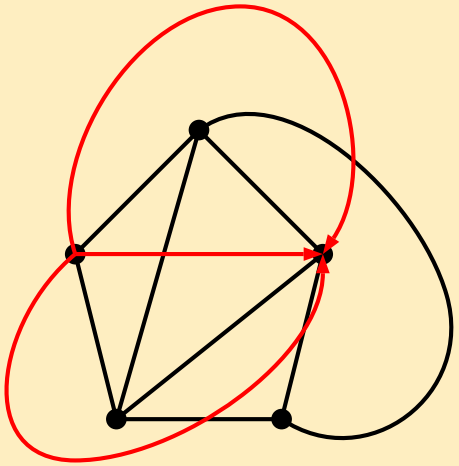
planar graph



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 obişnuită de a defini o clasă de grafuri prin interzicerea unor anumite subgrafuri.
- Dacă $\mathcal{F}$ este o mulţime de grafuri atunci un graf G este **$\mathcal{F}$-free** dacă G nu conţine niciun subgraf indus 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 $(C_k)_{k \geqslant 4}$-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 *

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

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

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

  u şi v sunt extremităţile mersului.
- **Parcurs**: un mers cu muchii distincte.
- **Drum**: un mers cu noduri distincte.
  Un nod este un mers (parcurs, drum) de lungime 0.

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

Drumuri şi 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.

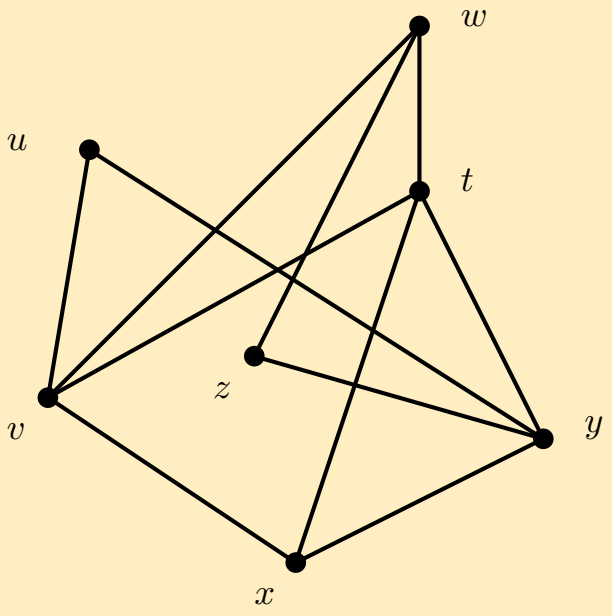
- **Mers închis**: un mers de la u la u.
- **Circuit** (**drum închis**): un mers cu noduri care sunt distincte cu excepţia extremităţilor care coincid.
- Un circuit este par sau impar în funcţie de paritatea lungimii sale.
- Lungimea celui mai scurt circuit (dacă există) este graţia, $g(G)$, lui G.
- Lungimea celui mai lung circuit este circumferinţa, $c(G)$, 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 *

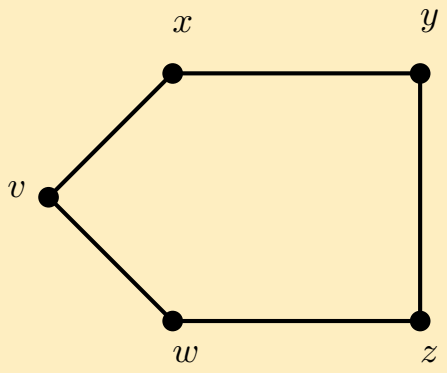
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

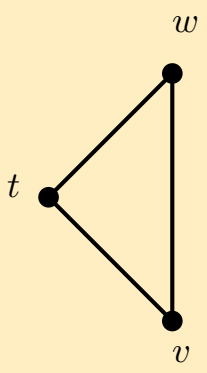
Exemplu

Mersuri închise, circuite.

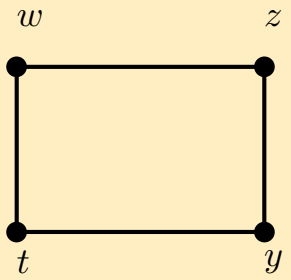


odd circuits:





even circuit:



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

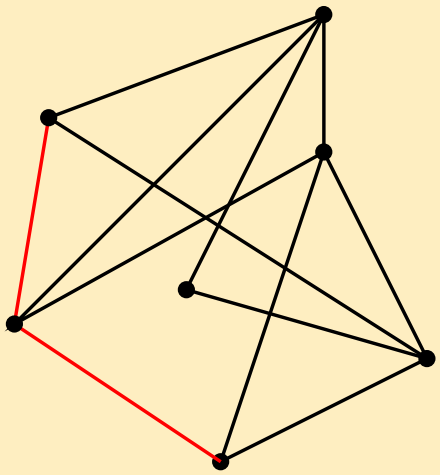
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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

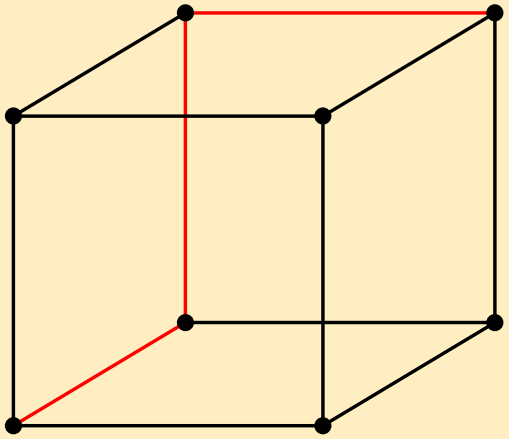
- **Distanţa în G de la u la v**, $d_G(u, v)$ lungimea celui mai scurt drum în G de la u la v (dacă există un astfel de drum).
- **Diametrul unui graf G**, $d(G)$:

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

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



$d(G) = 2$



$d(G) = 3$

Drumuri şi 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

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 relaţie binară este o relaţie de echivalenţă: $\rho \subseteq V \times V$, dată prin $u\rho v$ (i. e., $(u, v) \in \rho$) dacă există un drum în G între u şi v.
- Componentele conexe ale lui G sunt subgrafurile induse de clasele de echivalenţă ale relaţiei ρ.

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

Drumuri şi 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 şi circuite - Conexiune

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

Drumuri şi 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.

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

- **Punct de articulaţie (cut-vertex)**: un nod $v \in V$ astfel că $G - v$ este neconex.
- **Mulţime de articulaţie (vertex cutset)**: o mulţime of noduri $S \subseteq V$ aşa 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

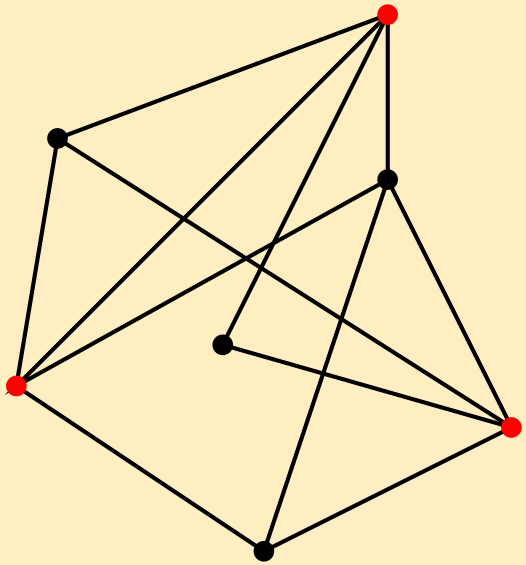
- Pentru $p \in \mathbb{N}^*$, G este **graf p-conex** dacă
  - $|V| = p$ şi $G = K_p$ sau
  - $|V| \geqslant p + 1$ şi G nu are mulţime de articulaţie de cardinal $< p$ (G nu poate fi deconectat prin ştergerea a mai puţin de p noduri).
- Evident, G este 1-conex dacă şi numai dacă este conex.
- **Numărul de conexiune pe noduri**, $k(G)$, al unui graf G este

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

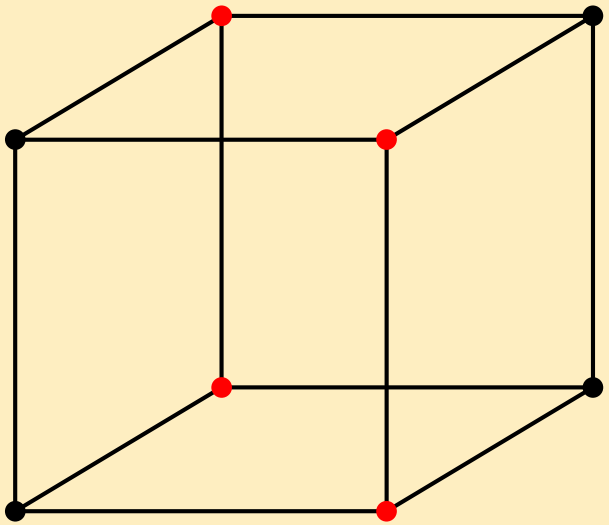
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru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

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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.
- **Mulţime de muchii de articulaţie (tăietură sau edge-cutset)**: O submulţime de muchii $S \subseteq E$ aşa încât $G - S$ este neconex.
- Pentru $p \in \mathbb{N}^*$, G este **graf p-muchie-conex** dacă G nu are o mulţime de muchii de articulaţie de cardinal $< p$ (G nu poate fi deconectat prin ştergerea a mai puţin de p muchii).
- **Numărul de conexiune pe muchii**, $\lambda(G)$, al unui graf G este

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

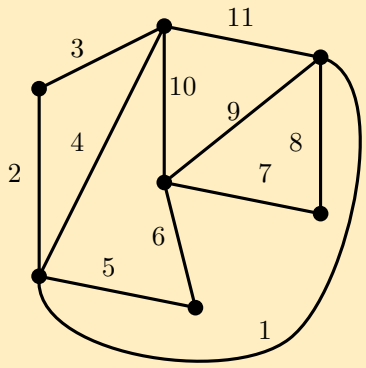
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms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

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

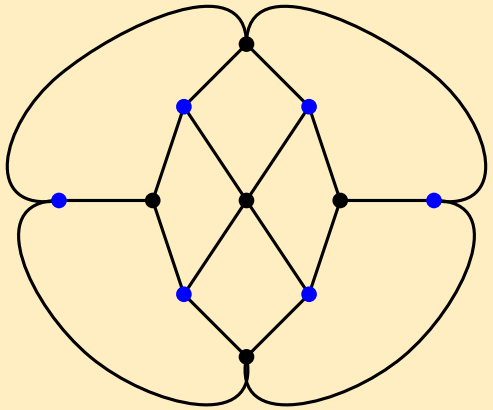
Fie G un (di)graf.

- 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.

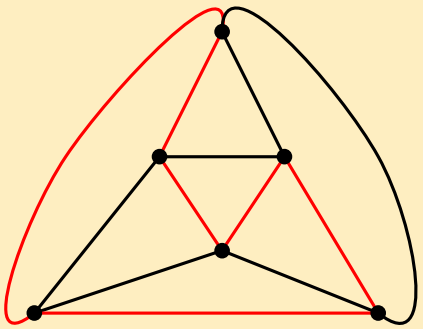
Recunoaşterea (di)grafurilor Euleriene se face în timp polinomial **(Euler, 1736)**.



an Eulerian graph



a non Hamiltonian graph
(bipartite of odd order)



a Hamiltonian 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

Probleme Hamiltoniene

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

NP-completă **(Karp, 1972)**.

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

$\mathbf{NH} \in \mathbf{NP}$?

* 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 *

Exerciţii 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.

Exerciţiul 1. Fie G_1 şi G_2 două grafuri. Arătaţi că dacă $G_1 \times G_2$ este conex, atunci G_1 şi G_2 sunt conexe.

Exerciţiul 2.
Fie $P(n) =$" În orice graf cu cel puţin n noduri există noduri distincte care sunt două câte două adiacente sau două câte două neadiacente."
Arătaţi 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 *

Exerciţii 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 -

Exerciţiul 3.

Fie D un turneu conţinând un circuit C de lungime$n \geqslant 4$. Arătaţi 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.

Exerciţiul 4.

Fie G un graf conex cu $n \geqslant 2$ noduri şi m muchii. Arătaţi că:

a) Dacă G are exact un circuit, atunci $m = n$.

b) Dacă G nu are frunze, atunci $m \geqslant n$.

c) Dacă G este a arbore, atunci are cel puţin două frunze.

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

Exerciţii pentru seminarul de săptămâna 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 -

Exerciţiul 5

Fie G un graf cu $n \geqslant 2$ noduri. Arătaţi că:

a) Dacă G este conex, atunci conţine cel puţin un nod care nu este punct de articulaţie.

b) Dacă $n \geqslant 3$, atunci G este conex dacă şi numai dacă conţine două noduri care nu sunt puncte de articulaţie.

Exerciţiul 6

Fie G un graf conex care nu conţine două noduri pendante (frunze) cu un vecin în comun. Arătaţi că există două noduri adiacente prin ştergerea 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 * C. Croitoru - Graph Algorithms *

Exerciţii 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

Exerciţiul 7

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

Exerciţiul 8

Fie G un graf. Arătaţi că:

a) Dacă G are exact două noduri de grad impar, atunci aceste două noduri sunt unite printr-un drum în G.

b) Dacă G este conex cu toate nodurile de grad par, atunci G are o muchie care nu este punte (ştergerea ei nu deconectează graful).

c) Dacă G este conex cu toate nodurile de grad par, atunci nicio muchie a lui G nu este punte.

Exerciţii pentru seminarul de săptămâna viitoare

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

Exerciţiul 9

Fie G un graf. Arătaţi că

a) Numărul de noduri de grad impar este par.

b) Dacă G este conex şi are k noduri de grad impar, atunci G este o reuniune $\lceil k/2 \rceil$ parcursuri disjuncte pe muchii.

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

Exerciţiul 10

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

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

Exerciţiul 11

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

Exercises for the next week seminar

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

Exerciţiul 12

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

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

Exerciţiul 13

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

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

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

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

Cuprins

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

C. Croitoru - Graph Algorithms *

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

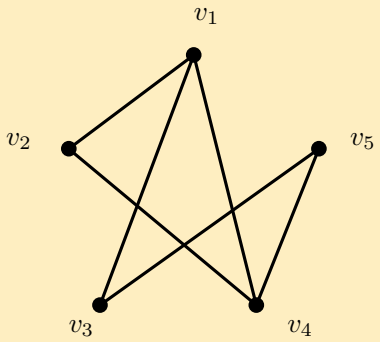
Fie G un graf cu $V(G) = \{v_1, \ldots, v_n\}$. **Matricea de adiacenţă** a lui G este matricea $A = (a_{ij})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_{n \times n}(\{0,1\})$, unde

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

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

Exemplu

Un graf şi matricea sa de adiacenţă.



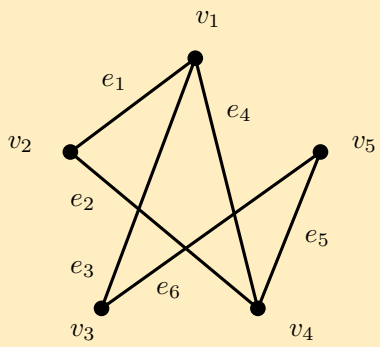
$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Fie G un graf cu $V(G) = \{v_1, \ldots, v_n\}$ şi $E(G) = \{e_1, \ldots, e_m\}$. **Matricea de incidenţă** a lui G este matricea $B = (b_{ij})_{1 \leqslant i,j \leqslant n} \in \mathcal{M}_{n \times m}(\{0,1\})$, unde

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

Exemplu

Un graf şi matricea sa de incidenţă.



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

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

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

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

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

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

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

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

Structuri de date pentru matricea de adiacenţă

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $V = \{1, 2, \ldots, n\}$.

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

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

Structuri de date pentru listele de adiacenţă

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

- 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 adiacenţă: unul în $A(u)$ şi unul în $A(v)$; spaţiul necesar este $\mathcal{O}(n + 2e)$.
- Dacă G este digraf, atunci spaţiul necesar este $\mathcal{O}(n + e)$.
- Listele de adiacenţă pot fi implementate folosind **liste înlănţuite** sau tablouri.
- Testarea dacă un nod u este adiacent cu un altul v în G necesită $\Omega(d_G(u))$ timp, dar **parcurgerea întregii mulţimi de vecini** $N_G(u)$ **(sau** $N_G^+(u)$**)**, pentru un nod $u \in V$, se poate face în $\Omega(d_G(u))$ (şi nu în $\mathcal{O}(n)$ ca în cazul matricei de adiacenţă).

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 mulţimea nodurilor care pot fi accesate (atinse) pe drumuri plecând dintr-un nod fixat al (di)grafului.

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

generează "eficient" muţimea

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

G va fi reprezentat cu liste de adiacenţă, deoarece, în timpul procesului de traversare, trebuie manipulată în mod eficient mulţimea 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 conţinând s;
while $Q \neq \varnothing$ **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 * C. Croitoru - Graph Algorithms *

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

Proprietăţi ale BFS. Nu este dificil de arătat că:

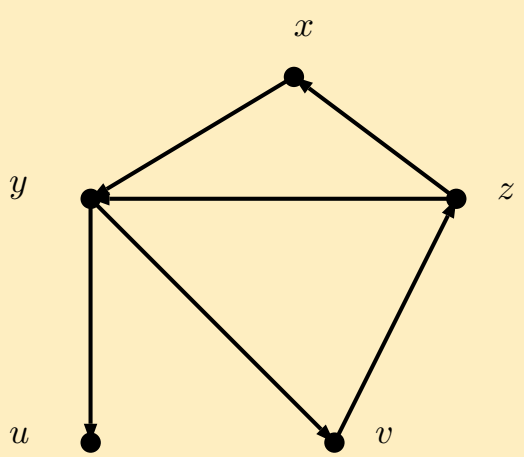
- $S = \{u \in V \ : \ label(u) \geqslant 0\}$;
- $\forall u \in V, label(u) = d_G(s, u)$ (distanţa în G de la s la u);
- Variabila parent defineşte **arborele bfs** asociat parcurgerii din s: dacă G este graf atunci arborele bfs este un arbore parţial al componentei conexe conţinând s; dacă G este digraf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

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

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

Exemplu

Două parcurgeri BFS ale unui aceluiaşi digraf (începând din z şi din v):



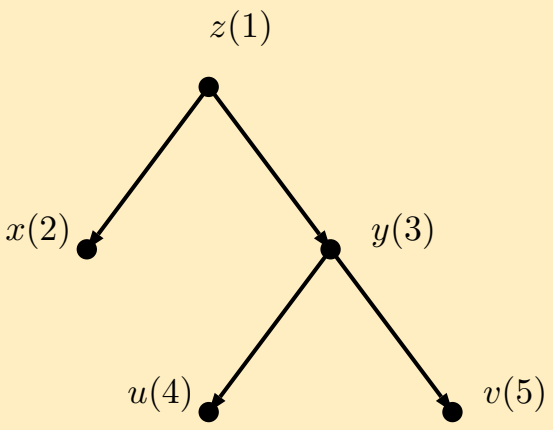
$A(x) = [y]$

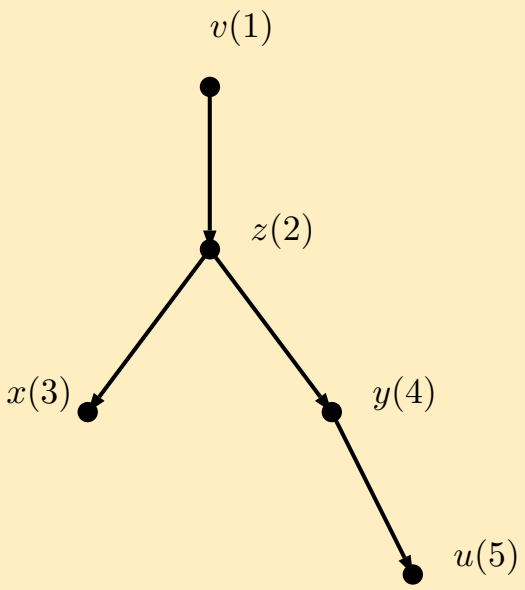
$A(y) = [u, v]$

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

$A(u) = \emptyset$

$A(v) = [z]$





(Am marcat în paranteze ordinea de vizitare.)

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

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

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 conţinând s; $n_S \leftarrow 0$;
while $S \neq \varnothing$ **do**
  $u \leftarrow top(S)$;
  if $((v \leftarrow next[A(u)]) \neq NULL)$ **then**
    if $label(v) < 0$ **then**
      n_S++; $label(v) \leftarrow n_S$;
      $parent(v) \leftarrow u$; $push(S, v)$;
  else
    $delete(S, u)$;

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

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

Proprietăţi ale DFS. Nu este dificil de arătat că:

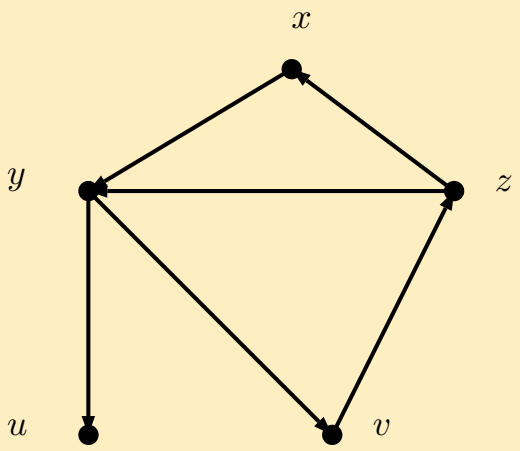
- $\{u \in V \ : \ label(u) \geqslant 0\}$ este exact muţimea S de noduri accesibile pe drumuri din s;
- $\forall u \in V, label(u) =$ momentul vizitării lui u (s este vizitat la momentul 0);
- Variabila parent defineşte **arborele dfs** asociat parcurgerii din s
- Complexitatea timp a $DFS(s)$ este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

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 aceluiaşi digraf (începând din z şi din x):



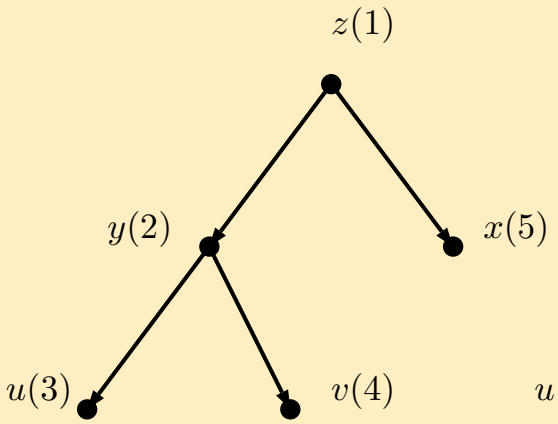
$A(x) = [y]$

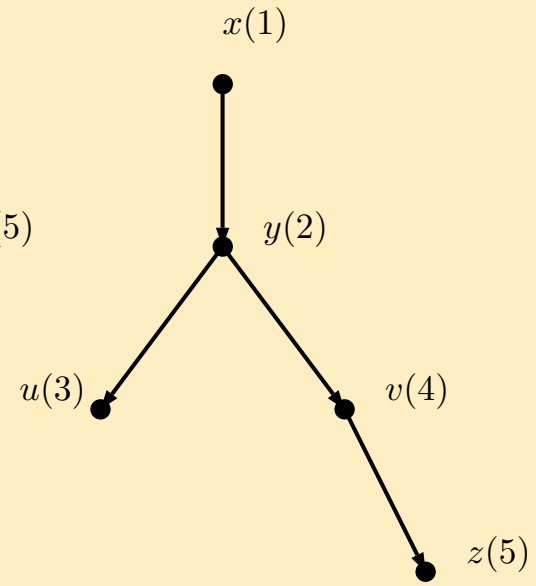
$A(y) = [u, v]$

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

$A(u) = \emptyset$

$A(v) = [z]$





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

Drumuri de cost minim - Notaţii

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $V = \{1, \ldots, n\}$.

- 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 adiacenţă, atunci $a(ij)$ este un câmp al elementului din lista de adiacenţă a lui i (reprezentând un arc ij).
- Pentru uşurinţa notaţiei vom folosi reprezentarea lui G cu o matrice de cost-adiacenţă $A = (a_{ij})_{1 \leqslant i,j \leqslant n}$, unde

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

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

Drumuri de cost minim - Notaţii

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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., $\infty > n \cdot \max_{ij \in E} a(ij)$) şi presupunem că $\infty \pm a = \infty$, $\infty + \infty = \infty$.
- Este de asemeni posibil să folosim ∞ ca un acces nereuşit la structura de date utilizată pentru reprezentarea matricei A.
- Pentru $i, j \in V$, mulţimea 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 - Notaţii

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

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

- 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 * 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; $a : E \to \mathbb{R}$; $s, t \in V, s \neq t$.
Găsiţi $P^*_{st} \in \mathcal{P}_{st}$, aşa încât $a(P^*_{st}) = \min\{a(P_{st}) \ : \ P_{st} \in \mathcal{P}_{st}\}$.

Drumurile de cost minim dintre un nod şi toate celelalte noduri

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

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

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

Remarci 1

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

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

1. Reţele de comunicare (Communication Networks). Digraful $G = (V, E)$ reprezintă o reţea de comunicaţii între nodurile din V, iar E modelează mulţimea legăturilor orientate dintre noduri.

- Dacă $a(e) \geqslant 0$ $(\forall e \in E)$ reprezintă lungimea conexiunii directe dintre extremităţile lui e, atunci problemele **P1 - P3** sunt probleme de drumuri de cost minim naturale.
- Dacă $a(e) \geqslant 0$ $(\forall e \in E)$ reprezintă timpul necesar pentru parcurgerea conexiunii directe dintre extremităţile 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 funcţionării corecte a conexiunii directe dintre extremităţile lui e şi dacă presupunem că muchiile funcţionează independent una de cealaltă, atunci problemele **P1 - P3** sunt probleme pentru determinarea drumurilor celor mai sigure:

Drumuri de cost minim - Aplicaţii

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 funcţioneze corect este (datorită independenţei)

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

Luând $a'(e) = -\log a(e)$,

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

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

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

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

2. Reţele PERT - Metoda drumului critic (Critical path method - CPM). PERT (**P**ath **E**valuation and **R**eview **T**echnique) este o metodă de a analiza (îndeosebi) îndeplinirea fiecărei sarcini dintr-un proiect mai complex.

- Fie $P = \{A_1, A_2, \ldots, A_n\}$ activităţile atomice ale unui mare proiect P (n este foarte mare). $(P, <)$ este o mulţime parţial ordonată, unde $A_i < A_j$ dacă $i \neq j$ şi activitatea A_j poate începe doar după ce activitatea A_i s-a terminat.
- Pentru fiecare activitate A_i, timpul necesar finalizării t_i este cunoscut (sau doar estimat).

Găsiţi o planificare a activităţilor acestui proiect care să minimizeze durata totală până la finalizare.

Drumuri de cost minim - Aplicaţii

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 ($p \in \{1, \ldots, n\}$) adăugăm un arc $i_p j_p$ de cost $a(i_p j_p) = t_p$;
- nodul i_p coresponde startului activităţii A_p iar nodul j_p este asociat finalizării ei;
- dacă activitatea A_k poate porni doar după activitatea A_p adăugăm arcul $j_p i_k$ (o activitate fictivă - dummy activity).

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

Drumuri de cost minim - Aplicaţii

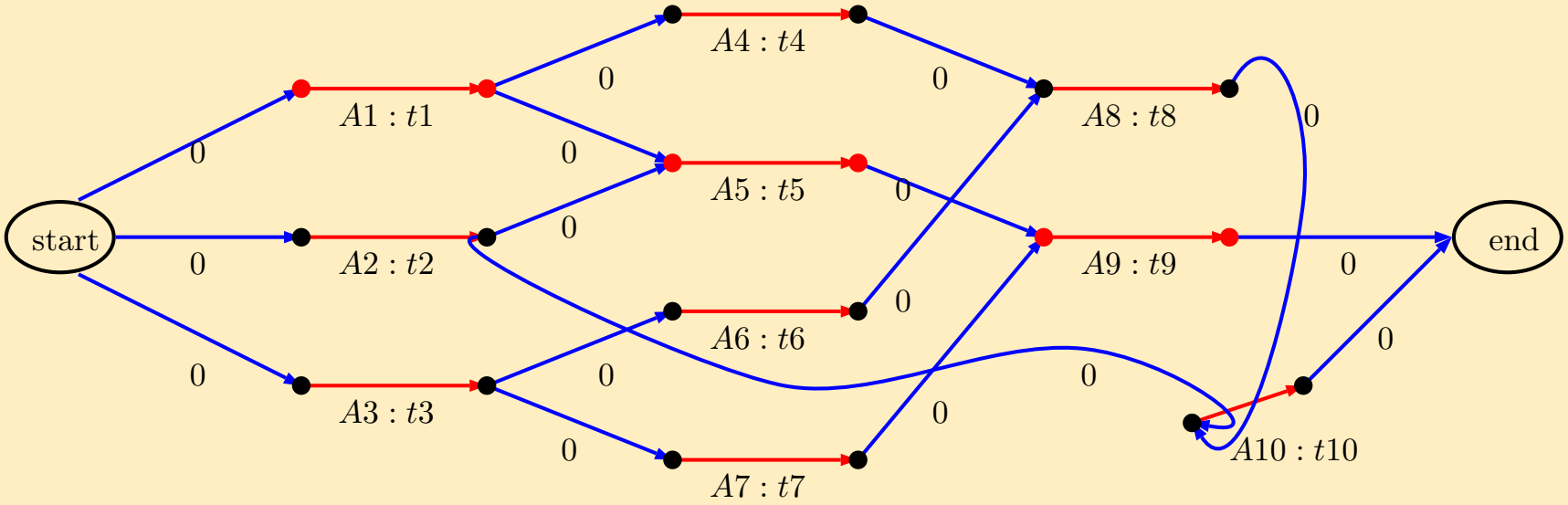
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

- Construcţia digrafului este încheiată după adăugarea unui nod s corespunzând momentului iniţial al proiectului, legat prin arce si_p pentru fiecare activitate A_p în care nu intră vreun arc şi a unui nod t corespunzând finalizării proiectului, legat prin arce $j_p t$ pentru fiecare activitate A_p din care nu iese vreun arc.
- În digraful obţinut **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 activităţi de pe acest drum implică o întârziere a întregului proiect.

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

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

Exemplu



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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $b \in \mathbb{N}$, şi n obiecte de dimensiuni $a_1, \ldots, a_n \in \mathbb{N}$. Se cunoaşte şi profitul $p_i \in \mathbb{N}$ al adăugării obiectului i ($i \in \{1, \ldots, n\}$) în rucsac. Se cere să se găsească o completare a rucsacului care să maximizeze profitul total.

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

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

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 * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Aplicaţii

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

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

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

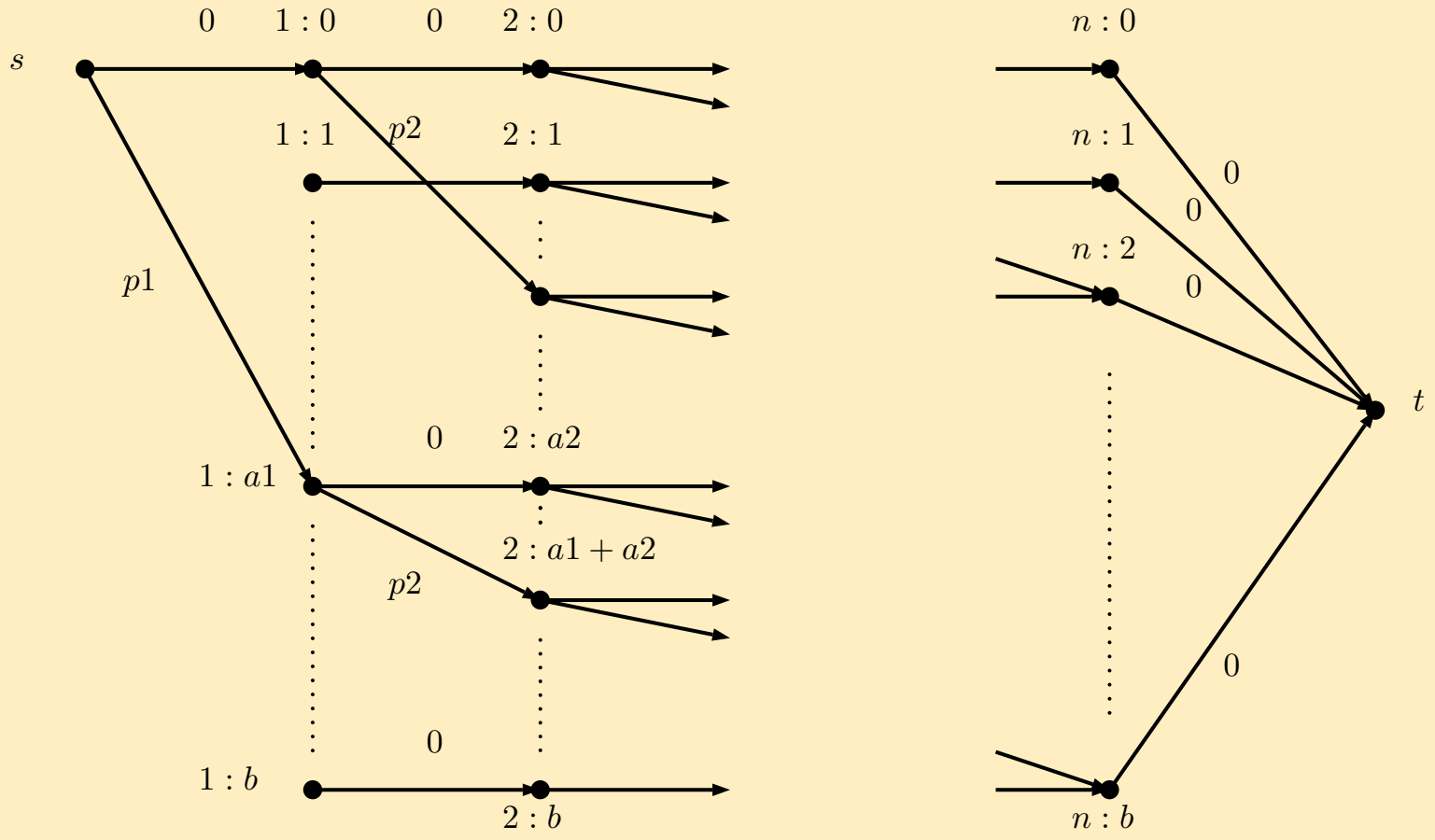
- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.

Remarci 2

- Fiecare drum de la s la t în G corespunde unei submulţimi de obiecte cu nivelul de umplere $\leqslant b$ şi cu profitul total egal cu costul drumului. Deoarece, reciproc, fiecărei umpleri a rucsacului îi corresponde 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-una procedurală, folosind programarea dinamică. Problema aceasta este **NP**-hard (ordinul lui G poate fi exponenţial în dimensiunea problemei).

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

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

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

Teorema 1

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

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

Atunci $(u_1, \ldots, u_n)$ este o soluţie a sistemului de ecuaţii

(B) $$\begin{cases} u_s = & 0 \\ u_i = & \min\limits_{j \neq i}(u_j + a_{ji}) \end{cases} \quad \textit{dacă şi numai dacă}$$

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 soluţie optimă a problemei **P2** şi $u_i = a(P^*_{si})$. Ipoteza (I) implică $u_s = 0$, i.e., prima ecuaţie 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

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

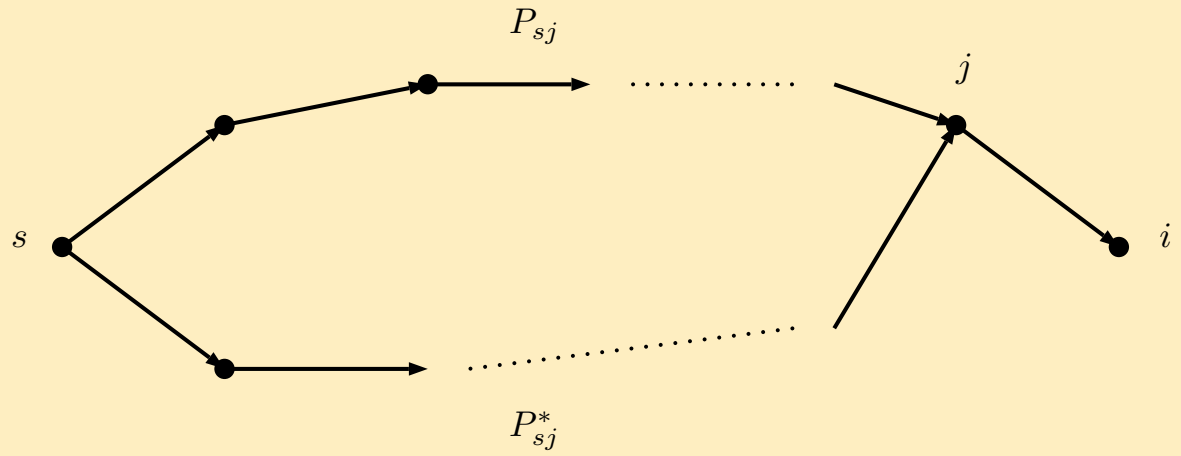
Arătăm că $u_i = u_j + a_{ji}$. Presupunem că $u_i > u_j + a_{ji}$, i. e., $a(P_{sj}) > a(P^*_{sj})$.

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 *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $P^1 = P^*_{sj} \circ (j, ji, i) \in \mathcal{P}_{si}$ şi $a(P^1) = a(P^*_{sj}) + a_{ji} < a(P_{sj}) + a_{ji} = a(P^*_{si})$, contradicţie ($P^*_{si}$ este un drum de cost minim).

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

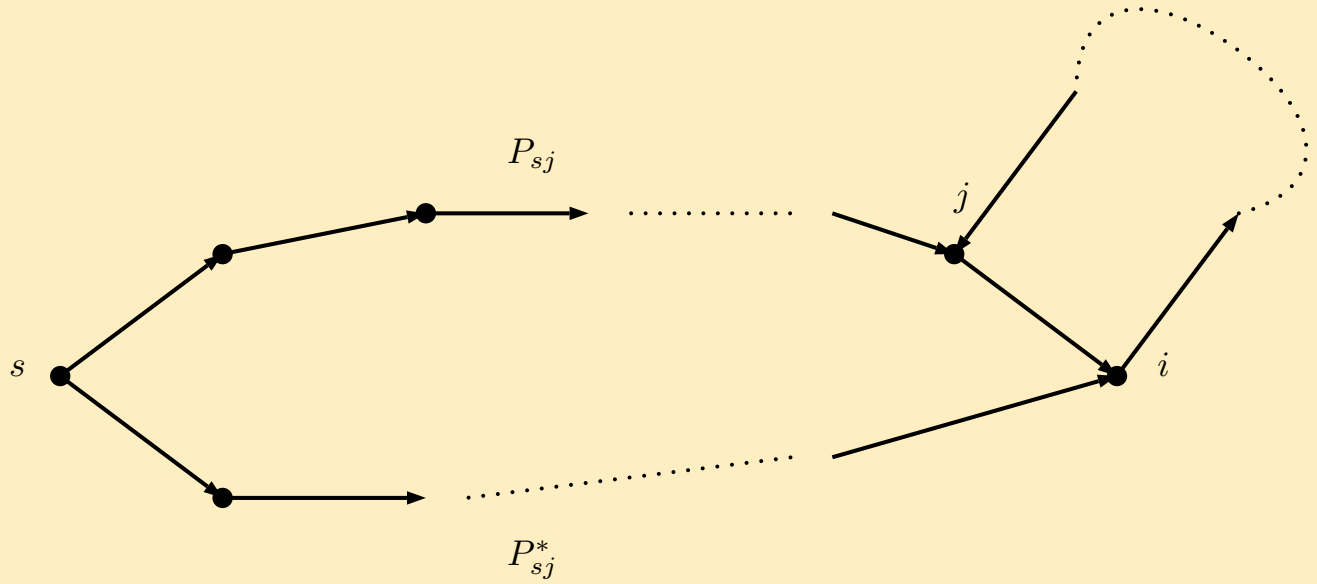


Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

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

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

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



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

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 $P^*_{si} = P_{sj} \circ Pji$, 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 *

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 $(u_1, \ldots, u_n)$ este o soluţie a sistemului (B), atunci

(a) $\exists P_{si} \in \mathcal{P}_{si}$ aşa 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 $u_s = 0$ şi drumul P_{ss} satisface $a(P_{ss}) = 0 = u_s$.
Dacă $i \neq s$, considerăm următorul algoritm

$v \leftarrow i$; $k \leftarrow 0$;
while $v \neq s$ **do**
  find w a. î. $u_v = u_w + a_{wv}$; // $\exists w$ deoarece u_v satisface (B)
  $i_k \leftarrow v$; $k++$; $v \leftarrow w$;

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

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

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

$$a(P) = a(i_{k+1}i_k) + \cdots + a(i_1 i_0) =$$

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

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

În fiecare iteraţie while $w \notin \{i_0, \ldots, i_{k-i}\}$ (altfel obţinem un circuit de cost 0, în contradicţie cu ipoteza (I)).

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

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

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

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

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

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

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

$\square$

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 demonstraţia de mai sus urmează că pentru a rezolva **P2** este suficient să se găsească o soluţie a sistemului de ecuaţii (B). Drumurile de cost minim corespunzătoare pot fi obţinute ca la (a) din demonstraţia 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) menţinem un tablou $before[1..n]$ cu elemente din $V \cup \{0\}$ cu semnificaţia finală $before[i]$ = nodul dinaintea lui i pe un drum de cost minim de la s la i. Nodurile de pe acest drum pot fi determinate în $\mathcal{O}(n)$ prin construcţia secvenţei $i, before[i], before[before[i]], \ldots, s$.

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 rezolva sistemul de ecuaţii (B) ocolesc (prin întreţinerea tabloului *before*) circuitele de cost 0, atunci problema **P2** este rezolvată, chiar dacă se pierde unicitatea soluţiei. Astfel, aceşti algoritmi vor rezolva **P2** în ipoteza

(I') $\qquad a(C) \geqslant 0$, pentru toate circuitele C 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 şi nu un digraf, putem folosi algoritmii pentru digrafuri înlocuind fiecare muchie a lui G cu o pereche simetrică de arce, fiecare cu acelaşi cost ca muchia. Această abordare funcţionează doar pentru muchii de cost nenegativ (dacă o muchie are cost negativ, atunci 2-circuitul corespunzător format cu cele două arce simetrice are cost negativ, deci ipoteza (I') este încălcată).

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 muţimile P_{ij} sunt finite (şi nevide), putem considera probleme similare cu **P1** - **P3** înlocuind min cu max.
- Relaţia 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 $a(C) \leqslant 0$ (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 *

Exerciţii 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 -

Exerciţiul 1.

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

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

Exerciţiul 2.

Arătaţi că nu există o ordonare $e_1, e_2, \ldots, e_{10}$ a muchiilor unui graf K_5, aşa încât: e_{10} şi e_1 nu sunt adiacente şi e_i şi e_{i+1} nu sunt adiacente pentru orice $1 \leqslant i \leqslant 9$.

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

Exerciţii pentru seminarul de săptămâna 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 -

Exerciţiul 3. Fie $G = (V, E)$ un graf de ordin n şi dimensiune m cu matricea de adiacenţă A. Din mulţimea celor 2^m orientări posibile ale tuturor muchiilor sale alegem una şi considerăm matricea de incidenţă nod-arc $Q \in \mathcal{M}_{n\times m}(\{-1, 0, 1\})$.

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

Arătaţi că $A + QQ^T$ este matrice diagonală şi aflaţi interpretarea combinatorială a elementelor sale diagonale.

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

Exerciţii 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

Exerciţiul 4. Fie $D = (V, E)$ un digraf cu $V = \{v_1, v_2, \ldots, v_n\}$ şi $E = \{e_1, e_2, \ldots, e_m\}$. Fie $B = (b_{ij}) \in \mathcal{M}_{n\times m}(\{-1, 0, 1\})$ matricea de incidenţă a lui D, unde

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

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

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

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

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

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

Arătaţi 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.

Exerciţiul 6. Arătaţi că un digraf are o singură ordonare topologică dacă şi numai dacă are un drum Hamiltonian iar toate celelate arce sunt orientate înainte relativ la direcţia de parcurgere a acestui drum.

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

Exerciţii 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

Exerciţiul 7. Fie M_G matricea de incidenţă muchie-nod a unui graf dat $G = (V, E)$, adică $M_G = (m_{ij})_{\substack{1 \leqslant i \leqslant m \\ 1 \leqslant j \leqslant n}}$, unde

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

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

(a) Arătaţi că dacă T este un arbore, atunci prin îndepărtarea din M_T a unei coloane corespunzătoare unui nod dat se obţine o matrice pătratică nesingulară.

(b) Arătaţi că dacă C este un circuit, atunci M_C este matrice nesingulară dacă şi numai dacă C este impar.

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

Exerciţii 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

Exerciţiul 8. Diametrul unui graf G este cea mai mare distanţa dintre any două noduri din G. Două noduri formează o pereche diametrală de noduri dacă distanţa dintre ele este cât diametrul. Arătaţi 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).

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

Exerciţii pentru seminarul de săptămâna 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

Exerciţiul 9. Arătaţi că o parcurgere DFS poate fi utilizată pentru a determina un circuit par într-un graf 3-regulat în $\mathcal{O}(n)$.

Exerciţiul 10.

(a) Arătaţi că pentru un graf bipartit cu n noduri şi m muchii avem $4m \leqslant n^2$.

(b) Descrieţi un algoritm de complexitate timp $\mathcal{O}(n+m)$ care să decidă dacă un graf dat (cu n noduri şi m muchii) este complementul unui graf bipartit.

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

Exerciţii pentru seminarul de săptămâna viitoare

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

Exerciţiul 11. Demonstraţi că un graf G este bipartit dacă şi numai dacă orice subgraf indus H al lui G satisface inegalitatea: $2\alpha(H) \geqslant |H|$.

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

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

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

(a) Arătaţi că G este S-lanţ dacă şi numai dacă este T-lanţ.

(b) Să presupunem că G (care este bipartit) are ordinul n, dimensiunea m şi este reprezentat folosind liste de adiacenţă. Descrieţi un algoritm de recunoaştere a unui S-lanţ de complexitate timp $\mathcal{O}(n+m)$.

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

Exerciţii 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.

Exerciţiul 13. Fie G un graf; notăm cu $b(G)$ graful obţinut din G prin inserarea simultană a câte unui nou nod pe fiecare muchie a lui G.

a) Arătaţi că $b(G)$ este graf bipartit.

b) Demonstraţi că $G \simeq H$ dacă şi numai dacă $b(G) \simeq b(H)$. De aici deduceţi 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Cuprins

1. Probleme de drumuri în digrafuri
   - Drumuri de cost minim - Problema **P2** pentru dags: sortarea topologică
   - Drumuri de cost minim - Problema **P2** pentru costuri nenegative
   - Drumuri de cost minim - Problema **P2** pentru costuri reale
   - Drumuri de cost minim - Rezolvarea problem **P3**
   - Înmulţirea (rapidă) a matricilor
2. Exerciţii pentru seminarul de săptămâna următare

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

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

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

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

Lema 1

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

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

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

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

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

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

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

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

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

□

Din demonstraţia de mai sus obţinem următorul algoritm pentru recunoaşterea unui dag şi construcţia unei ordonări topologice în cazul instanţelor "yes":

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

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

Drumuri de cost minim - Problema **P2** pentru dags

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

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

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

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

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

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

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

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

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

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

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

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

Iniţial $S = \{s\}$ şi în fiecare dintre cei $n-1$ paşi un nod nou este adăugat la S, obţinând $S = V$. Astfel, datorită invariantului (D) de mai sus, **P2** este rezolvată.

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

Algoritmul lui Dijkstra

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

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

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

Demonstraţia corectitudinii algoritmului lui Dijkstra

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

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

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

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

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

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

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

Demonstraţia corectitudinii algoritmului lui Dijkstra (cont.)

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

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

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

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

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

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

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

Demonstraţia corectitudinii algoritmului lui Dijkstra (cont.)

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

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

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

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

□

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

Complexitatea timp a algoritmului lui Dijkstra

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

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

timpul general necesar algoritmului pentru a actualiza valorile u_j este

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

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

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

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

Complexitatea timp a algoritmului lui Dijkstra

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

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

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

Remarci 1

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

  (i) $\forall i \in V,\ u_i + g(i) \leqslant \min\{a(P_{st}) \ :\ P_{st} \in \mathcal{P}_{st}$ şi $i \in V(P_{st})\}$
  (ii) $\forall ij \in E, g(i) \leqslant a_{ij} + g(j)$.
- Evident, $g(i) = 0, \forall i$ este un estimator consistent trivial.

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

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

Remarci 2

- Dacă $V(G)$ este a mulţime de puncte dintr-un spaţiu Euclidean, atunci, luând $g(i) =$ distanţa Euclideană de la i la t, obţinem un estimator consistent dacă sunt satisfăcute şi condiţiile din (ii).
- Dacă g este un estimator consistent, atunci alegerea lui j^* în algoritmul lui Dijkstra se face astfel

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

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

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

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

Remarci 3

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

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

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

Partition Shortest Path (PSP) algoritm

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

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

Algoritmul Bellman-Ford-Moore

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

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

vom rezolva sistemul

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

prin aproximări succesive.

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

G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms

Algoritmul Bellman-Ford-Moore

Definim

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

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

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

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

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

Algoritmul Bellman-Ford-Moore

$u_s^1 \leftarrow 0;$
for $(i \in V \setminus \{s\})$ **do**
  $u_i^1 \leftarrow a_{si}$; //evident (BM) are loc.
for $(k = \overline{1, n-2})$ **do**
  for $(i = \overline{1, n})$ **do**
    $u_i^{k+1} \leftarrow \min\left(u_i^k, \min_{j \neq i}\left(u_j^k + a_{ji}\right)\right);$

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

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

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

Algoritmul Bellman-Ford-Moore

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

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

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

Atunci, $A = B \cup C$, şi

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

Deoarece u_i^k satisface (BM) avem

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

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

Algoritmul Bellman-Ford-Moore

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

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

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

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

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

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

Remarci 4

Algoritmul Bellman-Ford-Moore

- Putem adăuga algoritmului următorul pas:

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

  În acest fel se obţine un test de $\mathcal{O}(n^3)$ dacă digraful G şi funcţia de cost a încalcă condiţia (I') (altfel, din demonstraţia corectitudinii, u_i^{n-1} nu poate fi scăzut). Un circuit de cost negativ poate fi găsit folosind vectorul *before*[].
- Dacă există $k < n - 1$ aşa încât $\mathbf{u}^k = \mathbf{u}^{k+1}$, atunci algoritmul poate fi oprit. Folosind această idee, este posibil să implementăm algoritmul în $\mathcal{O}(nm)$, memorând nodurile i pentru care valoarea u_i se modifică în coadă.

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

P3 Dat $G = (V, E)$ digraf; $a : E \to \mathbb{R}$.
Determină $P^*_{ij} \in \mathcal{P}_{ij}, \forall i, j \in V$, a. î. $a(P^*_{ij}) = \min\{a(P_{ij}) : P_{ij} \in$

- Fie $u_{ij} = \min\{a(P_{ij}) : P_{ij} \in \mathcal{P}_{ij}\}$. Astfel avem de determinat matricea $U = (u_{ij})_{n\times n}$, când matricea de cost-adiacenţă A este dată.
- Fiecare drum de cost minim poate fi obţinut în $\mathcal{O}(n)$ dacă, în timpul construcţiei matricei U, întreţinem matricea $Before = (before_{ij})_{n\times n}$, unde $before_{ij}$ = nodul dinaintea lui j pe drumul de cost minim de la i la j din G.
- Dacă perechea (G, a) satisface condiţia (I'), putem rezolva **P3** aplicând algoritmul Bellman-Ford-Moore pentru $s \in \{1, \ldots, n\}$, în $\mathcal{O}(n^4)$. Există şi soluţii mai eficiente pe care le vom prezenta pe slide-urile următoare.

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

Iterarea algoritmului lui Dijkstra

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

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

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

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

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

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

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

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

Iterarea algoritmului lui Dijkstra

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

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

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

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

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

Iterarea algoritmului lui Dijkstra

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

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

Algoritmul Floyd - Warshall

Fie

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

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

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

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

$$u_{ij}^{k+1} = \min\{u_{ij}^k, u_{ik}^k + u_{kj}^k\}, \forall i, j \in V, k = \overline{1, n}.$$

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

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

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 * C. Croitoru - Graph Algorithms *

Drumuri de cost minim - Rezolvarea problemei **P3**

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

for $(i = \overline{1, n})$ **do**
  for $(j = \overline{1, n}\ )$ **do**
    $before(i, j) \leftarrow i;$
    if $(i = j)$ **then**
      $a_{ii} \leftarrow 0;\ before(i, i) \leftarrow 0;$
for $(k = \overline{1, n}\ )$ **do**
  for $(i = \overline{1, n})$ **do**
    for $(j = \overline{1, n}\ )$ **do**
      if $(a_{ij} > a_{ik} + a_{kj})$ **then**
        $a_{ij} \leftarrow a_{ik} + a_{kj};\ before(i, j) \leftarrow before(k, j);$
        if $(i = j$ şi $a_{ij} < 0)$ **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 ştim că digraful nu are circuite de cost negative, atunci dacă elementele diagonale ale lui A sunt iniţializate cu ∞, valoarea finală a fiecărui element diagonal este costul minim al unui circuit care trece prin nodul corespunzător.

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

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

Înmulţirea (rapidă) a matricilor

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

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

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

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

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

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

Înmulţirea (rapidă) a matricilor

for $(i, j \in V)$ **do**
  if $(i \neq j)$ **then**
    $u_{ij}^0 \leftarrow \infty;$
  else
    $u_{ij}^0 \leftarrow 0;$
for $(k = \overline{0, n-2}\ )$ **do**
  for $(i, j \in V)$ **do**
    $u_{ij}^{k+1} = \min_{h \in V} (u_{ih}^k + a_{hj});$

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

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

Înmulţirea (rapidă) a matricilor

Considerăm următorul "produs de matrici"

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

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

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

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

În ipoteza (I') avem: $A^{(2^p)} = A^{(n-1)}, \forall p$ cu $2^p \geqslant n-1$.
Astfel, calculând succesiv, $A, A^{(2)}, A^{(4)} = A^{(2)} \otimes A^{(2)}, \ldots$,obţinem algoritm cu $\mathcal{O}(n^3 \log n)$ complexitate timp pentru rezolvarea problemei **P3**.

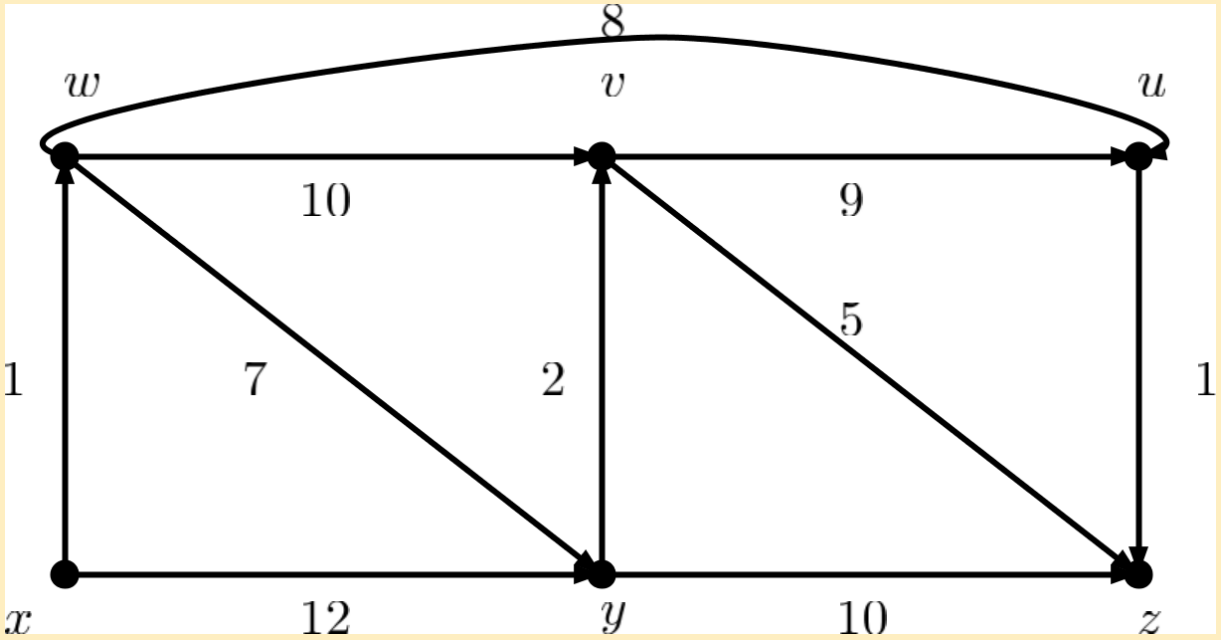
Drumuri de cost minim - Rezolvarea problemei **P3**

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

Exerciţii pentru seminarul de săptămâna următare

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

Exerciţiul 1'. Rulaţi algoritmul lui Dijkstra pe digraful următor.



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

Exerciţii pentru seminarul de săptămâna următare

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

Exerciţiul 1. Un graf G este dat funcţional: pentru fiecare nod $v \in V(G)$ putem obţine $N_G(v)$ pentru 1\$; alternativ, după o preprocesare care costă T\$ ($T >> 1$), putem obţine, $N_G(v)$ şi also $N_G(w)$, pentru toate nodurile $w \in V(G)$. În G se construieşte un drum P plecând dintr-un nod arbitrar şi alegând un vecin nevizitat încă atâta vreme cât este posibil. După ce drumul este construit putem să îi comparăm costul, $Online(P)$, cu cel mai mic cost posibil, $Offline(P)$. Descrieţi o strategie de plată (de accesare a mulţimilor de vecini) aşa încât

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

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

Exerciţii pentru seminarul de săptămâna următare

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

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

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

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

Exerciţiul 3. În problema determinării drumurilor de cost minim de la un nod dat s la toate celelalte noduri dintr-un digraf $G = (V, E)$, avem o funcţie de cost $c : E \to \{0, 1, \ldots, C\}$ unde $C \in \mathbb{N}$ nu depinde de $n = |V|$ sau de $m = |E|$. Cum se poate modifica algoritmul lui Dijkstra pentru a reduce complexitatea timp la $\mathcal{O}(n + m)$?

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

Exerciţii pentru seminarul de săptămâna următare

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

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

Fie

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

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

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

Exerciţiul 4. (cont'd)

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

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

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

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

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

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

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

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

Exerciţii pentru seminarul de săptămâna următare

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

Exerciţiul 7. Fie G un graf conex. Arătaţi că

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

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

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

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

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

Exerciţii pentru seminarul de săptămâna următare

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

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

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

Arătaţi că $m \leqslant n$;

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

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

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

(b) Demonstraţi că (a) este echivalentă cu

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

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

Cuprins

1. Conexiune
   - Teorema lui Menger
   - p-conexiune
   - Teorema lui König
   - Teorema lui Hall
   - Teorema lui Dirac
2. Arbori
   - Elemente de bază
   - Generarea tuturor arborilor parţiali
   - Numărarea arborilor parţiali: Teorema lui Kirchhoff
3. Exerciţii pentru seminarul de săptămâna viitoare

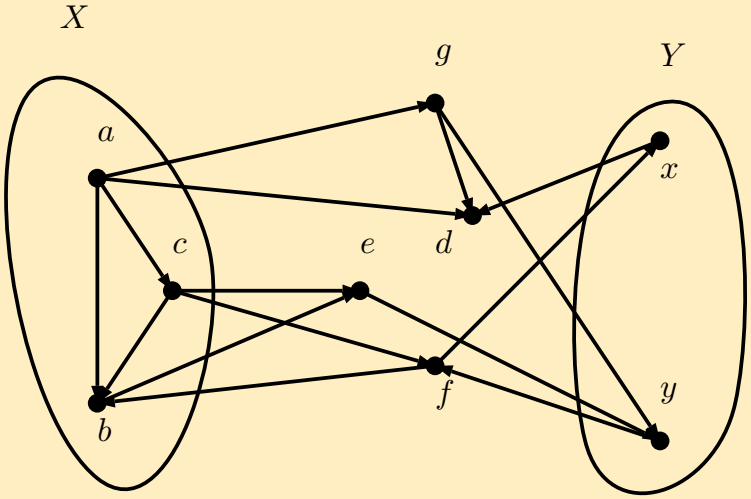
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Definiţia 1

Fie $G = (V, E)$ un (di)graf şi $X, Y \subseteq V$. Un **XY-drum** *este orice drum P din G de la un nod $x \in X$ la un nod $y \in Y$ astfel încât $V(P) \cap X = \{x\}$ şi $V(P) \cap Y = \{y\}$.*

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)

C. Croitoru - Graph Algorithms * C. Croitoru - 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 şi P_2 sunt **disjuncte** (pe noduri) dacă $V(P_1) \cap V(P_2) = \varnothing$.
- Motivată de problemele practice din reţelele de comunicaţii şi de asemeni de studiile teoretice asupra conexiunii în (di)grafuri, este de interes determinarea mulţimilor 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)** şi reprezintă unul dintre rezultatele fundamentale din Teoria grafurilor.

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

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

Definiţia 2

Fie $G = (V, E)$ un (di)graf şi $X, Y \subseteq V$. O **mulţime XY-separatoare** *în G este orice submulţime $Z \subseteq V$ astfel încât*

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

Notăm cu

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

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

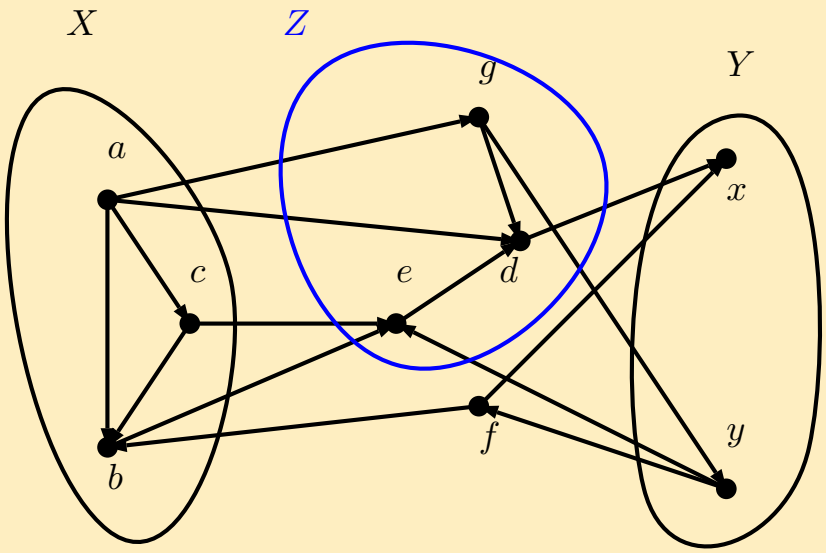
Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Din definiţie urmează că:

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

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

Exemplu



A XY-separating set: $Z = \{g, e, d\}$

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

- Dacă $Z \in \mathbf{S}(X, Y; G)$, atunci $A \in \mathbf{S}(X, Y; G)$, $\forall A$ astfel încât $Z \subseteq A \subseteq V$.
- Dacă $Z \in \mathbf{S}(X, Y; G)$ şi $T \in \mathbf{S}(Z, Y; G)$, atunci $T \in \mathbf{S}(X, Y; G)$.

Teorema 1

Teorema lui Menger. Fie $G = (V, E)$ un (di)graf şi $X, Y \subseteq V$. Atunci $p(X, Y; G) = k(X, Y; G)$.
(I. e., numărul maxim de XY-drumuri disjuncte = cardinalul minim al unei mulţimi XY-separatoare.)

Demonstraţie:
$k(X, Y; G) \geqslant p(X, Y; G) = r$. Fie $P_1, \ldots, P_r$ XY-drumuri disjuncte din G; $Z \cap V(P_i) \neq \varnothing$, $\forall Z \in \mathbf{S}(X, Y; G)$. Deoarece P_i sunt disjuncte ($i = \overline{1, r}$):

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

Astfel, $|Z| \geqslant r$, $\forall Z \in \mathbf{S}(X, Y; G)$; urmează că $k(X, Y; G) \geqslant r$.

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

$k(X, Y; G) \leqslant p(X, Y; G)$. Omisă. (Vom arăta mai târziu că $\forall G = (V, E)$ şi $\forall X, Y \subseteq V$, $\exists k(X, Y; G)$ XY-drumuri disjuncte în G folosind fluxuri în anumite reţele.) $\square$

Menger (1927) a enunţat 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

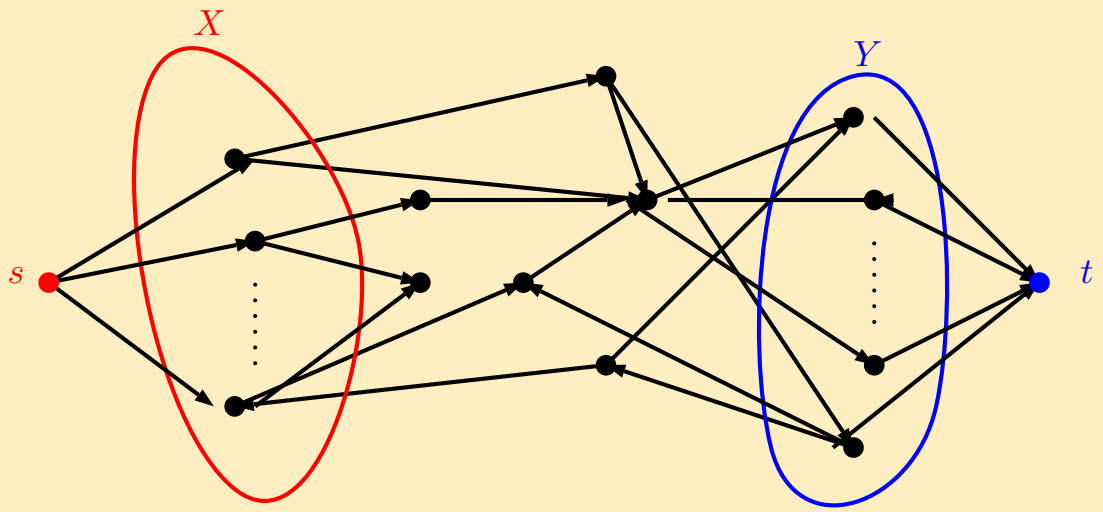
Fie $G = (V, E)$ un (di)graf şi $s, t \in V$, astfel încât $s \neq t$, $st \notin E$. Există k drumuri intern-disjuncte de la s la t în G dacă şi numai dacă există cel puţin un drum de la s la t în (di)graful obţinut din G prin ştergerea oricărei mulţime de $< k$ noduri diferite de s şi t.

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

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

Demonstraţia echivalenţei:

Teorema 1 $\Rightarrow$ Teorema 2: luăm $X = N_G^+(s)$ $(N_G(s))$ şi $Y = N_G^-(t)$ $(N_G(t))$.



Teorema 2 $\Rightarrow$ Teorema 1: adăugăm două noi noduri s şi t (di)grafului G, şi toate muchiile (orientate) de la s la orice nod din X şi de la orice nod din Y la t. $\square$

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

Aplicaţii: p-conexiune

- Un graf G este p-conex ($p \in \mathbb{N}^*$) dacă fie $G = K_p$, fie $|G| > p$ şi $G \setminus A$ este conex pentru orice $A \subseteq V(G)$ cu $|A| < p$.
- Din Teorema 2, o caracterizare echivalentă a p-conexiunii este: Un graf G este p-conex ($p \in \mathbb{N}^*$) dacă fie $G = K_p$, fie $\forall st \in E(\overline{G})$ există p drumuri intern-disjuncte de la s la t în G.
- **numărul de conexiune pe noduri** al grafului G, $k(G)$, este cel mai mare p pentru care G este p-conex.
- Urmează că, pentru a calcula $k(G)$, trebuie aflat

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

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

Aplicaţii: Teorema lui König

- O **acoperire cu noduri a grafului G** este o mulţime $X \subseteq V(G)$ de noduri astfel încât $G - X$ este un graf nul (orice muchie din G are cel puţin o extremitate în X).
- Un caz special al Teoremei 1 se obţine când G este bipartit şi X, Y sunt cele două clase ale bipartiţiei 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.

Demonstraţie: Cardinalul maxim al unui cuplaj în G este $p(S, T; G) = k(S, T; G)$, din Teorema 1. Deoarece o mulţime de noduri este o mulţime ST-separatoare dacă şi numai dacă este o acoperire cu noduri, Teorema 3 este dovedită. $\square$

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

Aplicaţii: Teorema lui Hall

- Fie I şi S mulţimi finite nevide. O familie submulţimi ale lui S (indexată după I) este o funcţie $A : I \to 2^S$. Notăm $\mathcal{A} = (A_i)_{i \in I}$ şi (folosind notaţia funcţională) $\mathcal{A}(J) = \bigcup_{j \in J} A_j$ (pentru $J \subseteq I$).
- O funcţie de reprezentare pentru familia $\mathcal{A} = (A_i)_{i \in I}$ este orice funcţie $r_{\mathcal{A}} : I \to S$ 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 reprezentanţi pentru $\mathcal{A}$.
- Dacă funcţia de reprezentare, $r_{\mathcal{A}}$, este injectivă, atunci $r_{\mathcal{A}}(I)$ este o submulţime a lui S şi este numită **sistem de reprezentanţi distincţi** pentru $\mathcal{A}$, sau o **transversală** a lui $\mathcal{A}$.
- Problema centrală în Teoria Transversalelor este de a caracteriza familiile care admit o transversală (cu anumite proprietăţi). Teorema lui **Hall (1935)** este primul rezultat de acest tip.

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

Teorema 4

Hall, 1935 *Familia $\mathcal{A} = (A_i)_{i \in I}$ de submulţimi ale lui S are o transversală dacă şi numai dacă*

$$\text{(H)} \qquad |\mathcal{A}(J)| \geqslant |J|, \forall J \subseteq I.$$

Demonstraţie: "$\Rightarrow$" Dacă $r_{\mathcal{A}}$ este o funcţie 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, $|\mathcal{A}(J)| \geqslant |r_{\mathcal{A}}(J)| \geqslant |J|$.
"$\Leftarrow$" Fie $G_{\mathcal{A}} = (I, S; E)$ graful bipartit asociat familiei $\mathcal{A}$ (dacă $I \cap S \neq \varnothing$, 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 şi numai dacă $G_{\mathcal{A}}$ are un cuplaj de cardinal $|I|$.

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

Demonstraţia Teoremei lui Hall (continuare): Arătăm că dacă relaţia (H) are loc, atunci orice acoperire cu noduri a lui $G_{\mathcal{A}}$ are cel puţin $|I|$ noduri, şi - din Teorema lui Konig - $G_{\mathcal{A}}$ are un cuplaj de cardinal $|I|$.

Fie $X = I' \cup S' \subseteq I \cup S$ o acoperire cu noduri a lui $G_{\mathcal{A}}$: urmează că $N_{G_{\mathcal{A}}}(I \setminus I') \subseteq S'$, adică, $\mathcal{A}(I \setminus I') \subseteq S'$. Atunci,

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

Deoarece are loc (H), obţinem

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

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

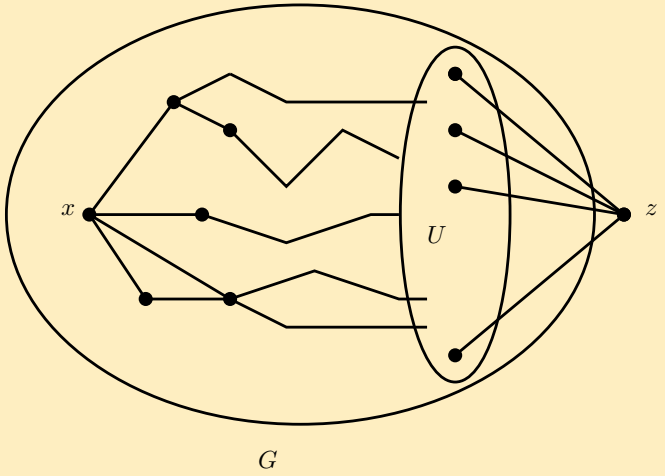
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

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

Lemă

Fie $G = (V, E)$ un graf p-conex de ordin $|G| \geqslant p + 1$, $U \subseteq V$, $|U| = p$ şi $x \in V \setminus U$. Atunci există p xU-drumuri astfel încât oricare două dintre ele îl au numai pe x drept nod comun.

Demonstraţie: Fie $G' = (V \cup \{z\}, E')$, unde $E' = E \cup \{zu : u \in U\}$.



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

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

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

Demonstraţie (continuare). Atunci, G' este un graf p-conex. Într-adevăr, fie $A \subseteq V(G')$ cu $|A| \leqslant p-1$. Dacă $A \subseteq V(G)$, atunci $G'-A$ este conex (din k-conexiunea lui G, $G-A$ este conex; cum $|A| < p$, $\exists u \in U \setminus A$ şi, 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' şi perechii x, z. $\square$

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

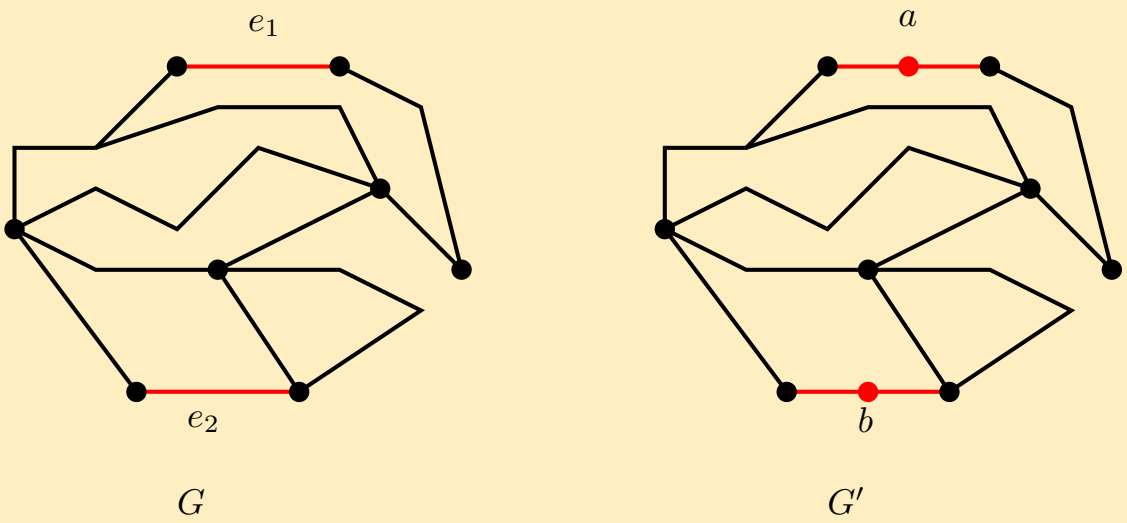
Propoziţie

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

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

Demonstraţie: Inducţie după p.

Pentru $p = 2$, trebuie să arătăm că într-un graf 2-conex, G, orice două muchii e_1 şi e_2 aparţin unui circuit. Fie G' graful obţinut din G prin inserarea unui nod a pe e_1 şi a unui nod b pe e_2:



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

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

Demonstraţie (continuare). În pasul inductiv, fie $p \geqslant 3$, presupunem că Propoziţia este adevărată pentru orice graf p'-conex cu $2 \leqslant p' < p$, şi considerăm un graf p-conex G, două dintre muchiile, sale e_1 şi e_2 şi o mulţime de $p-2$ noduri $\{x_1, x_2, \ldots, x_{p-2}\}$.

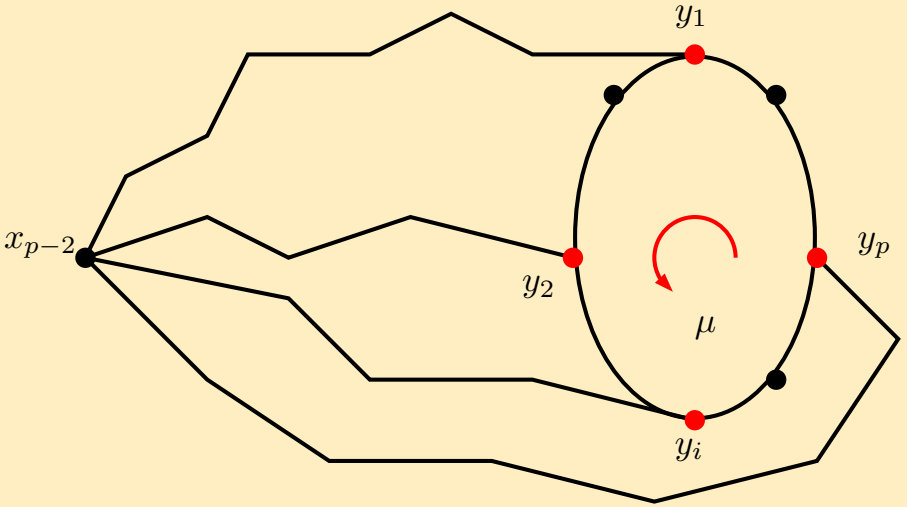
Putem presupune că nicio extremitate v a lui e_1 sau e_2 nu aparţine mulţimii $\{x_1, x_2, \ldots, x_{p-2}\}$ (altfel, aplicăm ipoteza inductivă şi obţinem că în graful $(p-1)$-conex, G, există un circuit C conţinând e_1, e_2 şi mulţimea de noduri $\{x_1, x_2, \ldots, x_{p-2}\} \setminus \{v\}$; iar v este un nod al lui C deoarece e_1 şi e_2 sunt muchii ale lui C).

Graful $G - x_{p-2}$ este $(p-1)$-conex. Din ipoteza inductivă, există un circuit μ care conţine $x_1, x_2, \ldots, x_{p-3}$, e_1 şi e_2. Fie Y mulţimea nodurilor lui μ. Evident, $|Y| \geqslant p$ (mulţimii de $p-3$ noduri $x_1, x_2, \ldots, x_{p-3}$, îi adăugăm cel puţin trei extremităţi ale muchiilor e_1 şi 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}.

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

Demonstraţie (continuare). Fie $P_{x_{p-2}y_1}, P_{x_{p-2}y_2}, \ldots P_{x_{p-2}y_p}$ aceste drumuri, unde ordinea $y_1, \ldots, y_p$ se obţine în urma unei parcurgeri a lui μ.

Nodurile $y_1, \ldots, y_p$ împart circuitul μ în drumurile $P_{y_1y_2}$, $P_{y_2y_3}, \ldots, P_{y_{p-1}y_p}$, $P_{y_py_1}$:



Cel puţin unul dintre drumurile de mai sus nu conţine în interior niciun element din mulţimea $x_1, x_2, \ldots, x_{p-3}$, e_1 şi e_2 (pigeon hole principle).

Fie $P_{y_1y_2}$ acest drum (altfel, renumerotăm nodurile y_i).

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

Demonstraţie (continuare). Atunci,

$$P_{x_{p-2}y_2}, P_{y_2y_3}, ..., P_{y_py_1}, P_{y_1x_{p-2}}$$

este circuitul din G care conţine $x_1, x_2, \ldots, x_{p-2}$, e_1 şi e_2. $\square$

Teorema 5

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

Demonstraţie: Fie $G = (V, E)$ un graf p-conex, $p \geqslant 2$. Fie $x_1, x_2, \ldots, x_p$ p noduri ale lui G. Deoarece G este conex, există muchiile $e_1 = xx_{p-1}$ şi $e_2 = yx_p$. Atunci, teorema urmează din Propoziţia 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

O aplicaţie interesantă a acestei teoreme (şi a demonstraţiei propoziţiei) este următoarea condiţie suficientă pentru ca un graf să fie Hamiltonian dată de Erdös şi Chvatal.

Teorema 6

(Erdös-Chvatal, 1972) *Fie $G = (V, E)$ un graf p-conex. Dacă $\alpha(G) \leqslant p$ atunci G este graf Hamiltonian.*

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

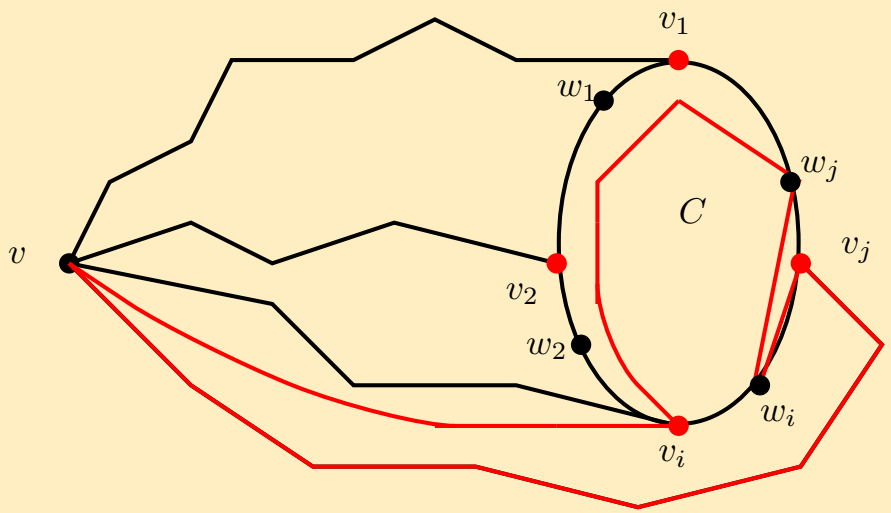
Demonstraţie: Să presupunem, prin contradicţie că G nu este Hamiltonian. Fie C un cel mai lung circuit din G.
Din Teorema lui Dirac $|C| \geqslant p$ şi din presupunerea noastră, există un nod $v \in V(G) \setminus V(C) \neq \varnothing$.

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

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

Demonstraţie (continuare). Cum $|C| \geqslant p$, putem repeta argumentul din demonstraţia Propoziţiei de mai sus pentru a arăta că există P_{vv_1}, $P_{vv_2}, \ldots, P_{vv_p}$, p vC-drumuri care se intersectează două câte două doar în v şi cu extremităţi v_i etichetate în ordinea în care apar la o parcurgere a circuitului.

Fie w_i succesorul nodului v_i pe circuit.



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

Demonstraţie (continuare). Observăm că $vw_i \notin E$ (altfel, circuitul $vw_i, w_i, C \setminus \{w_i v_i\}, P_{v_i v}$ este mai lung decât C, contradicţie).
Deoarece $\alpha(G) \leqslant p$, mulţimea $\{v, w_1, w_2, \ldots, w_p\}$ nu este stabilă, şi 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, şi drumul $P_{v_j v}$ oferă un circuit mai lung decât C, contradicţie). $\square$

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

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

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 afirmaţii sunt echivalente:

(i) *G este un arbore (este conex şi nu are circuite).*

(ii) *G este conex şi este minimal cu această proprietate.*

(iii) *G **nu are circuite** şi este maximal cu această proprietate.*

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

Demonstraţie: Omisă. □

Minimalitatea şi maximalitatea din afirmaţiile de mai sus sunt relativ la relaţia de ordine parţială dată de incluziune pe submulţimile muchii. Mai precis afirmaţile (ii) şi (iii) înseamnă:

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

(ii) G este conex şi $\forall e \in E$, $G - e$ nu este conex.

(iii) G nu are circuite şi $\forall e \notin E$, $G + e$ are un circuit.

Definiţie

Fie $G = (V, E)$ un (multi)graf. Un **arbore parţial** *G este un graf parţial al lui G, $T = (V, E')$ ($E' \subseteq E$), care este arbore. Notăm cu $\mathcal{T}_G$ mulţimea tuturor arborilor parţiali ai lui G.*

Remarci

1. $\mathcal{T}_G \neq \varnothing$ dacă şi numai dacă G este conex. Într-adevăr, dacă $\mathcal{T}_G \neq \varnothing$, atunci există un arbore parţial $T = (V, E')$ al lui G. T este conex, deci între orice două noduri ale lui G there este un drum P în T. Deoarece $E' \subseteq E$, P este un drum şi în G, deci G este conex.

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

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 construcţie, T este graf parţial al lui G, şi are loc afirmaţia (ii) din Teorema 7, deci T este un arbore.

2. O altă demonstraţie constructivă (dacă G este conex atunci $\mathcal{T}_G \neq \varnothing$) se bazează pe observaţia că există o muchie în cross între cele două clase ale oricărei bipartiţii a lui V: $\exists e = v_1 v_2 \in E$ cu $v_i \in V_i$, $i = \overline{1,2}$.
Dacă $|V| = n > 0$ atunci următorul algoritm construieşte un arbore parţial 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 * C. Croitoru - Graph Algorithms *

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

$k \leftarrow 1$; $T_1 \leftarrow (\{v\}, \varnothing)$; // $v \in V$
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 construcţie, T_{k+1} este conex şi nu are circuite). Mai mult, avem $|V(T_k)| = k$ şi $|E(T_k)| = k - 1$, $\forall k = \overline{1, n}$.

3. Dacă această construcţie este aplicată unui arbore G cu n noduri, vom obţine 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 afirmaţii sunt echivalente pentru un graf $G = (V, E)$ cu n noduri:

(i) *G este un arbore.*

(ii) *G este conex şi are $n - 1$ muchii.*

(iii) *G nu are circuite şi are $n - 1$ muchii.*

(iv) *$G = K_n$ pentru $n \in \{1, 2\}$, iar pentru $n \geqslant 3$ $G \neq K_n$ şi $G + e$ are exact un circuit, pentru orice muchie $e \in E$.*

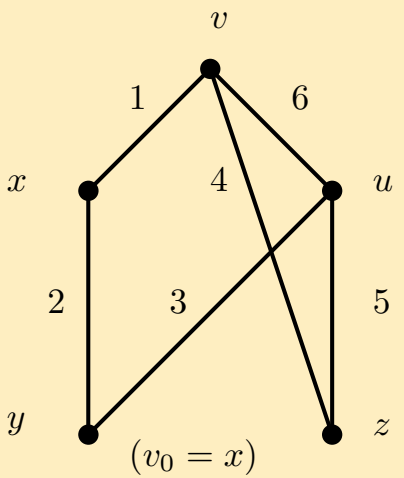
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie: Omisă. □

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

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

- Descriem o metodă simplă de tip **backtracking** pentru a genera toţi arborii parţiali ai unui graf conex $G = (V, E)$, unde $V = \{1, \ldots, n\}$, $|E| = m$.
- Mulţimea de muchii, E, va fi reprezentată cu un tablou $E[1..2, 1..m]$ cu elemente din V, cu semnificaţia: dacă $v = E[1, i]$ şi $w = E[2, i]$, 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 ($E[1, i] = v_0$, $\forall i = \overline{1, d_G(v_0)}$), pentru $v_0 \in V$.



1	2	3	4	5	6
x	x	y	z	z	u
v	y	u	v	u	v

Arbori - 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 parţial $T \in \mathcal{T}_G$ va fi reprezentat ca o mulţime de $n-1$ indecşi (în ordine crescătoare) ai coloanelor din tabloul E (desemnându-i muchiile).
- Întimpul generării, menţinem un vector $T[1..n-1]$ cu elemente din $\{1,\ldots,m\}$ şi o variabilă flag $i \in \{1,\ldots,n\}$ cu următoarele semnificaţii:

  Căutăm toţi arborii parţiali ai lui G, cu proprietatea că cele mai mici $i-1$ muchii sunt: $T[1] < T[2] < \ldots < T[i-1]$.
- Pentru exemplul de mai sus, dacă $i = 2$, $T[1] = 1$, şi $T[2] = 2$, atunci arborii care vor fi găsiţi sunt $\{1,2,3\}$, $\{1,2,5\}$, şi $\{1,2,6\}$. Dacă, $i = 2$, $T[1] = 3$, şi $T[2] = 5$ atunci arborele care trebuie găsit este $\{3,5,6\}$. Dar dacă $i = 2$, $T[1] = 1$, şi $T[2] = 6$, niciun arbore nu va fi găsit.

Arbori - 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

ALL-ST-Gen(i)

// sunt generaţi toţi arborii parţiali G , cu cele mai mici $i-1$ muchii: $T[1], \ldots, T[i-1]$

if $(i = n)$ **then**

// $\{T[1], \ldots, T[n-1]\}$ este arbore parţial

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 $(\langle \{T[1], \ldots, T[i-1]\} \cup \{j\} \rangle_G$ 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 * C. Croitoru - Graph Algorithms *

Arbori - Generarea $\mathcal{T}_G$

- Prin apelul All-ST-Gen(1) obţinem $\mathcal{T}_G$.
- Pentru a testa dacă graful $\langle\{T[1], \ldots, T[i-1]\} \cup \{j\}\rangle_G$ nu are circuite, observăm că, din construcţie,

$$\langle\{T[1], \ldots, T[i-1]\}\rangle_G$$

  nu are circuite, deci este o pădure (fiecare componentă conexă este un arbore).
- Fie $root[1..n]$ un vector (global) cu elemente din V şi semnificaţia: $root[v]$ = rădăcina componentei conexe care conţine v (unul dintre nodurile sale).
- Înaintea apelului All-ST-Gen(1), vectorul *root* este iniţializat pentru a satisface proprietatea: $root[v] \leftarrow v$ $(\forall v \in V)$ (deoarece atunci, $\{T[1], \ldots, T[i-1]\} = \varnothing$).

Arbori - Generarea $\mathcal{T}_G$

- În timpul apelurilor recursive, când se testează dacă muchia j poate fi adăugată mulţimii $\{T[1], \ldots, T[i-1]\}$ fără a crea vreun circuit, fie $v = E[1, j]$ şi $w = E[2, j]$. Atunci, $\langle \{T[1], \ldots, T[i-1]\} \cup \{j\} \rangle_G$ nu are circuite dacă şi numai dacă v şi w sunt în componente conexe diferite ale pădurii, i.e., $root[v] \neq root[w]$.
- Pentru a actualiza vectorul *root*, în locurile marcate cu **A** şi **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**
    $S \leftarrow S \cup \{u\}$; $root[u] \leftarrow root[w]$;

Arbori - 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 Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 $root[w]$; aceste noduri sunt salvate în mulţimea S.
- După apelul All-ST-Gen$(i+1)$, vector *root* trebuie setat din nou la valoarea dinaintea apelului, aceasta poate fi făcută B prin:

  for $(u \in S)$ **do**
  $\quad root[u] = x;$

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

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

Fie $G = (V, E)$ un multi-graf cu $V = \{1, 2, \ldots, n\}$, şi matricea de adiacenţă $A = (a_{ij})_{n \times n}$ (a_{ij} = multiplicitatea muchiei ij dacă $ij \in E$, 0 altfel). Fie

$$\mathbf{D = diag(d_G(1), d_G(2), \ldots, d_G(n))} = \begin{pmatrix} d_G(1) & 0 & \ldots & 0 \\ 0 & d_G(2) & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & d_G(n) \end{pmatrix}.$$

Matricea Laplaciană a lui G (sau **Laplacianul**) este definită ca fiind:

$$\mathbf{L[G] = D - A.}$$

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 $L[G]$ obţinut prin ştergerea liniei i şi a coloanei j.

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

Teorema 9

(Kirchoff-Trent). *Fie G un (multi)graf cu mulţimea nodurilor $\{1, \ldots, n\}$ şi Laplacianul $L[G]$. Atunci, numărul arborilor parţiali ai lui G este:* $|\mathcal{T}_G| = det(L[G]_{ii}), \forall 1 \leqslant i \leqslant n$.

Demonstraţie: *Omisă.* $\square$

Corolar

(Formula lui Cayley). $|\mathcal{T}_{K_n}| = n^{n-2}$.

Demonstraţie:

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

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 obţinem

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

□

Exerciţii pentru seminarul de săptămâna viitoare

Exerciţiul 1. Fie $G = (V, E)$ un graf conex şi $v \in V$ 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, mulţimea $A = \{v\}$ satisface următoarele proprietăţi:

(i) $v \in A$ şi $[A]_G$ este conex.

(ii) $N = N_G(A) \neq \varnothing$.

(iii) $R = V \setminus (A \cup N) \neq \varnothing$.

(a) Arătaţi că, dacă $A \subseteq V$ este orice mulţime noduri care satisface (i) - (iii) şi maximală (relativ la "$\subseteq$") cu aceste proprietăţi, atunci $\forall x \in R$ şi $\forall y \in N$ avem $xy \in E$.

(b) Dovediţi că dacă A este ca la (a) şi G este $\{C_k\}_{k \geqslant 4}$-free, atunci N este o clică în G.

(c) Deduceţi că K_n ($n \in \mathbb{N}^*$) sunt singurele grafuri regulate, triangulate şi conexe.

Exerciţii pentru seminarul de săptămâna viitoare

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

Exerciţiul 2. Un graf de ordin cel puţin trei este numit confidenţial conex dacă, pentru orice trei noduri distincte a, b şi c, există un drum de la a la b astfel încât c este diferit de şi nu este adiacent cu niciun nod intern (dacă există) al acestui drum. (Un exemplu de graf confidenţial conex este graful complet K_n, cu $n \geqslant 3$.)
Arătaţi că un graf conex, necomplet, $G = (V, E)$, cu cel puţin trei noduri este confidenţial conex dacă şi numai dacă:

(i) pentru orice $v \in V$, $N_{\overline{G}}(v) \neq \varnothing$ şi induce un subgraf conex;

(ii) orice muchie a lui G face parte dintr-un C_4 indus sau este muchia mediană a unui P_4 indus.

Exerciţiul 3. Dovediţi că un graf conex, p-regulat şi bipartit este 2-conex.

Exerciţii pentru seminarul de săptămâna viitoare

Exerciţiul 4. Fie $G = (V, E)$ un digraf. Demonstraţi că:

(a) G este tare conex dacă şi numai dacă pentru orice $S \subsetneq V$, $S \neq \varnothing$, există măcar un arc care pleacă din S.

(b) Dacă G este tare conex şi poate fi deconectat prin ştergerea a cel mult p arce (i. e., $\exists A \subseteq E$, $|A| \leqslant p$ astfel încât $G - A$ nu este tare conex), atunci G poate fi deconectat prin inversarea a cel mult p arce (adică $\exists B \subseteq E$, $|B| \leqslant p$ astfel încât $G' = (V, (E \setminus B) \cup \{uv : vu \in B\})$ nu este tare conex).

Exerciţiul 5. Fie G un graf 2-muchie-conex ($G - e$ este conex, $\forall e \in E(G)$). Definim următoarea relaţie binară $e \asymp f$ dacă $e = f$ or $G - \{e, f\}$ nu este conex.

(a) Arătaţi că $e \asymp f$ dacă şi numai dacă e şi f aparţin aceloraşi circuite.

(b) Arătaţi că o clasă de echivalenţă $[e]_{\asymp}$ este inclusă într-un circuit.

(c) Ştergând toate muchiile dintr-o clasă de echivalenţă $[e]_{\asymp}$, componentele conexe ale grafului rămas sunt grafuri 2-muchie-conexe.

Exerciţii pentru seminarul de săptămâna viitoare

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

Exerciţiul 6.

(a) Fie G un graf cu cel puţin 3 noduri. Dacă G este 2-conex, atunci putem să-i orientăm muchiile aşa încât graful orientat rezultat să fie tare conex.

(b) Reciproca afirmaţiei de mai sus este adevărată?

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

Exerciţiul 7.

(a) Fie G un graf 2-conex, necomplet şi $xy \in E(G)$. Arătaţi că $G - xy$ sau $G|xy$ este 2-conex.

(b) Daţi câte un exemplu de un graf G şi o muchie $xy \in E(G)$ astfel ca: (b1) $G - xy$ şi $G|xy$ 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;

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

Exerciţiul 8. Pentru un graf conex G dat aplicăm următorul algoritm:

$\mathcal{Q} \leftarrow \{G\}$; // $\mathcal{Q}$ este o coadă;
while $(\mathcal{Q} \neq \varnothing)$ **}**
  $H \leftarrow$ **pop**$(\mathcal{Q})$;
  fie $A \subseteq V(H)$ **o mulţime de articulaţie minimală din** H;
  fie $G_1, \ldots, G_k$ **componentele conexe ale lui** $H - A$;
  pentru $(j = 1$ **to** $k)$
    push$(\mathcal{Q}, [A \cup V(G_j)]_G)$;
  }

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

a) Arătaţi că orice graf adăugat în $\mathcal{Q}$ este conex.

b) Dovediţi că numărul total de grafuri adăugate la coada $\mathcal{Q}$ este cel mult $|G|^2$.

Exerciţii 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

Exerciţiul 9. Fie $G = (V, E)$ un graf conex şi T_1, T_2 doi arbori parţiali ai lui G ($T_1, T_2 \in \mathcal{T}_G$).

(a) Dovediţi că T_1 poate fi transformat în T_2 prin aplicarea repetată a următoarei proceduri: şterge o muchie şi adaugă o altă muchie arborelui curent.

(b) Dacă, în plus, G este 2-conex arătaţi că T_1 poate fi transformat în T_2 prin aplicarea repetată a următoarei proceduri: şterge o muchie uv şi adaugă o altă muchie uw arborelui curent.

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

Exerciţiul 10. Demonstraţi că mulţimea de muchii a unui graf complet K_n ($n \geqslant 2$) poate fi partiţionată în $\lceil n/2 \rceil$ submulţimi fiecare reprezentând mulţimea de muchii ale unui arbore (subgraf al lui K_n).

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

Exerciţii 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 -

Exerciţiul 11. Fie n un întreg pozitiv şi $G_n = (V, E)$ un graf definit astfel:

- $V = \{(i, j) : 1 \leqslant i, j \leqslant n\}$;
- $(i, j)(k, l) \in E$ (pentru $(i, j) \neq (k, l)$ din V) dacă şi numai dacă $i = l$ sau $j = k$.

Arătaţi că G_n este universal pentru familia arborilor de ordin n: pentru orice arbore T de ordin n, $\exists A \subseteq V$ astfel încât $T \cong [A]_{G_n}$.

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

Exercise 12. Arătaţi că un turneu este tare conex dacă şi numai dacă are un circuit hamiltonian.

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

Exerciţii 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

Exerciţiul 13. Se consideră reţeaua stradală a unui oraş. Se ştie că putem îndepărta posibilitatea de a merge în cerc în oraş pe această reţea prin blocarea a cel mult p sensuri de străzi (prin blocarea unui singur sens se înţelege obstrucţionarea sensului vizat de pe stradă). Arătaţi că putem îndepărta posibilitatea de a merge în cerc în oraş pe reţeaua sa stradală prin inversarea a cel mult p sensuri de străzi.
(Inversarea unui sens pe o stradă cu două sensuri implică păstrarea doar a celuilalt sens; inversarea sensului unei străzi cu un singur sens implică introducerea celuilalt sens.)

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

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

Cuprins

1. Problema arborelui parţial de cost minim
   - Metoda generală MST
   - Algoritmul lui Prim
   - Algoritmul lui Kruskal
2. Cuplaje
   - Cuplaje maxime – Acoperire minimă cu muchii
3. Exerciţii pentru seminarul din săptămâna următoare

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

Problema arborelui parţial 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 * C. Croitoru - Graph Algorithms *

Problema MST. Dat $G = (V, E)$ un graf şi $c : E \to \mathbb{R}$ ($c(e)$ este costul muchiei e) găsiţi $T^* \in \mathcal{T}_G$ astfel încât

$$c(T^*) = \min_{T \in \mathcal{T}_G} c(T),$$

unde $c(T) = \sum_{e \in E(T)} c(e)$.

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

Metoda generală MST

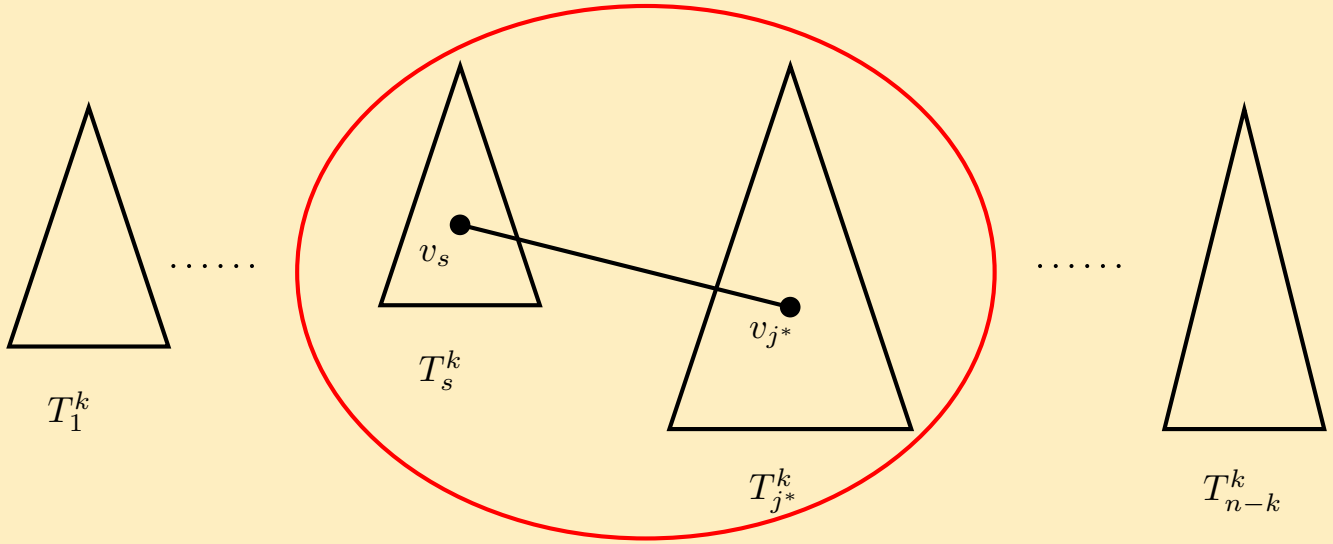
- Se porneşte cu familia $\mathcal{T}^0 = (T_1^0, T_2^0, \ldots, T_n^0)$ de arbori disjuncţi: $T_i^0 = (\{i\}, \varnothing)$, $i = \overline{1, n}$.
- La fiecare pas k $(0 \leqslant k \leqslant n-2)$, din familia $\mathcal{T}^k = (T_1^k, T_2^k, \ldots, T_{n-k}^k)$ de $n-k$ arbori disjuncţi astfel încât $V = \bigcup_{i=1}^{n-k} V(T_i^k)$ şi $\bigcup_{i=1}^{n-k} E(T_i^k) \subseteq E$, construieşte $\mathcal{T}^{k+1}$ după cum urmează:

  - alege $T_s^k \in \mathcal{T}^k$;
  - determină o muchie de cost minim $e^* = v_s v_{j^*}$ din mulţimea de muchii ale lui G cu o extremitate $v_s \in V(T_s^k)$ şi 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 obţinut din T_s^k şi $T_{j^*}^k$ prin adăugarea muchiei e^*.

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 $V(T_s^k)$ şi cealaltă în $V \setminus V(T_s^k)$, atunci G nu este conex şi nu există vreun MST în G.
- Construcţia de mai sus este sugerată în imaginea de mai jos:



- Familia $\mathcal{T}^{n-1}$ are doar un arbore, T_1^{n-1}.

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

Teorema 1

Dacă $G = (V, E)$ este un graf conex cu $V = \{1, 2, \ldots, n\}$, atunci T_1^{n-1} construit de algoritmul anterior este un MST al lui G.

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

Demonstraţie: Demonstrăm (prin inducţie) că (*) $\forall k \in \{0, \ldots, n-1\}$ există un arbore parţial 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 $T_1^{n-1} = T_{n-1}^*$ şi teorema este demonstrată.
Pentru $k = 0$, avem $E(\mathcal{T}^0) = \varnothing$ şi, deoarece G este conex, există un MST T_0^*; astfel, proprietatea (*) este adevărată.

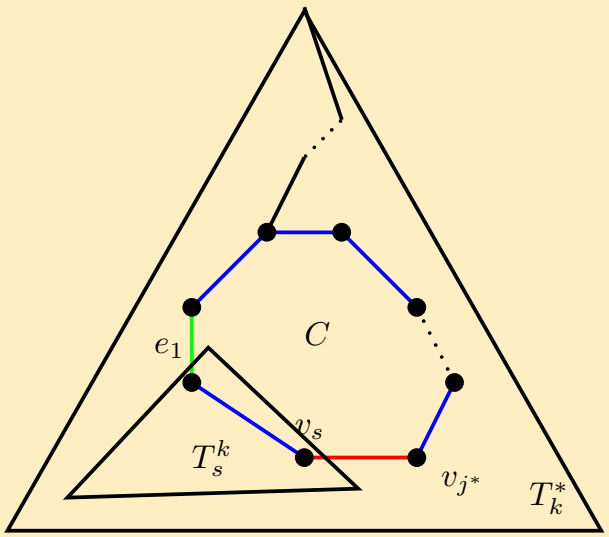
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru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

Demonstraţie (continuare). Dacă proprietatea (*) are loc pentru $0 \leqslant k \leqslant n-2$, atunci există un MST al lui G, T_k^*, astfel încât $E(\mathcal{T}^k) \subseteq E(T_k^*)$. Din construcţie, $E(\mathcal{T}^{k+1}) = E(\mathcal{T}^k) \cup \{e^*\}$. Dacă $e^* \in E(T_k^*)$, atunci luăm $T_{k+1}^* = T_k^*$ şi proprietatea are loc şi pentru $k+1$.
Să presupunem că $e^* \notin E(T_k^*)$. Atunci, $T_k^* + e^*$ are exact un circuit C, conţinând $e^* = v_s v_{j^*}$. Deoarece $v_{j^*} \notin V(T_s^k)$, urmează că există o muchie $e_1 \neq e^*$ in C cu o extremitate în $V(T_s^k)$ şi cealaltă în $V \setminus V(T_s^k)$. Din modul de alegere al e^* avem $c(e^*) \leqslant c(e_1)$ şi $e_1 \in E(T_k^*) \setminus E(\mathcal{T}^k)$. Fie $T^1 = T_k^* + e^* - e_1$; evident, $T^1 \in \mathcal{T}_G$ (fiind conex cu $n-1$ muchii). Deoarece $e_1 \in E(T_k^*) \setminus E(\mathcal{T}^k)$, avem $E(\mathcal{T}^{k+1}) \subseteq E(T^1)$.

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

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

Demonstraţie (continuare).



Pe de altă parte, deoarece $c(e^*) \leqslant c(e_1)$, avem $c(T^1) = c(T_k^*) + c(e^*) - c(e_1) \leqslant c(T_k^*)$.
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 conţinând toate muchiile din $E(\mathcal{T}^{k+1})$. Luând $T_{k+1}^* = T^1$ încheiem demonstraţia teoremei. $\square$

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

Remarci

- Demonstraţia de mai sus rămâne adevărată şi pentru funcţii de cost $c : \mathcal{T}_G \to \mathbb{R}$ care satisfac: $\forall T \in \mathcal{T}_G, \forall e \in E(T), \forall e' \notin E(T)$

$$c(e') \leqslant c(e) \Rightarrow c(T + e' - e) \leqslant c(T).$$

- 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 $\mathcal{T}^k$, legaţi printr-o muchie de cost minim printre toate muchiile cu extremităţi în arbori diferiţi ai familiei.

Algoritmul lui Prim

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

- În strategia lui Prim T_s^k **este arborele de ordin maxim din familia** $\mathcal{T}^k$.
- Urmează că la fiecare pas $k > 0$ al metodei generale, $\mathcal{T}^k$ are un arbore, $T_s^k = (V_s, E_s)$, cu $k+1$ noduri şi $n-k-1$ arbori fiecare cu câte un nod.
- **Implementarea lui Dijkstra:** Fie α şi β doi vectori de dimensiune n; elementele lui α sunt noduri din $V(G)$ iar elementele lui β sunt numere reale, cu următoarea semnificaţie:

(S) $$\forall j \in V \setminus V_s, \beta[j] = c(\alpha[j]j) = \min_{i \in V_s, ij \in E} c(ij)$$

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

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

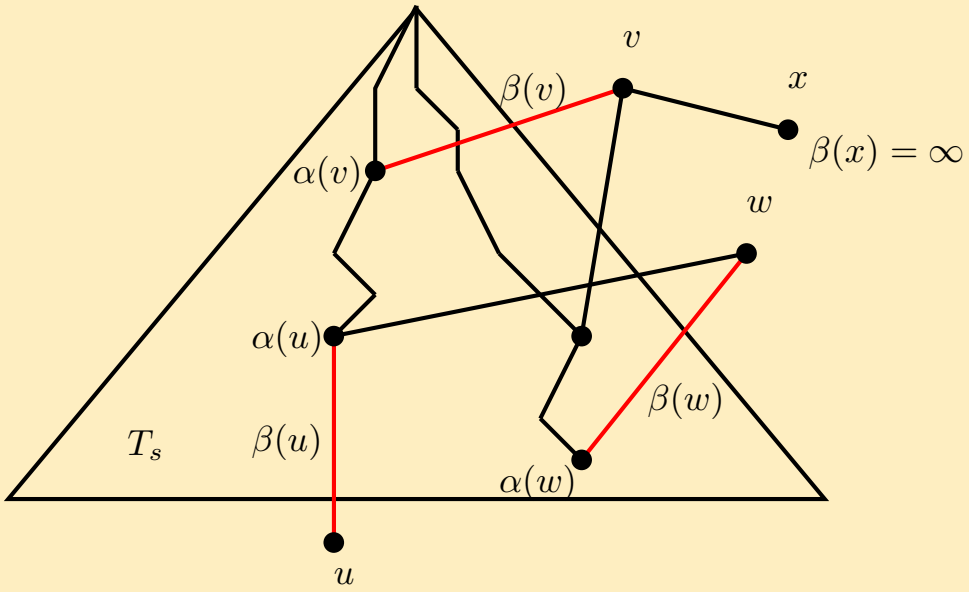
Algoritmul lui Prim

$V_s \leftarrow \{s\}$; $E_s \leftarrow \varnothing$; // pentru un $s \in V$.
for $(v \in V \setminus \{s\})$ **do**
  $\alpha[v] \leftarrow s$; $\beta[v] \leftarrow c(sv)$; // dacă $ij \notin E$, atunci $c(ij) = \infty$.
while $(V_s \neq V)$ **do**
  find $j^* \in V \setminus V_s$ a. î. $\beta[j^*] = \min_{j \in V \setminus V_s} \beta[j]$;
  $V_s \leftarrow V_s \cup \{j^*\}$; $E_s \leftarrow E_s \cup \{\alpha[j^*]j^*\}$;
  for $(j \in V \setminus V_s)$ **do**
    if $(\beta[j] > c[j^*j])$ **then**
      $\beta[j] \leftarrow c[j^*j]$; $\alpha[j] \leftarrow j^*$;

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



- Se observă că (S) este satisfăcută după iniţializare. În bucla while, strategia metodei generale este respectată şi de asemeni semnificaţia lui (S) este păstrată de testul din bucla for.
- Complexitatea timp: $\mathcal{O}(n-1) + \mathcal{O}(n-2) + \cdots + \mathcal{O}(1) = \mathcal{O}(n^2)$ care este bună pentru grafuri de dimensiune $\mathcal{O}(n^2)$.

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

- **În algoritmul lui Kruskal T_s^k este unul dintre cei doi arbori din familia $\mathcal{T}^k$, legaţi printr-o muchie de cost minim printre toate muchiile cu extremităţi în arbori diferiţi ai familiei.**
- Această alegere poate fi făcută prin sortarea iniţială a muchiilor crescător după cost şi, după aceea, prin parcurgerea listei astfel obţinute. Dacă notăm cu T arborele T_s^k, algoritmul poate fi descris astfel.

sort $E = \{e_1, e_2, \ldots, e_m\}$ a. î. $c(e_1) \leqslant \ldots \leqslant c(e_m)$;
$T \leftarrow \varnothing$; $i \leftarrow 1$;
while $(i \leqslant m)$ **do**
  if $(\langle T \cup \{e_i\}\rangle_G$ 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

- 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 mulţimile de noduri ale arbori, $V(T_1^k), V(T_2^k), \ldots, V(T_{n-k}^k)$ (la fiecare pas k al metodei generale), şi să testăm dacă muchia curentă are ambele extremităţi in aceeaşi mulţime.
- Aceste mulţimi 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 mulţimea de noduri ale grafului G din acel arbore.
- Mai precis, avem a funcţie $find(v)$ care determină cărei mulţimi îi aparţine nodul v, adică, **returnează rădăcina arborelui care reţine mulţimea lui v**.

Algoritmul lui Kruskal – Union-Find

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

- În metoda generală este necesară o reuniune (disjunctă) a mulţimilor de noduri a doi arbori (pentru a obţine $\mathcal{T}^{k+1}$).
- Folosim procedura $union(u, w)$ cu următoarea semnificaţie: **realizează reuniunea a două mulţimi de noduri, una căreia îi aparţine v şi una căreia îi aparţine w**.
- Putem rescrie bucla while a algoritmului, folosind aceste două proceduri, după cum urmează:

while $(i \leqslant m)$ **do**
  let $e_i = vw$;
  if $(find(v) \neq find(w))$ **then**
    $union(v, w)$;
    $T \leftarrow T \cup \{e_i\}$;
  $i++$;

Algoritmul lui Kruskal – Union-Find – Prima soluţie

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

- Tabloul $root[1..n]$ cu elemente din V are semnificaţia: $root[v] =$ rădăcina arborelui care reţine mulţimea căreia îi aparţine v.
- Adaugă la pasul de iniţializare (corespunzând familiei $\mathcal{T}^0$):

  for $(v \in V)$ **do**
  $\quad root[v] \leftarrow v;$
- Funcţia *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 $(i \in V)$ **do**
  $\quad$ **if** $(root[i] = root[v])$ **then**
  $\quad\quad root[i] = root[w];$

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

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

- Sunt $\mathcal{O}(m)$ apeluri ale funcţiei *find în timpul buclei while.*
- *În şirul de apeluri find, se intercalează exact* $n-1$ *apeluri union (un apel pentru fiecare muchie din MST-ul final).*
- *Astfel, time necesar buclei while este* $\mathcal{O}(m\mathcal{O}(1)+(n-1)\mathcal{O}(n)) = \mathcal{O}(n^2)$.

Complexitatea timp a algoritmului este $\mathcal{O}(max(m \log n, n^2))$.
Dacă G are multe muchii, $m = \mathcal{O}(n^2)$, algoritmul lui Prim este mai eficient.

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

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

- Tabloul $pred[1..n]$ cu elemente din $V \cup \{0\}$ are semnificaţia $pred[v]$ =nodul dinaintea lui v pe drumul unic către v de la rădăcina arborelui care reţine mulţimea căruia îi aparţine v.
- Adaugă la pasul de iniţializare (corespunzând familiei $\mathcal{T}^0$):

  for $(v \in V)$ **do**
  $\quad pred[v] \leftarrow 0;$

- Funcţia $find(v)$ are complexitatea in $\mathcal{O}(h(v))$, unde $h(v)$ este lungimea drumului din arbore de la v la rădăcina acestui arbore:

  function $find(v : V)$;
  $i \leftarrow v$;// o variabilă locală.
  while $(pred[i] > 0)$ **do**
  $\quad i \leftarrow pred[i]$;
  return i;

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

Algoritmul lui Kruskal – Union-Find – A doua soluţie

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $union(root_1, root_2 : V)$
  $pred[root_1] \leftarrow root_2;$

- Bucla while a algoritmului este modificată astfel:

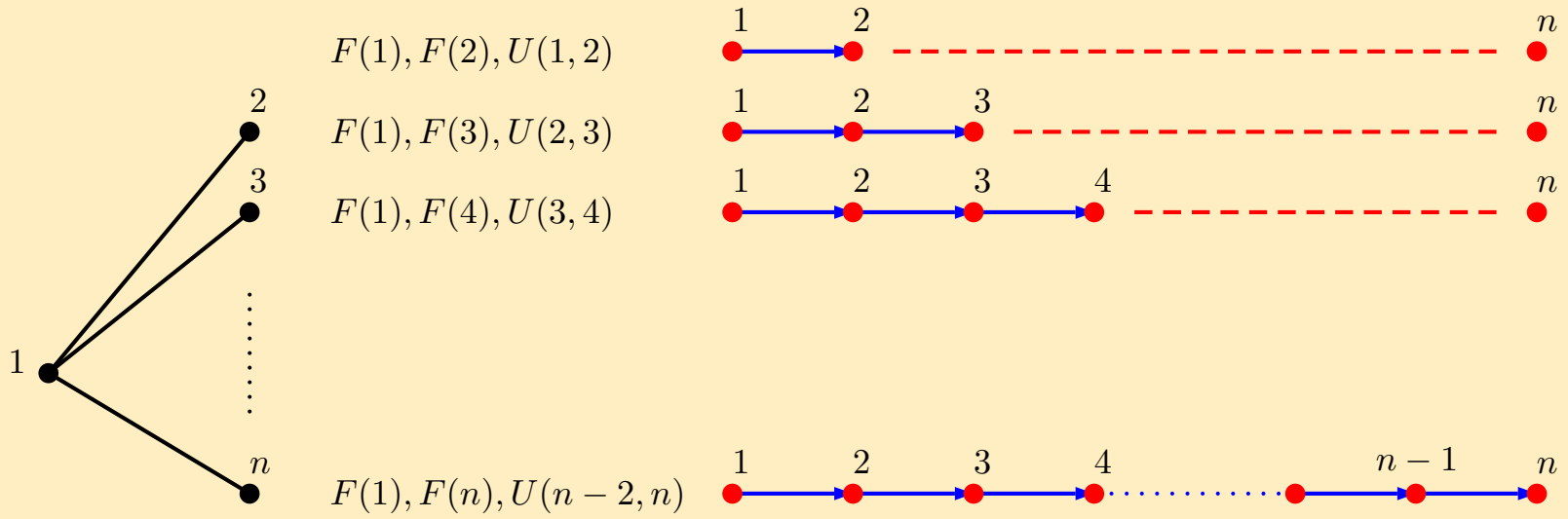
  while $(i \leqslant m)$ **do**
  let $e_i = vw$; $x \leftarrow find(v)$; $y \leftarrow find(w)$;
  if $(x \neq y)$ **then**
  $union(x, y)$;
  $T \leftarrow T \cup \{e_i\}$;
  $i++$;

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

Algoritmul lui Kruskal – Union-Find – A doua soluţie

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

Dacă executăm bucla while în această formă pentru graful $G = K_{1,n-1}$ cu lista sortată amuchiilor, $E = \{12, 13, ..., 1n\}$, atunci şirul de apeluri ale celor două proceduri este (F şi U abreviază find şi *union*):



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

Algoritmul lui Kruskal – Union-Find – A doua soluţie

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 acelui arbore care reţine 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 ţinând în rădăcina fiecărui arbore cardinalul mulţimii pe care arborele o reţine. Mai precis, semnificaţia $pred[v]$, unde v este o rădăcină, este:

$$pred[v] < 0 \Leftrightarrow v \text{ este rădăcina unui arbore}$$

$$\text{care reţine o mulţime cu } -pred[v] \text{noduri}$$

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

Algoritmul lui Kruskal – Union-Find – A doua soluţie

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

  for $(v \in V)$ **do**
  $\quad pred[v] \leftarrow -1;$

- Procedura *union* are complexitatea $\mathcal{O}(1)$ pentru a întreţine noua semnificaţie:

  procedure $union(root_1, root_2 : V)$
  $t \leftarrow pred[root_1] + pred[root_2];$
  if $(-pred[root_1] \geqslant -pred[root_2])$ **then**
  $\quad pred[root_2] \leftarrow root_1;\ pred[root_1] \leftarrow t;$
  else
  $\quad pred[root_1] \leftarrow root_2;\ pred[root_2] \leftarrow t;$

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

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

Afirmaţie. Cu această implementare a procedurilor **find** şi **union** algoritmul are următorul invariant:

$$(*)\ \forall v \in V, -pred[find(v)] \geqslant 2^{h(v)}$$

Cu alte cuvinte, numărul de noduri din arborele căruia îi aparţine v este cel puţin 2 la puterea "distanţa de la v la rădăcină".

Demonstraţia afirmaţiei. După pasul de iniţializare avem $h(v) = 0$, $find(v) = v$, şi $-pred[v] = 1$, $\forall v \in V$, deci $(*)$ are loc cu egalitate.

Să presupunem că $(*)$ are loc înaintea unei iteraţii din bucla while. Sunt posibile două cazuri:

- În această iteraţie while nu este apelată **union**. Tabloul pred nu este actualizat, deci $(*)$ are loc şi după această iteraţie.

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

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 iteraţie while avem un apel al procedurii **union**. Fie $union(x, y)$ acest apel, şi să presupunem că în procedura union se execută atribuirea $pred[y] \leftarrow x$. Aceasta înseamnă că înaintea aceastei iteraţii avem $-pred[x] \geqslant -pred[y]$.

  Nodurile v pentru care $h(v)$ se modifică în această iteraţia sunt acelea pentru care, înaintea iteraţiei aveam $find(v) = y$ şi $-pred[y] \geqslant 2^{h(v)}$.

  După iteraţia while, avem $h'(v) = h(v) + 1$ şi $find'(v) = x$. Astfel, trebuie să verificăm că $-pred'[x] \geqslant 2^{h'v)}$. Într-adevăr, $-pred'[x] = -pred[x] - pred[y] \geqslant 2 \cdot (-pred[y]) \geqslant 2 \cdot 2^{h(v)} = 2^{h(v)+1} = 2^{h'(v)}$.

Urmează că $(*)$ este un invariant al algoritmului. $\square$

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 soluţie

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $(*)$ obţinem

$$h(v) \leqslant \log(-pred[find(v)]) \leqslant \log n, \forall v \in V.$$

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 *

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

Complexitatea timp a buclei while din soluţia de mai sus este datorată şirului 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 puţin timp. Mai precis, funcţia 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 soluţie

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

Dacă $A : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ este funcţia lui Ackermann dată prin:

$$(1) \qquad A(m,n) = \begin{cases} n+1, & \text{dacă } m = 0 \\ A(m-1,1), & \text{dacă } m > 0 \text{ şi } n = 0 \\ A(m-1, A(m, n-1)), & \text{dacă } m > 0 \text{ şi } n > 0 \end{cases}$$

şi dacă notăm, $\forall m \geqslant n > 0$,

$$\alpha(m,n) = \min\{z \ : \ A(z, 4\lceil m/n \rceil) \geqslant \log n, z \geqslant 1\}$$

obţinem că complexitatea timp a buclei **while** folosind **union** din cea de-a doua soluţie şi **find** de mai sus, devine $\mathcal{O}(m \cdot \alpha(m,n))$.
Să notăm că $\alpha(m,n)$ este o funcţie care creşte foarte încet (pentru valori practice ale lui n, $\alpha(m,n) \leqslant 3$); astfel cea de-a treia soluţie 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 $A \subseteq E$ şi $v \in V$, notăm $d_A(v) = |\{e : e \in A, e \text{ incidentă cu } v\}|$, i. e., gradul lui v în subgraful generat de A, $\langle A \rangle_G$.

Definiţie

Un **cuplaj (mulţime independentă de muchii)** în G este o mulţime de muchii $M \subseteq E$ astfel încât

$$d_M(v) \leqslant 1, \forall v \in V.$$

Familia tuturor cuplajelor din graful G este notată cu $\mathcal{M}_G$:

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

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

Se observă că $\mathcal{M}_G$ satisface:

(i) $\varnothing \in \mathcal{M}_G$;

(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 mulţimea tuturor nodurilor lui G saturate de M este notată cu $S(M)$. Evident,

$$S(M) = \bigcup_{e \in M} e, \text{ şi } |S(M)| = 2 \cdot |M|.$$

- Un nod $v \in V$ cu $d_M(v) = 0$ este numit **expus** (relativ) la M, iar mulţimea tuturor nodurilor lui G expuse relativ la M este notată cu $E(M)$. Evident că $E(M) = V \setminus S(M)$, şi $|E(M) = |V| - 2 \cdot |M|$.

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

Problema cuplajului maxim:

P$_1$ 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 *

Definiţie

O **acoperire cu muchii** a lui G este o mulţime de muchii $F \subseteq E$ astfel încât

$$d_F(v) \geqslant 1, \forall v \in V(G).$$

Familia acoperirilor cu muchii ale graful 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 proprietăţi

(i) $\mathcal{F}_G \neq \varnothing \Leftrightarrow G$ nu are noduri izolate (caz în care $E \in \mathcal{F}_G$);

(ii) $F \in \mathcal{F}_G, F' \supseteq F \Rightarrow F' \in \mathcal{F}_G$.

Problem acoperirii minime cu muchii:

$\mathbf{P_2}$ dat un graf $G = (V, E)$, găsiţi $F^* \in \mathcal{F}_G$ astfel încât

$$|F^*| = \min_{F \in \mathcal{F}_G} |F|.$$

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 şi 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

Demonstraţie: "$\leqslant$" Fie M^* un cuplaj de cardinal maxim in G; considerăm următorul algoritm:

$F \leftarrow M^*$;
for $(v \in E(M^*))$ **do**
  find $v' \in S(M^*)$ a. î. $vv' \in E$;
  $F \leftarrow F \cup \{vv'\}$;

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

Demonstraţie (continuare).

Să notăm că, $\forall v \in E(M^*)$, deoarece G nu are noduri izolate, există o muchie incidentă cu v, şi, cum M^* este maximal relativ la relaţia de incluziune, această muchie are celălalt capăt în $S(M^*)$.

Mulţimea F de muchii construită este o acoperire cu muchii şi $|F| = |M^*| + |E(M^*)| = |M^*| + n - 2 \cdot |M^*| = n - |M^*|$. Astfel

$$|F^*| \leqslant |F| = n - |M^*|. \tag{2}$$

"$\geqslant$" 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;
  $M \leftarrow M \setminus \{e\}$;

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

Demonstraţie (continuare).

Evident că algoritmul construieşte un cuplaj M în G. Dacă muchia e incidentă cu v (eliminată din M într-o iteraţie while) este $e = vv'$, atunci $d_M(v') = 1$ şi în următoarea iteraţie vom avea $d_M(v') = 0$, astfel la fiecare iteraţie while este creat un nod expus relativ la cuplajul final, M (dacă ar exista o altă muchie e', în mulţimea curentă M, incidentă cu v', atunci deoarece $e \in F^*$, $F^* \setminus \{e\}$ ar fi o acoperire cu muchii, în contradicţie cu alegerea lui F^*).

Astfel, dacă M este cuplajul construit de algoritm, avem: $|F^*| - |M| = |E(M)| = n - 2 \cdot |M|$, i. e.,

$$|F^*| = n - |M| \geqslant n - |M^*|. \tag{3}$$

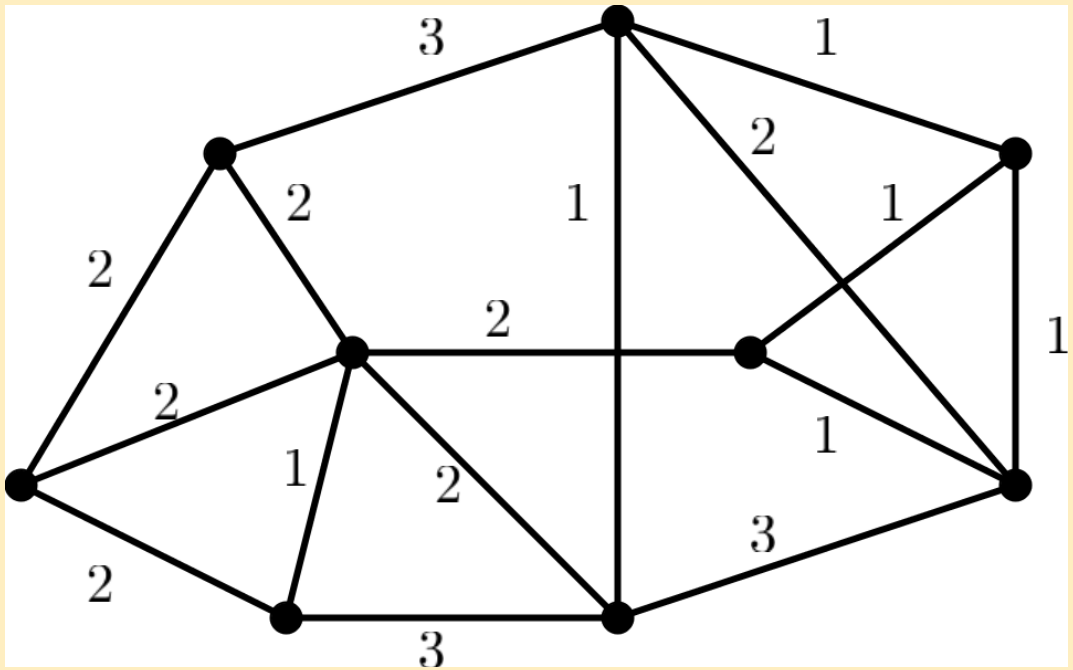
Din (2) şi (3) derivă concluzia teoremei. $\square$

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

Exerciţii 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 -

Exercise 1'. Determinaţi un arbore parţial de cost minim în graful de mai jos.



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

Exerciţii pentru seminarul din săptămâna următoare

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

Exerciţiul 1. Fie $G = (V, E)$ un graf conex şi $c : E \to \mathbb{R}$ o funcţie de cost pe muchiile sale. O submulţime $A \subseteq E$ este numită *tăietură* dacă există o bipartiţie (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 conţine un singur arbore parţial de cost minim.

(b) Arătaţi că, dacă c este funcţie injectivă, atunci G conţine un singur arbore parţial de cost minim.

(c) Reciprocele afirmaţiilor de mai sus sunt adevărate?

Exerciţiul 2. Fie $G = (V, E)$ un graf conex de ordin n, $c : E \to \mathbb{R}$, şi $\mathcal{T}_G^{min}$ familia arborilor săi parţiali de cost (c) minim. Definim $H = (\mathcal{T}_G^{min}, E(H))$ unde $T_1 T_2 \in E(H) \iff |E(T_1) \Delta E(T_2)| = 2$. Arătaţi că H este conex şi că diametrul său este cel mult $n - 1$.

Exerciţii 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

Exerciţiul 3. Fie $G = (V, E)$ un graf conex şi $c : R \to \mathbb{R}$. Pentru un arbore parţial $T = (V, E') \in \mathcal{T}_G$, şi $v \neq w \in V$ notăm cu P^T_{vw} singurul vw-drum din T. Arătaţi că un arbore parţial $T^* = (V, E^*)$ este de cost minim dacă şi numai dacă

$$\forall\, e = vw \in E \setminus E^*, \forall\, e' \in E(P^{T^*}_{vw}), \text{ avem } c(e) \geqslant c(e').$$

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

Exerciţiul 4. Fie $G = (V, E)$ un graf 2-muchie-conex şi $c : E \to \mathbb{R}$. Dacă $T = (V, E')$ este arbore parţial de cost minim al lui G şi $e \in E'$, $T - e$ are exact două componente conex T'_1 şi 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 $G - e$. Arătaţi că, dacă T^* este un arbore parţial de cost minim al lui G, şi $e \in E(T^*)$, atunci $T^* - e + e_{T^*}$ este un arbore parţial de cost minim $G - e$.

Exerciţiul 5. Fie $G = (V, E)$ un graf conex şi $c : E \to \mathbb{R}$ o funcţie de cost injectivă pe muchiile sale. Considerăm următorul algoritm

for $(e \in E)$ **do**

$\gamma(e) \leftarrow r$; // toate muchiile sunt colorate cu roşu; în timpul execuţiei vor fi roşii, albastre sau verzi

while $((\exists A \subseteq E$, tăietură cu $\gamma(e') \neq v$, unde $c(e') = \min_{e \in A} c(e))$ sau

$(\exists C$, un circuit cu $\gamma(e') \neq a$, unde $c(e') = \max_{e \in C} c(e)))$ **do**

pentru o tăietură, A, $\gamma(e') \leftarrow v$;

pentru un circuit, C, $\gamma(e') \leftarrow a$;

return $H = (V, \{e \in E : \gamma(e) = v\})$;

Arătaţi că

(a) o muchie aparţine unui arbore parţial de cost minim dacă şi numai dacă este de cost minim într-o tăietură;

(b) o muchie nu aparţine nici unui arbore parţial de cost minim dacă şi numai dacă este de cost maxim într-un circuit;

Exerciţiul 5 (continuare).

(c) algoritmul nu se opreşte câtă vreme mai există muchii roşii în graf;

(d) algoritmul se opreşte pentru orice alegere a muchiilor e', iar H este singurul arbore parţial de cost minim din G.

Exerciţiul 6. Fie H un graf conex, $\varnothing \neq A \subseteq V(H)$, şi $w : E(H) \to \mathbb{R}_+$. 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 toţi arborii care conţin A, şi care sunt sunt subgrafuri ale lui H:

$$s[T(H, A, w)] = \sum_{e \in E_T} w(e) =$$

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

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

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

Exerciţiul 6 (continuare).

(b) Fie $G = (V, E)$ un graf conex cu $V = \{1, 2, \ldots, n\}$, şi $A \subseteq V$; avem de asemeni şi o funcţie de cost $c : E \to \mathbb{R}_+$. Considerăm graful complet K_n (cu $V(K_n) = V$) şi definim $\overline{c} : E(K_n) \to \mathbb{R}_+$:

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

Demonstraţi că $s[T(G, A, c)] = s[T(K_n, A, \overline{c})]$ şi arătaţi cum se poate construi un arbore Steiner $T(K_n, A, \overline{c})$ dintr-un arbore Steiner $T(G, A, c)$.

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

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

Exerciţiul 7. Considerăm o ordonare $E = \{e_1, e_2, \ldots, e_m\}$ a muchiillor unui graf conex $G = (V, E)$ de ordin n. Pentru orice submulţime $A \subseteq E$ definim $\mathbf{x}^A \in GF^m$ vectorul caracteristic m-dimensional al mulţimii A: $\mathbf{x}_i^A = 1 \Leftrightarrow e_i \in A$. GF^m este spaţiul m-dimensional peste $\mathbb{Z}_2$.

(a) Arătaţi că submulţimea vectorilor caracteristici corespunzători tuturor tăieturilor din G împreună cu vectorul nul este un subspaţiu X al lui GF^m.

(b) Arătaţi că submulţimea vectorilor caracteristici corespunzători tuturor circuitelor din G generează un subspaţiu U al lui GF^m care este ortogonal pe X.

(c) Arătaţi că $dim(X) \geqslant n - 1$.

(d) Arătaţi că $dim(U) \geqslant m - n + 1$.

(e) În final, dovediţi că inegalităţile de mai sus sunt de fapt egalităţi.

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

Exerciţiul 8. Fie $G = (V, E)$ un graf conex şi $c : E \to \mathbb{R}$ o funcţie de cost pe muchiile sale.

a) Fie T^* un arbore parţial de cost c-minim al lui G şi $\epsilon > 0$. Arătaţi că T^* este singurul arbore parţial de cost $\overline{c}$-minim al lui G, unde

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

b) Deduceţi de aici că pentru orice arbore parţial 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

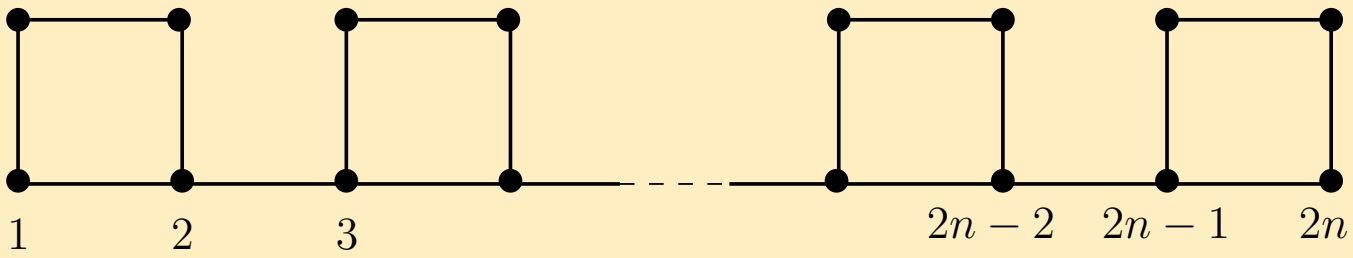
Exerciţiul 9. Fie $G = (V, E)$ un graf conex şi $c : E \to \mathbb{R}$ o funcţie de cost injectivă. Fie T^* un arbore parţial de cost minim al lui G şi T_0 un arbore parţial cu cel de-al doilea cel mai mic cost în G.

(a) T_0 este unicul arbore parţial cu cel de-al doilea cel mai mic cost în G?

(b) Arătaţi că $|E(T^*) \Delta E(T_0)| = 2$.

(c) Descrieţi un algoritm pentru a determina un arbore parţial cu cel de-al doilea cel mai mic cost în G.

Exerciţiul 10. Determinaţi numărul de cuplaje maxime ale următorului graf:



Exerciţii 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 -

Exerciţiul 11. Doi copii se joacă pe un graf dat, G, astfel: fiecare alege alternativ un nod nou $v_0, v_1, \ldots$ 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. Arătaţi că jucătorul care începe jocul are întotdeauna o strategie de câştig dacă şi numai dacă G nu are un cuplaj perfect.

Exerciţiul 12. Fie S o mulţime nevidă şi finită, $k \in \mathbb{N}^*$, iar $\mathcal{A} = (A_i)_{1 \leqslant i \leqslant k}$ şi $\mathcal{B} = (B_i)_{1 \leqslant i \leqslant k}$ două partiţii ale lui S. Arătaţi că $\mathcal{A}$ şi $\mathcal{B}$ admit un sistem comun de reprezentanţi, i. e., există $r_{\mathcal{A}}, r_{\mathcal{B}} : \{1, 2, \ldots, k\} \to S$ aşa încât pentru orice $1 \leqslant i \leqslant k$, $r_{\mathcal{A}}(i) \in A_i$ şi $r_{\mathcal{B}}(i) \in B_i$, iar cele două funcţii au aceaşi imagine.

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

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

Cuprins

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 reţele
   - Reţele de transport
   - Flux, valoarea fluxului, problema fluxului maxim
3. Exerciţii pentru seminarul din săptămâna a 9-a (25-29 noiembrie)

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

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 şi $\mathcal{M}_G$ familia cuplajelor sale.

Problem cuplajului maximum.

$\mathbf{P_1}$ dat un graf $G = (V, E)$, determinaţi $M^* \in \mathcal{M}_G$ astfel încât

$$|M^*| = \max_{M \in \mathcal{M}_G} |M|.$$

Fie $B = (b_{ij})_{n \times m}$ matricea de incidenţă a lui G (se consideră câte o ordonare fixată a celor n noduri al lui G şi 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 componentele 1, atunci problema $\mathbf{P_1}$ poate fi descrisă echivalent astfel:

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

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

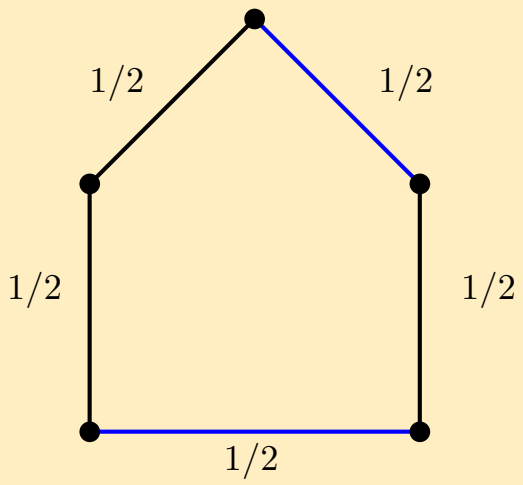
$\mathbf{P}'_1$ este o problemă ILP (Integer Linear Programming), care este în general **NP**-hard. Vom încerca să rezolvăm $\mathbf{P}'_1$, folosind problema (relaxată) asociată LP (Linear Programming)

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

Soluţiile optime ale lui $\mathbf{LP}'_1$ pot avea componente ne-întregi şi de asemeni valorile lor optime pot fi mai mari decât $\nu(G)$. De exemplu, dacă $G = C_{2n+1}$, atunci $\nu(G) = n$, dar soluţia $x_i = 1/2$, $\forall 1 \leqslant i \leqslant 2n+1$ 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 *

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



Urmează că dacă graful G are circuite impare, problema $\mathbf{P_1}$ nu poate fi rezolvată folosind $\mathbf{LP'_1}$. Mai precis,

Teorema 1

(Balinski, 1971) Punctele extreme ale politopului $B\mathbf{x} \leqslant \mathbf{1}_n$, $\mathbf{x} \geqslant 0$, $\mathbf{x} \in \mathbb{R}^m$, au componentele din $\{0, 1/2, 1\}$. Componentele $1/2$ apar dacă şi 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 $\mathbf{P_1}$ este uşoară dacă graful este bipartit: se rezolvă problema $\mathbf{LP'_1}$ şi soluţia (întreagă) optimă găsită este o soluţie optimă şi pentru $\mathbf{P_1}$.
- Adaptări combinatoriale ale algoritmului simplex pentru rezolvarea problemei de programare liniară au condus la aşa numita "metodă ungară" pentru rezolvarea $\mathbf{P_1}$ în grafuri bipartite.
- Vom discuta o soluţie mai rapidă găsită de Hopcroft şi Karp (1973).
- Cu toate acestea, teorema de de dualitate din programarea liniară şi integralitatea soluţiei optime pot fi folosite pentru a obţine şi explica rezultatele (deja dovedite) despre 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 - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 şi numai dacă

$$|N_G(A)| \geqslant |A|, \forall A \subseteq S.$$

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

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

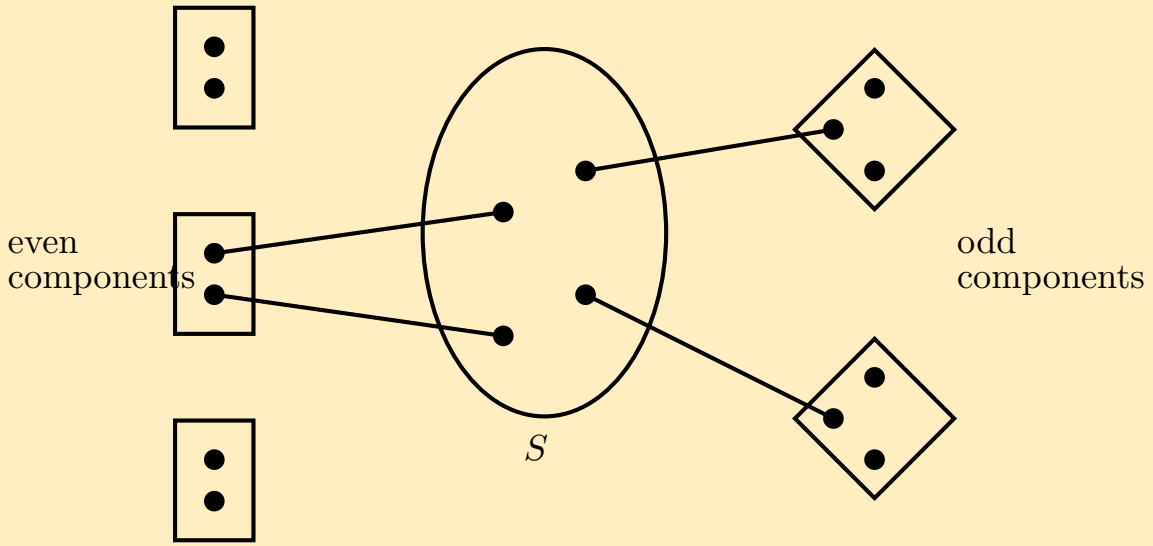
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Fie G be un graf. Evident, $|M| \leqslant |G|/2$, $\forall M \in \mathcal{M}_G$. 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 şi cealaltă în S. Deoarece extremităţile muchiilor diferite sunt distincte, urmează că $|S| \geqslant q(G - S)$.
Pentru $S = \varnothing$ în (T), obţinem $q(G - \varnothing) \leqslant 0$, deci $q(G) = 0$.

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 -

Teorema 4

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

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

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

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

Demonstraţie. Necesitatea condiţiei (T) a fost dovedită în discuţia de mai sus. Demonstrăm prin inducţie după $n = |G|$ că dacă un graf $G = (V, E)$ satisface (T), atunci G are un cuplaj perfect.

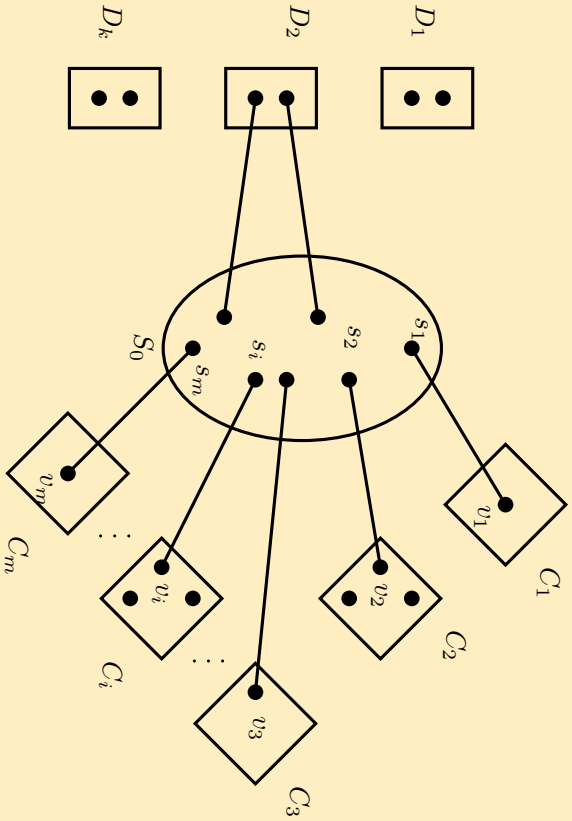
Pentru $n = 1, 2$ teorema are loc evident. În pasul inductiv, fie G un graf cu $n \geqslant 3$ noduri care satisface (T) şi să presupunem că orice graf G' cu $|G'| < n$ şi care satisface (T) are un cuplaj perfect.

Fie $S_0 \subseteq V(G)$ astfel încât $q(G - S_0) = |S_0|$ şi maximală cu proprietatea că avem egalitate în (T) (i. e., pentru orice supramulţime S a lui S_0 avem $q(G - S) < |S|$).

Observăm că familia de submulţimi ale lui $V(G)$ pentru care (T) este satisfăcută cu egalitate este nevidă, şi, deci, există un S_0 (pentru fiecare nod v_0 care nu este punct de articulaţie în componetnta lui din G, avem $q(G - v_0) = 1 = |\{v_0\}|$, deoarece fiecare componentă conexă este pară şi în fiecare componentă conexă cu cel puţin un nod, există un astfel de nod v_0).

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

Fie $m = |S_0| > 0$, $C_1, C_2, \ldots, C_m$ componentele impare ale lui $G - S_0$ şi $D_1, D_2, \ldots, D_k$ componentele pare ale lui $G - S_0$ ($k \geqslant 0$):



Vom construi un cuplaj perfect compus din

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, $\{e_1, \ldots, e_m\}$, muchia e_i având un capăt $s_i \in S$ şi celălalt $v_i \in C_i$ $(i = 1, \overline{1, m})$;

c) cuplaj perfect in fiecare subgraph $C_i - v_i$ $(i = 1, \overline{1, m})$.

a) Pentru orice $1 \leqslant i \leqslant k$ graful $[D_i]_G$ admite cuplaj perfect. Într-adevăr, deoarece $m > 0$, urmează că $|D_i| < n$ şi din ipoteza inductivă este suficient să arătăm că $G' = [D_i]_G$ satisface (T).
Fie $S' \subseteq D_i$. Dacă $q(G' - S') > |S'|$, atunci obţinem următoarea contradicţie:

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

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

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 $H = (S_0, \{C_1, \ldots, C_m\}; E')$ graful bipartit cu o clasă a bipartiţiei S_0, cealaltă clasă mulţimea componentelor conexe impare ale lui $G - S_0$, şi 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 condiţia din teorema lui Hall pentru existenţa unui cuplaj M_0 care saturează $\{C_1, \ldots, C_m\}$.

Fie $A \subseteq \{C_1, \ldots, C_m\}$. Atunci $B = N_H(A) \subseteq S_0$, şi din construcţia lui H, în graful G nu avem nicio muchie de la un nod $v \in S_0 - B$ la un nod $w \in C_i \in A$. Astfel componentele conexe impare din A rămân componente conexe impare şi în $G - B$; astfel $q(G - B) \geqslant |A|$. Deoarece G satisface condiţia lui Tutte (T), avem $|B| \geqslant q(G - B)$. Am obţinut că $|N_H(A)| = |B| \geqslant |A|$.

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

Din teorema lui Hall H are un cuplaj M_0, care saturează $\{C_1, \ldots, C_m\}$; deoarece $|S_0| = m$, M_0 este perfect.

$$M_0 = \{s_1v_1, s_2v_2, \ldots, s_mv_m\}, S_0 = \{s_1, \ldots, s_m\}, v_i \in C_i, \forall i = \overline{1, m}.$$

c) $\forall i \in \{1, \ldots, m\}$ graful $G' = [C_i - v_i]_G$ admite cuplaj perfect. Folosind ipoteza inductivă, este suficient să dovedim că G' satisface (T).
Fie $S' \subseteq C_i - v_i$. Dacă $q(G' - S') > |S'|$, atunci, deoarece $q(G' - S') + |S'| \equiv 0 \ (mod\ 2)$ (pentru că $|G'|$ este par), urmează că $q(G' - S') \geqslant |S'| + 2$.
Dacă $S'' = S_0 \cup \{v_i\} \cup S'$, avem

$$|S''| \geqslant q(G - S'') = q(G - S_0) - 1 + q(G' - S') = |S_0| - 1 + q(G' - S') \geqslant$$

$$\geqslant |S_0| - 1 + |S'| + 2 = |S''|,$$

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

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

Astfel $\forall S' \subseteq C_i - v_i$, $q(G' - S') \leqslant |S'|$ şi G' are cuplaj perfect (din ipoteza inductivă).

Evident cuplajul lui G obţinut reunind cuplajele de la a), b), şi c) de mai sus saturează toate nodurile lui G, şi demonstraţia prin inducţie a teoremei se încheie

Graph Algorithms * C. Croitoru - Graph Algorithms *

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

Fie $G = (V, E)$ un graf şi $M \in \mathcal{M}_G$ un cuplaj al lui G.

Definiţie

Un **drum alternat** *în* G relativ la cuplajul M *este un drum*

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

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

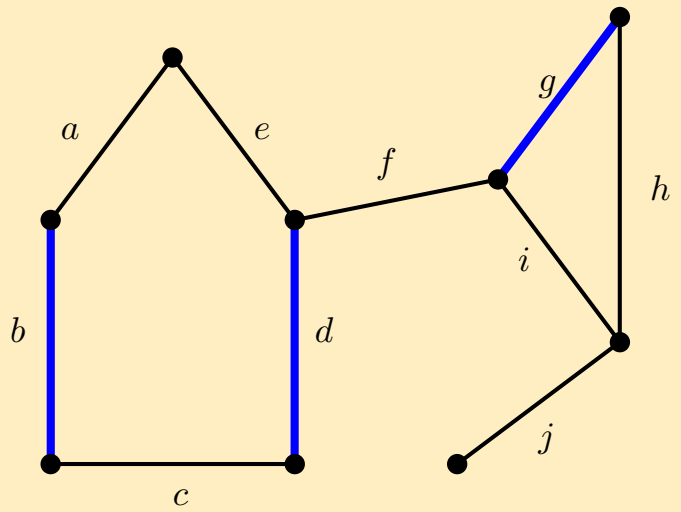
Observăm că, deoarece M este un cuplaj, dacă P este un drum alternat relativ la M, atunci dintre orice două muchii consecutive ale lui P exact una aparţine lui M (muchiile aparţin alternativ lui M şi $E \setminus M$).
În cele ce urmează, când ne vom referi la un drum P vom înţelege mulţimea muchiilor sale.

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

Definiţie

Un **drum de creştere** *al lui G* relativ la cuplajul M *este un drum alternat care uneşte două noduri distincte expuse relativ la M.*

Observăm că, din definiţia de mai sus, urmează că dacă P este un drum de creştere 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

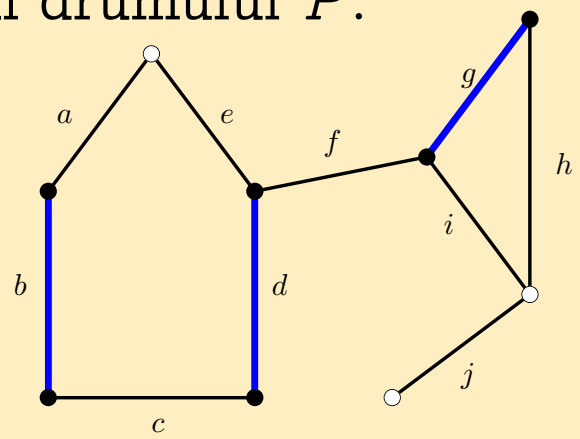
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teorema 5

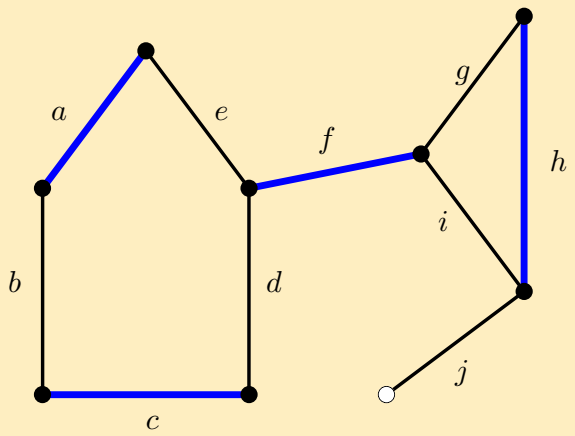
(Berge, 1959) M este un cuplaj de cardinal maxim in graful G dacă şi numai dacă nu există drum de creştere în G relativ la M.

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

Demonstraţie. "$\Rightarrow$" Fie M un cuplaj de cardinal maxim în G. Să presupunem că P este un drum de creştere 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 obţinut 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



$M \Delta P$

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 contradicţie cu alegerea lui M.

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

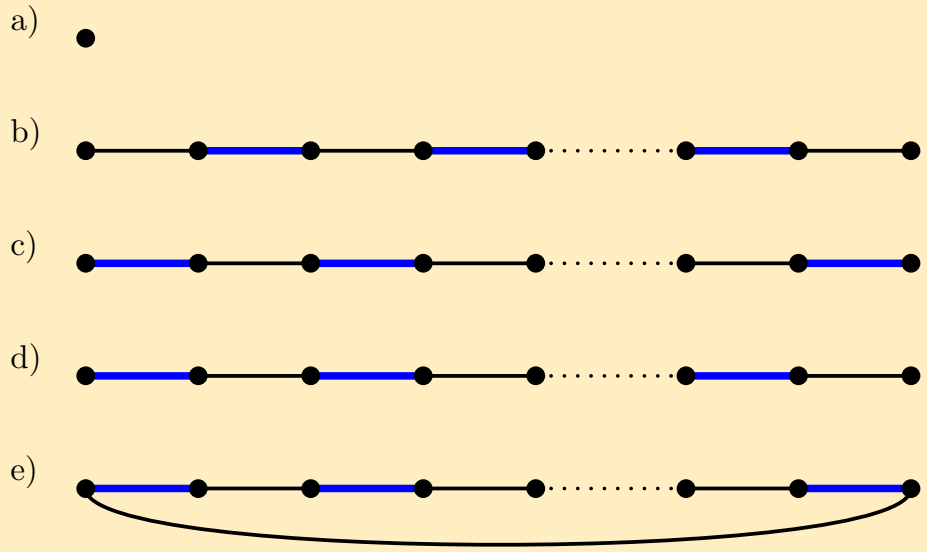
Dacă M^* este un cuplaj de cardinal maxim în G, vom arăta că$|M^*| = |M|$. Fie G' subgraful generat de $M \Delta M^*$ în G ($G' = (V, M \Delta M^*)$).

Să observăm că $d_{G'}(v) \leqslant 2, \forall v \in V$ şi deci componentele conexe ale lui G' pot fi noduri izolate, drumuri de lungime cel puţin unu, sau circuite.

Avem cinci posibilităţi (se văd mai jos; muchiile albastre sunt din M^*, muchiile negre sunt din M).

Cazul b) nu apare deoarece este un drum de creştere relativ la M^*, care este un cuplaj de cardinal maxim. Cazul c) nu apare deoarece este un drum de creştere relativ la M.

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



Dacă notăm cu $m_M(C)$ numărul de muchii din M din componenta conexă C a lui G' şi cu $m_{M^*}(C)$ numărul de muchii din M^* din aceeaşi componentă conexă a lui G', obţinem că $m_M(C) = m_{M^*}(C)$. Astfel,

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

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

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

Deci $|M| = |M^*|$. $\square$

Obţinem o strategie pentru a determina un cuplaj de cardinal maxim:

fie M un cuplaj în G (e. g., $M = \varnothing$);
while ($\exists P$ drum de creştere relativ la M) **do**
  $M \leftarrow M \Delta P$;
end while

La fiecare iteraţie while cardinalul lui M creşte cu 1, astfel, în cel mult $n/2$ iteraţii obţinem un cuplaj fără drumuri de creştere, adică de cardinal maxim. Neajunsul acestui algoritm este următorul: condiţia 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 *

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 $M, N \in \mathcal{M}_G$, $|M| = r$, $|N| = s$ şi $s > r$. Atunci în $M \Delta N$ există cel puţin $s - r$ drumuri de creştere relativ la M disjuncte pe noduri.

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

Demonstraţie. Fie $G' = (V, M \Delta N)$ şi C_i $(i = \overline{1, p})$ componentele conexe ale lui G'. Pentru fiecare $1 \leqslant i \leqslant p$, notăm cu $\delta(C_i)$ diferenţa dintre numărul de muchii ale lui N din C_i şi 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 şi N sunt cuplaje, C_i sunt drumuri sau circuite. Astfel $\delta(C_i) \in \{-1, 0, 1\}$. $\delta(C_i) = 1$ dacă şi numai dacă C_i este un drum de creştere relativ la M.

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

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

Demonstraţie (continuare). Deoarece

$$\sum_{i=1}^{p} \delta(C_i) = |N \setminus M| - |M \setminus N| = s - r,$$

urmează că există cel puţin $s - r$ componente conexe ale lui G' cu $\delta(C_i) = 1$, adică, există cel puţin $s - r$ drumuri de creştere disjuncte pe noduri conţinute în $M \Delta N$. $\square$

Lema 2

Dacă $\nu(G) = s$ şi $M \in \mathcal{M}_G$ cu $|M| = r < s$, atunci există în G un drum de creştere relativ la M de lungime cel mult $2\lceil r/(s-r) \rceil + 1$.

Demonstraţie. Fie $N \in \mathcal{M}_G$ cu $|N| = s = \nu(G)$.

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

Demonstraţie (continuare). Din Lema 1, există $s - r$ drumuri de creştere disjuncte pe muchii (drumurile disjuncte pe noduri sunt şi disjuncte pe muchii) conţinute în $M \Delta N$. Urmează că cel puţin unul dintre ele are cel mult $\lceil r/(s-r) \rceil$ muchii din M. Lungimea acestui drum de creştere este cel mult $2\lceil r/(s-r) \rceil + 1$. $\square$

Definiţie

Fie $M \in \mathcal{M}_G$. Un drum de creştere minim *relativ la M în G este un drum de creştere de lungime minimă printre toate drumurile de creştere 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 creştere minim relativ la M, şi P' un drum de creştere relativ la $M \Delta P$. Atunci $|P'| \geqslant |P| + 2|P \cap P'|$.

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

Demonstraţie. Fie $N = (M \Delta P) \Delta P'$. Atunci $M \Delta N = P \Delta P'$ şi $|N| = |M| + 2$. Din Lema 1, există două drumuri de creştere relativ la M disjuncte pe muchii, P_1 şi P_2, conţinute în $M \Delta N$. Deoarece P este un drum de creştere minim relativ la M, avem $|P \Delta P'| \geqslant |P_1| + |P_2| \geqslant 2|P|$ şi, deci, $|P| + |P'| - 2|P \cap P'| \geqslant 2|P|$. $\square$

Considerăm următorul algoritm:

$M_0 \leftarrow \varnothing$; $i = 0$;
while ($\exists$ drumuri de creştere relativ la M) **do**
  let P_i un drum de creştere minim relativ la M_i;
  $M_{i+1} \leftarrow M_i \Delta P_i$;
  $i++$;
end while

Fie $P_0, P_1, \ldots, P_{\nu(G)-1}$ secvenţa de drumuri de creştere minime construite de acest algoritm.

C. Croitoru - Graph Algorithms * C. Croitoru - 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) $|P_i| \leqslant |P_{i+1}|$ şi $|P_i| = |P_{i+1}|$ dacă şi numai dacă P_i şi P_{i+1} sunt disjuncte pe noduri, $\forall 1 \leqslant i \leqslant \nu(G) - 2$.
b) $\forall i < j < \nu(G) - 1$, dacă $|P_i| = |P_j|$, atunci P_i şi P_j sunt disjuncte pe noduri.

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

Demonstraţie. a) Luând $P = P_i$ şi $P' = P_{i+1}$ în Lema 3, obţinem $|P_{i+1}| \geqslant |P_i| + 2|P_i \cap P_{i+1}| \geqslant |P_i|$. Egalitatea are loc dacă şi numai dacă P_i şi P_{i+1} sunt disjuncte pe muchii, de unde rezultă că sunt disjuncte şi pe noduri (fiind drumuri alternate).
b) rezultă aplicând succesiv a) $\square$

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

Teorema 6

(Hopcroft, Karp, 1973) Fie G un graf cu $\nu(G) = s$. Numărul de întregi distincţi din secvenţa $|P_0|, |P_1|, \ldots, |P_{s-1}|$ (P_i sunt drumuri de creştere minime construite de algoritmul de mai sus) nu este mai mare decât $2\lfloor\sqrt{s}\rfloor + 2$.

Demonstraţie. Fie $r = \lceil s - \sqrt{s}\rceil$. Atunci $|M_r| = r$ şi

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

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-secvenţa $|P_r|, \ldots, |P_{s-1}|$ există cel mult $s - r \leqslant \lfloor\sqrt{s}\rfloor + 1$ întregi distincţi. Urmează că în secvenţa $|P_0|, |P_1|, \ldots, |P_{s-1}|$ nu există mai mult de $2\lfloor\sqrt{s}\rfloor + 2$ întregi distincţi. $\square$

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

Dacă algoritmul de mai sus este descompus în faze astfel încât în fiecare fază se determină o familie maximală de drumuri de creştere minime disjuncte pe noduri , atunci - din Lema 4 - lungimea drumurilor de creştere minime din faza următoare nu va descreşte (altfel, mulţimea drumurilor de creştere minime construite în faza curentă nu este maximală).

Din Teorema 6, obţinem 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:

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

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

$M \leftarrow \varnothing$;
repeat
  determină $\mathcal{P}$ o familie maximală de drumuri de creştere minime relativ la M disjuncte pe noduri;
  for $(P \in \mathcal{P})$ **do**
    $M \leftarrow M \Delta P$;
  end for
until $(\mathcal{P} = \varnothing)$

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** şi **Karp** au arătat că aceasta poate fi obţinută în $\mathcal{O}(n+m)$ şi, deci, întreg algoritmul are complexitatea timp $\mathcal{O}(m\sqrt{n})$.

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

Acest rezultat a fost extins la grafuri arbitrare de către **Micali** şi **Vazirani (1980)** folosind o structură de date elaborată pentru a întreţine etichetele asociate nodurilor pentru construcţia drumurilor de creştere minime.

Să considerăm cazul grafurilor bipartite: $G = (S, T; E)$ şi $M \in \mathcal{M}_G$. Pornind cu una dintre clase, de exemplu S, considerăm mulţimea extremităţilor iniţiale ale unor drumuri de creştere $S \cap E(M)$. Din fiecare astfel de nod pornim, în paralel, construcţia unor drumuri alternate într-o manieră bfs. Primul drum de creştere obţinut opreşte construcţia, oferind lungimea minimă a unui drum de creştere. Familia $\mathcal{P}$ este obţinută folosind etichetele şi listele de adiacenţă în $\mathcal{O}(n + m)$.

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

- 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 *

Reţele 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 **reţea (de transport)** cu sursa s şi destinaţia t este o tuplă $R = (G, s, t, c)$ unde:

- $G = (V, E)$ este un digraf,
- $s, t \in V$; $s \neq t$; $d_G^+(s) > 0$; $d_G^-(t) > 0$,
- $c : E \to \mathbb{R}_+$; $c(e)$ este capacitatea arcului e.

Vom presupune că $V = \{1, 2, \ldots, n\}$ ($n \in \mathbb{N}^*$) şi $|E| = m$. Extindem funcţia c la $c : V \times V \to \mathbb{R}_+$ prin

$$c((i,j)) = \begin{cases} c(ij), & \text{dacă } ij \in E \\ 0, & \text{altfel} \end{cases}$$

şi notăm $c((i,j)) = c_{ij}$.

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

Definiţie

Un **flux** *in* $R = (G, s, t, c)$ *este o funcţie* $x : V \times V \to \mathbb{R}$ *a. î.*

(i) $0 \leqslant x_{ij} \leqslant c_{ij}, \forall (i,j) \in V \times V,$

(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.
- Constrângerile (i) cer ca **fluxul pe fiecare arc să fie ne-negativ şi să nu depăşească capacitatea**.
- Constrângerile (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 iese**).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 funcţia c la $V \times V$, constrângerile (i) implică $x_{ij} = 0$ când $ij \notin E$. Astfel, un flux este de fapt o funcţie definită pe E. Preferăm extensia sa pe $V \times V$ pentru a simplifica notaţiile.

Fie x un flux în $R = (G, s, t, c)$. Dacă adunăm toate constrângerile (ii) (pentru $i \in V \setminus \{s, t\}$) obţinem

$$0 = \sum_{i \neq s,t} \left(\sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} \right) = \sum_{i \neq s,t} \sum_{j \neq s,t} x_{ji} - \sum_{i \neq s,t} \sum_{j \neq s,t} x_{ij} +$$

$$+ \sum_{i \neq s,t} x_{si} + \sum_{i \neq s,t} x_{ti} - \sum_{i \neq s,t} x_{is} - \sum_{i \neq s,t} x_{it} =$$

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

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

$$= \left(\sum_{i \in V} x_{si} - \sum_{i \in V} x_{is} \right) - \left(\sum_{i \in V} x_{it} - \sum_{i \in V} x_{ti} \right),$$

Definiţie

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 destinaţia reţelei sau, după cum am demonstrat mai sus, fluxul net care părăseşte sursa reţelei.
Să observăm că în orice reţea $R = (G, s, t, c)$ există un flux: x^0, **fluxul nul**, cu $x^0_{ij} = 0$, $\forall ij \in E$ şi $v(x^0) = 0$.

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

Problema fluxului maxim: Dată $R = (G, s, t, c)$ o reţea, să se determine un flux de valoare maximă.

Problema fluxului maxim poate fi văzută ca o problemă LP:

$$\begin{aligned} \max \quad & v \\ & \sum_{j \in V} x_{ji} - \sum_{j \in V} x_{ij} = 0, \forall i \neq s, t \\ & \sum_{j \in V} x_{js} - \sum_{j \in V} x_{sj} = -v \\ & \sum_{j \in V} x_{jt} - \sum_{j \in V} x_{tj} = v \\ & 0 \leqslant x_{ij} \leqslant c_{ij}, \forall ij \in E \end{aligned}$$

Cu toatea acestea, vom considera a abordare combinatorială directă care este importantă când există constrângeri de integralitate a unor variabile (de exemplu fluxul de pe arce).

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

Exerciţiul 1. Fie X o mulţime finită, $X_1, \ldots, X_n \subseteq X$, şi $d_1, d_2, \ldots d_n \in \mathbb{N}$. Arătaţi că există n submulţimi disjuncte $Y_i \subseteq X_i$, $|Y_i| = d_i$, $\forall i = \overline{1, n}$ dacă şi numai dacă

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

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

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

Exerciţiul 2. Orice graf p-regulat bipartit admite cuplaj perfect.

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

Exerciţiul 3. Fie $G = (S, T; E)$ un graf bipartit. Folosind teorema lui Hall demonstraţi că, pentru fiecare $0 \leqslant k \leqslant |S|$, G are un cuplaj de cardinal cel puţin $|S| - k$ dacă şi numai dacă $|N_G(A)| \geqslant |A| - k$, $\forall A \subseteq S$.

Exerciţiul 4. Utilizând teorema lui Tutte arătaţi că un graf 2-muchie conex şi 3-regulat admite cuplaj perfect.

Exerciţiul 5. Fie $G = (V, E)$ un graf. O submulţime $A \subseteq V$ este numită *m-independentă* dacă G are un cuplaj M care saturează toate nodurile din A. Arătaţi că, pentru orice două mulţimi *m-independente* A şi B cu $|A| < |B|$, se poate determina un nod $b \in B \setminus A$ astfel încât $A \cup \{b\}$ este de asemeni *m-independentă* ($\Rightarrow$ toate mulţimile maximal *m-independente* au acelaşi cardinal).

Exerciţiul 6. Fie $T = (V, E)$ un arbore cu rădăcină; îi notăm cu r rădăcina şi cu $parent(v)$ strămoşul direct al oricărui nod $v \neq r$. Un cuplaj M al lui T este numit *propriu* dacă orice nod nesaturat de M, $v \neq r$, are un frate w astfel încât $w\, parent(v) \in M$.

(a) Arătaţi că orice cuplaj *propriu* este un cuplaj de cardinal maxim.

(b) Găsiţi în $\mathcal{O}(n)$ un cuplaj *propriu* pentru un arbore de ordin n.

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

Cuprins

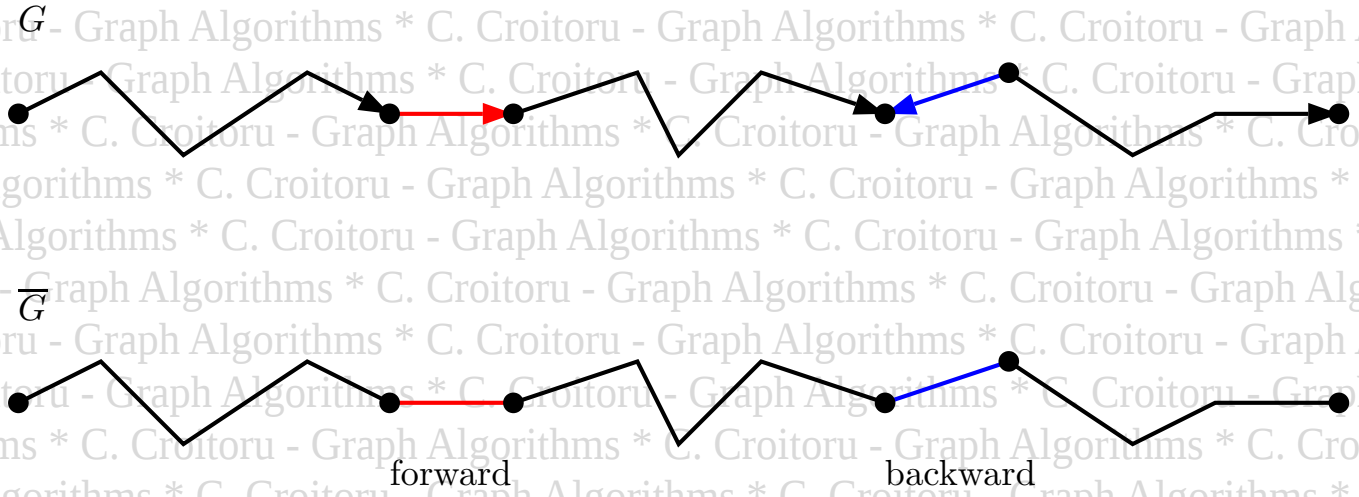
1. Fluxuri în reţele
   - Drumuri de creştere
   - Secţiuni
   - Teorema drumului de creştere
   - Teorema fluxului întreg
   - Teorema flux maxim – secţiune minimă
   - Algoritmul lui Ford & Fulkerson
   - Algoritmul lui Edmonds & Karp
2. Exerciţii pentru seminarul din săptămâna viitoare

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

Definiţie

Fie P un drum în $\widetilde{G}$ – graful suport al digrafului G – şi $e = v_i v_j$ o muchie a lui P, $e \in E(P)$. Dacă e coresponde arcului $v_i v_j$ atunci e este un **arc înainte** *(***forward** *sau* **direct***) al lui P, altfel (e corespunde arcului $v_j v_i$) e este un* **arc înapoi** *(***backward** *sau* **invers***) al lui P.*



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

Definiţie

Fie $R = (G, s, t, c)$ o reţea şi 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 direct, atunci $x_{ij} < c_{ij}$,*
- *dacă ij este a arc invers, atunci $x_{ji} > 0$.*

Dacă P este un A-drum în R relativ la fluxul x, atunci **capacitatea reziduală** *a arcului $ij \in E(P)$ este*

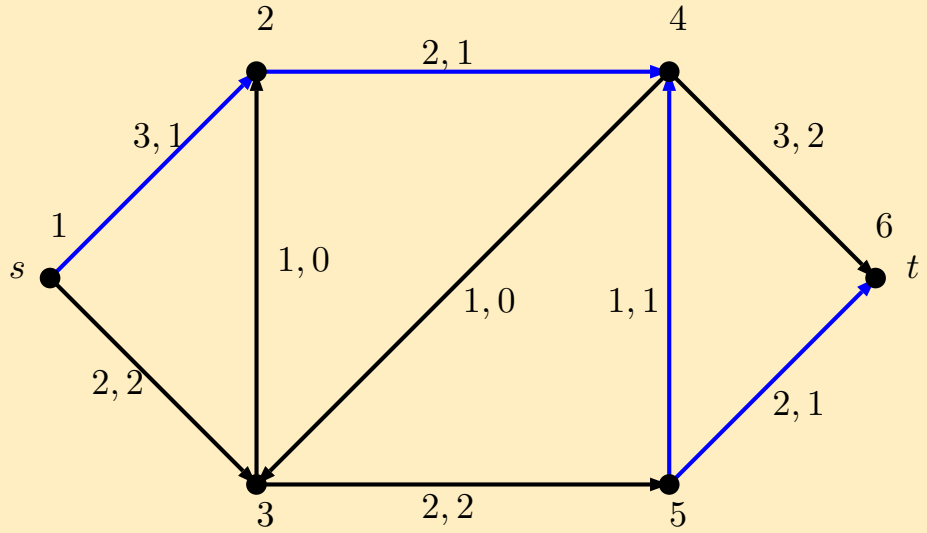
$$r(ij) = \begin{cases} c_{ij} - x_{ij}, & \text{dacă } ij \text{ este un arc direct al lui } P \\ x_{ji}, & \text{dacă } ij \text{ este un arc invers al lui } P \end{cases}$$

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

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

Exemplu

Considerăm reţeaua 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 ($x_{56} = 1 < c_{56} = 2$), şi arcul backward 45 ($x_{45} = 1 > 0$). Capacitatea rexidulă a lui P: $r(P) = \min\{2, 1, 1, 1\} = 1$.

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

Definiţie

Un **drum de creştere** *relativ la fluxul x în reţeaua $R = (G, s, t, c)$ 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 creştere relativ la fluxul x în $R = (G, s, t, c)$, atunci $x^1 = x \otimes r(P)$ definit prin

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

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

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

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

Demonstraţie. Din definiţia 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 şi ik. Avem următoarele două cazuri posibile:

a) li şi ik sunt arce directe:

$$\sum_j x^1_{ji} - \sum_j x^1_{ij} = \sum_{j \neq l} x_{ji} - \sum_{j \neq k} x_{ij} + x^1_{li} - x^1_{ik}$$

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

b) li este arc direct şi ik arc invers:

$$\sum_j x^1_{ji} - \sum_j x^1_{ij} = \sum_{j \neq l,k} x_{ji} - \sum_j x_{ij} + x^1_{li} + x^1_{ki}$$

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

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

c) li arc invers şi ik arc direct: similar cu b).

d) li şi ik arce inverse: similar cu a).

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

lt arc direct:

$$v(x^1) = \sum_j x^1_{jt} - \sum_j x^1_{tj} = \sum_{j \neq l} x_{jt} - \sum_j x_{tj} + x^1_{lt} =$$

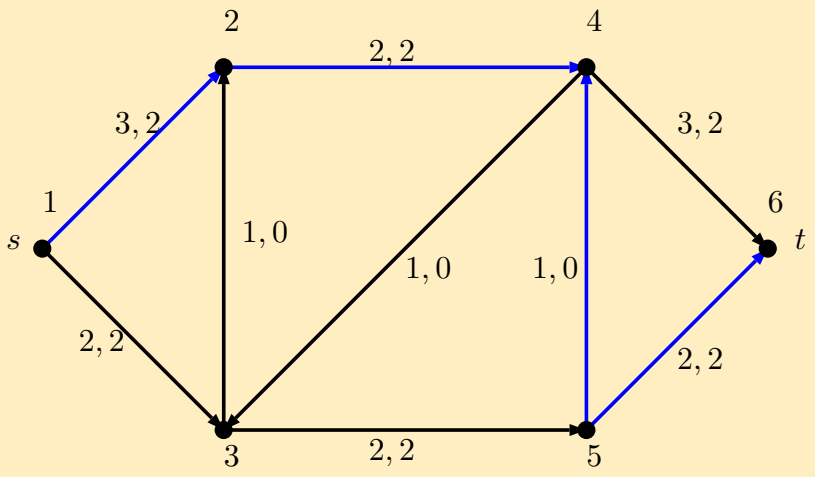
$$= \sum_{j \neq l} x_{jt} - \sum_j x_{tj} + x_{lt} + r(P) = v(x) + r(P).$$

lt arc invers:

$$v(x^1) = \sum_j x^1_{jt} - \sum_j x^1_{tj} = \sum_j x_{jt} - \sum_{j \neq l} x_{tj} - x^1_{tl} =$$

$$= \sum_j x_{jt} - \sum_{j \neq l} x_{tj} - (x_{tl} - r(P)) = v(x) + r(P). \square$$

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 şi numele drumurilor de creştere şi capacitatea reziduală.
- Din definiţie, dacă P este un drum de creştere, atunci $r(P) > 0$ şi $v(x \otimes r(P)) > v(x)$. Urmează că dacă există un drum de creştere relativ la fluxul x, atunci x nu este un flux de valoare maximă.

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

Definiţie

Fie $R = (G, s, t, c)$ o reţea. O **secţiune** *în R este o partiţie (S, T) a lui $V(G)$ cu $s \in S$ şi $t \in T$.* **Capacitatea secţiunii** *(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 *

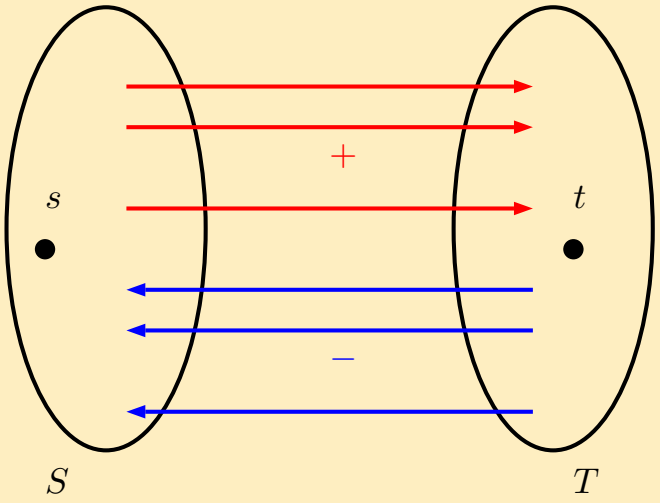
Lema 2

Dacă x este un flux în $R = (G, s, t, c)$ şi (S, T) este o secţiune în această reţea, atunci

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

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

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



Demonstraţie.

$$v(x) = \left(\sum_j x_{sj} - \sum_j x_{js} \right) + 0 =$$

$$= \left(\sum_j x_{sj} - \sum_j x_{js} \right) + \sum_{i \in S, i \neq s} \left(\sum_j x_{ij} - \sum_j x_{ji} \right) =$$

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

Demonstraţie (continuare).

$$= \sum_{i \in S} \left(\sum_j x_{ij} - \sum_j x_{ji} \right) = \sum_{i \in S} \sum_j (x_{ij} - x_{ji}) =$$

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

$$= 0 + \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}). \ \square$$

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

Lema 3

Dacă x este un flux în $R = (G, s, t, c)$ şi (S, T) este o secţiune în această reţea, atunci

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

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

Demonstraţie. Din Lema 2

$$v(x) = \sum_{i \in S} \sum_{j \in T} (x_{ij} - x_{ji}) \overset{x_{ij} \leqslant c_{ij}}{\leqslant} \sum_{i \in S} \sum_{j \in T} (c_{ij} - x_{ji}) \overset{x_{ji} \geqslant 0}{\leqslant}$$

$$\overset{x_{ji} \geqslant 0}{\leqslant} \sum_{i \in S} \sum_{j \in T} c_{ij} = c(S, T). \square$$

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

Remarcă

Dacă $\overline{x}$ este un flux în R şi $(\overline{S}, \overline{T})$ este o secţiune astfel încât $v(\overline{x}) = c(\overline{S}, \overline{T})$, atunci, $\forall x$ flux în R, avem $v(x) \leqslant c(\overline{S}, \overline{T}) = v(\overline{x})$, i. e., $\overline{x}$ este un flux de valoare maximă în R.

Similar, $\forall (S, T)$ secţiune în R, avem $c(S, T) \geqslant v(\overline{x}) = c(\overline{S}, \overline{T})$, adică, $(\overline{S}, \overline{T})$ este o secţiune 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

Teorema 1

Un flux x este un flux de valoare maximă dacă şi numai dacă nu există un drum de creştere relativ la x în R.

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

Demonstraţie. "$\Rightarrow$" Dacă P este un drum de creştere 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 creştere relativ la x în R. Fie

$$S = \{i \ : \ i \in V \text{ şi } \exists P \text{ un } A\text{-drum în } R \text{ de la } s \text{ la } i\}.$$

Evident $s \in S$ (există un A-drum de lungime 0 de la s la s) şi $t \notin S$ (nu există vreun A-drum de la s la t). Astfel, luând $T = V \setminus S$, (S, T) este o secţiune în R.

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

Demonstraţie (continuare). $\forall i \in S$ şi $\forall j \in T$ avem:

- dacă $ij \in E$, atunci $x_{ij} = c_{ij}$ şi
- 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 $j \in S$ - contradicţie). 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 *

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 capacităţile 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 -

Demonstraţie. Considerăm următorul algoritm:

$x^0 \leftarrow 0$; $i \leftarrow 0$;
while ($\exists P_i$ un drum de creştere 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 definiţia lui $r(P_i)$, dacă toate capacităţile sunt întregi şi x^i este întreg, atunci $r(P_i)$ este întreg şi astfel x^{i+1} este întreg).

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

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

Demonstraţie (continuare). Mai mult, în fiecare iteraţie while, valoarea fluxului curent creşte (cu cel puţin 1), astfel algoritmul se opreşte în cel mult $c(\{s\}, V \setminus \{s\}) \in \mathbb{Z}_+$ iteraţii while.
Nu mai există drumuri de creştere relativ la fluxul final, astfel, din Teorema 1, fluxul este de valoare maximă. $\square$

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

Remarcă

Algoritmul de mai sus se opreşte şi când toate capacităţile sunt numere raţionale.

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

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 – secţiune minimă) Valoarea maximă a unui flux în reţeaua $R = (G, s, t, c)$ este egală cu capacitatea minimă a unei secţiuni în R.

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

Linia demonstraţiei. Dacă descriem un algoritm care, plecând cu un flux iniţial x^0 (e.g., $x^0 = 0$), construieşte într-un număr finit de paşi un flux x fără drumuri de creştere, atunci secţiunea considerată în demonstraţia Teoremei 1 satisface, împreună cu x, cerinţa teoremei. Pentru capacităţi raţionale, algoritmul considerat în demonstraţia Teoremei 2 satisface această condiţie. Pentru capacităţi arbitrare reale vom prezenta mai târziu un astfel de algoritm datorat lui **Edmonds şi Karp (1972)**.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 demonstraţie scurtă a teoremei de mai sus constă în a arăta că există un flux de valoare maximă şi de a aplica construcţia din demonstraţia Teoremei 1. Un flux de valoare maximă există întotdeauna observând că acesta este soluţia optimă a unei probleme de programare liniară (peste un politop nevid).
- Importanţa algoritmică a Teoremei 3 este dată de faptul că mulţimea tuturor secţiunilor dintr-o reţea este finită, pe când mulţimea tuturor fluxurilor este infinită.

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

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

Algoritmul întreţine o etichetare a nodurilor din reţea pentru a determina drumuri de creştere relativ la fluxul curent x. Când nu mai există drumuri de creştere, fluxul x este de valoare maximă.
Fie $R = (G = (V, E), s, t, c)$ o reţea şi 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, $e_2 \in \{direct, invers\}$ reprezintă direcţia arcului ij, iar $e_3 = r(P)$.

Iniţial s este etichetat $(\cdot, \cdot, \infty)$. Celelalte noduri primesc eventual etichete prin vizitarea (scanarea) nodurilor deja etichetate:

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

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 ($ij \in E$ şi $x_{ij} < c_{ij}$) **then**
    etichetează j cu $e = (i, direct, \min\{e_3[i], c_{ij} - x_{ij}\})$;
  end if
  if ($ji \in E$ şi $x_{ji} > 0$) **then**
    etichetează j cu $e = (i, invers, \min\{e_3[i], x_{ji}\})$;
  end if
end for

Semnificaţia componentelor etichetelor este întreţinută de procedura scan.

Când, în procedura scan, nodul t este etichetat, se poate detecta un drum de creştere P, relativ la fluxul curent x, astfel:

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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, şi modificarea $x \otimes r(P)$ se poate face în timpul acestei explorări folosind a doua componentă a etichetelor.

Pentru noul flux, se porneşte cu o nouă etichetare (de la s).

Dacă toate noduri etichetate au fost scanate şi nodul t nu a primit etichetă, urmează că fluxul curent nu admite drumuri de creştere, deci este de valoare maximă. Dacă S este mulţimea nodurilor etichetate şi $T = V \setminus S$, atunci (S, T) este o secţiune 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 *

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

porneşte cu un flux iniţial $x = (x_{ij})$ (e. g., $x = 0$)
$e(s) \leftarrow (\cdot, \cdot, \infty)$;
while ($\exists$ noduri etichetate şi nescanate) **do**
  alege i un nod etichetat şi nescanat;
  scan(i);
  if (t a fost etichetat) **then**
    modifică fluxul pe drum dat de etichete;
    şterge toate etichetele; $e(s) \leftarrow (\cdot, \cdot, \infty)$;
  end if
end while
$S \leftarrow \{i : i \in V, i$ este etichetat$\}$;
$T \leftarrow V \setminus S$;

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

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

Complexitatea timp: Fiecare creştere a fluxului curent necesită cel mult $2m$ ($m = |E|$) inspecţii ale arcelor pentru etichetarea altor noduri. Dacă toate capacităţile sunt intregi, sunt necesare cel mult v creşteri (v fiind valoarea fluxului maxim). Astfel algoritmul are complexitatea timp $\mathcal{O}(mv)$.

Dacă U este un majorant al tuturor capacităţilor pe arce, atunci $v \leqslant (n-1)U$ (acesta este un majorant al secţiunii $(\{s\}, V \setminus \{s\})$), deci algoritmul are complexitatea timp $\mathcal{O}(nmU)$.

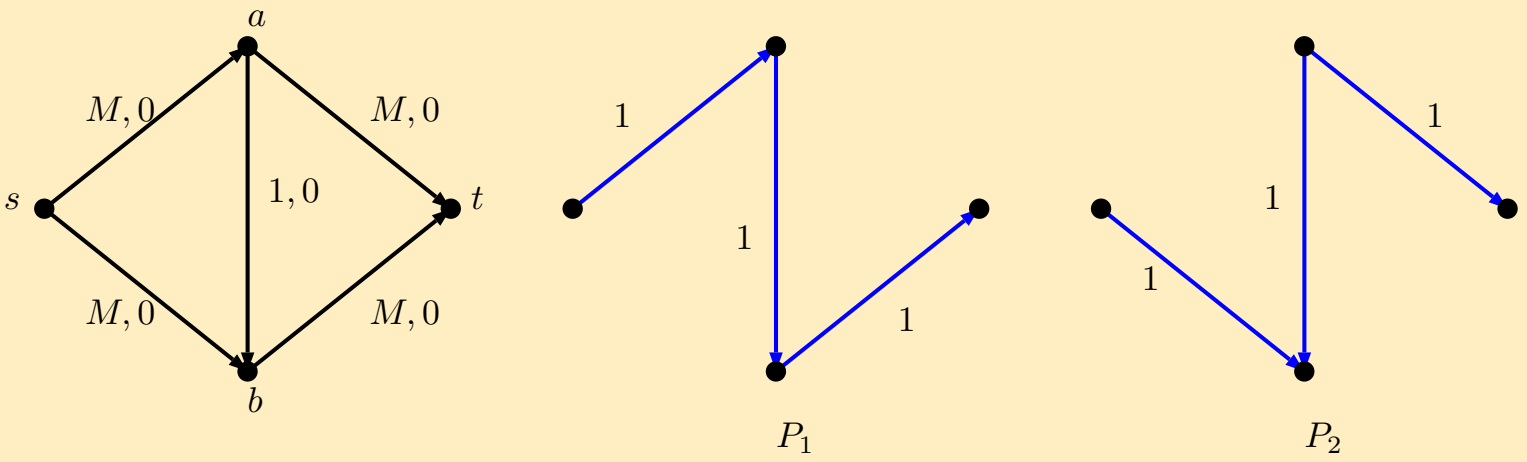
Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Remarcă

E posibil ca algoritmul să nu se termine pentru capacităţi iraţionale. Această situaţie nu apare în implementările practice, dar neajunsul acestei descrieri a algoritmului este dat de faptul că numărul de creşteri ale fluxului curent depinde de capacităţi (şi nu de dimensiunile reţelei).

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



Dacă operaţia alege din algoritmul de mai sus determină drumurile de creştere $P_1, P_2, P_1, P_2, \ldots,$, unde $P_1 = (s, sa, a, ab, b, bt, t)$ şi $P_2 = (s, sb, b, ba, a, at, t)$, atunci fiecare creştere a fluxului se face cu 1, şi algoritmul necesită $2M$ creşteri, ceea ce este prea mult.

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

Aceste deficienţe 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

Definiţie

Un **cel mai scurt drum de creştere** *relativ la fluxul x în R este un drum de creştere de lungime minimă printre toate drumurile de creştere relativ la x.*

Fie x un flux în reţeaua R. Fie x^k $(k \geqslant 1)$ secvenţa de fluxuri:

$x^1 \leftarrow x$;

$x^{k+1} \leftarrow x^k \otimes r(P_k)$, P_k un cel mai scurt drum de creştere relativ la x^k;

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

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

$$\sigma_i^k = \text{ lungimea minimă a unui } A\text{-drum de la } s \text{ la } i \text{ relativ la } x^k.$$

$$\tau_i^k = \text{ lungimea minimă a unui } A\text{-drum de la } i \text{ la } t \text{ relativ la } x^k.$$

Lema 4

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

$$\sigma_i^{k+1} \geqslant \sigma_i^k \text{ şi } \tau_i^{k+1} \geqslant \tau_i^k.$$

Demonstraţie. Omisă.

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

Teorema 4

(Edmonds & Karp, 1972) Dacă $x = x^1$ este un flux arbitrar în reţeaua R, atunci secvenţa de fluxuri $x^1, x^2, \ldots,$ obţinută din x^1 prin creşteri succesive cu drumuri de creştere de lungime minimă, are cel mult $mn/2$ termeni (în cel mult $mn/2$ creşteri succesive se obţine un flux x cu proprietatea că nu există drum de creştere relativ la x).

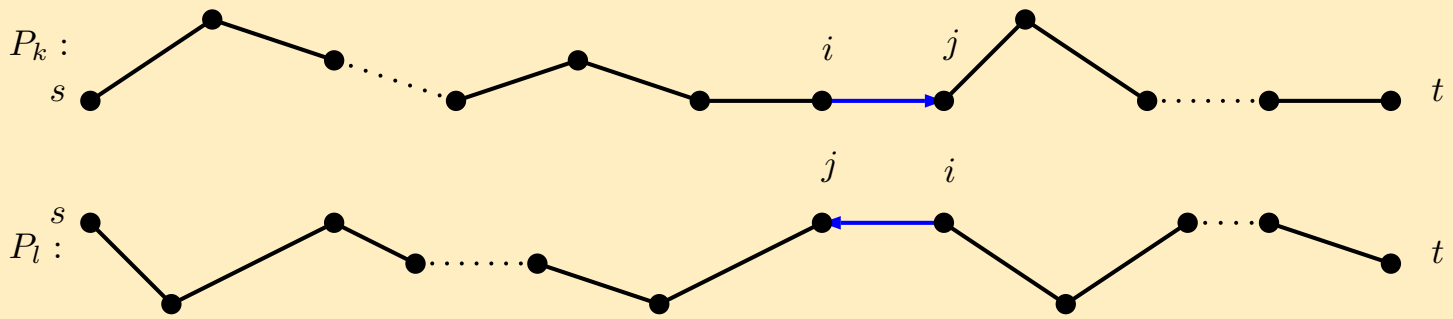
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie. Dacă P este un drum de creştere relativ la un flux x în R, cu capacitatea reziduală $r(P)$, un arc critic în P este un arc $e \in E(P)$ cu $r(e) = r(P)$. În $x \otimes r(P)$, fluxul de pe arce critice devine fie egal cu capacitatea (pe arcele directe), sau nul (pe arcele inverse). Fie ij un arc critic de pe un cel mai scurt drum de creştere P_k relativ la x^k. Lungimea lui P_k este $\sigma_i^k + \tau_i^k = \sigma_j^k + \tau_j^k$

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

Deoarece ij este a critic în P_k , in x^{k+1} nu mai poate fi utilizat în aceeaşi direcţie ca în P_k. Fie P_l (cu $l > k$) primul cel mai scurt drum de creştere relativ la x^l în care fluxul de pe arcul ij va fi modificat, (când arcul va fi folosit în direcţie inversă decât în P_k). Avem două cazuri:

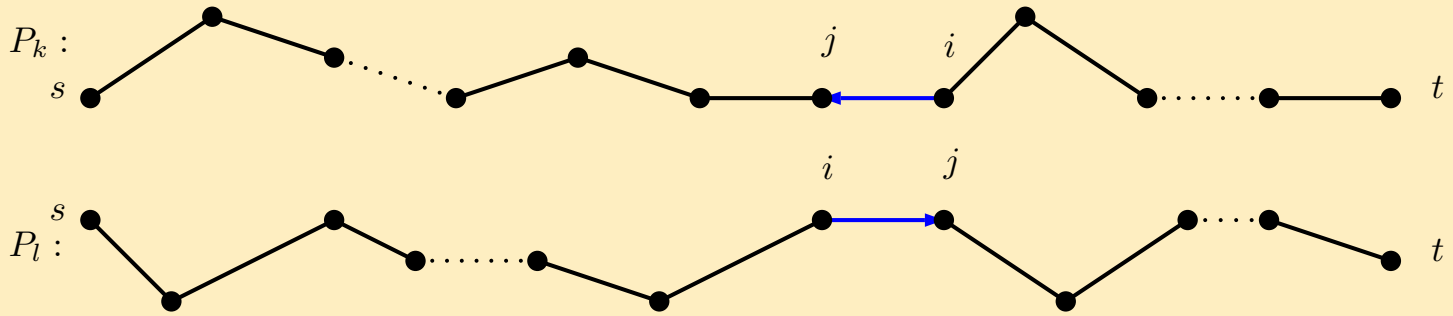
ij este un arc direct în P_k. Atunci $\sigma_j^k = \sigma_i^k + 1$; în P_l ij va fi arc invers, deci $\sigma_i^l = \sigma_j^l + 1$.



Urmează că $\sigma_i^l + \tau_i^l = \sigma_j^l + 1 + \tau_i^l \geqslant \sigma_j^k + 1 + \tau_i^k = \sigma_i^k + \tau_i^k + 2$ (din Lema 4). Am obţinut că $length(P_l) \geqslant length(P_k) + 2$.

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

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



Urmează că $\sigma_j^l + \tau_j^l = \sigma_i^l + 1 + \tau_j^l \geqslant \sigma_i^k + 1 + \tau_j^k = \sigma_j^k + \tau_j^k + 2$. Obţinem că $length(P_l) \geqslant length(P_k) + 2$.

Astfel orice cel mai scurt drum de creştere pe care arcul ij este critic are lungimea cu cel puţin 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 creştere).

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

Orice drum de creştere are cel puţin un arc critic. Astfel în secvenţa (P_k) avem cel mult $mn/2$ drumuri de creştere cele mai scurte.

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

Corolar

În orice reţea există un flux x cu proprietatea că nu există drumuri de creştere relativ la x.

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

Remarci

- Demonstraţia Teoremei 4 este acum încheiată.
- Singura modificare din **Algoritmul lui Ford & Fulkerson** este în alegerea nodului etichetat care va fi scanat: se foloseşte regula "primul etichetat-primul scanat" adică, se întreţine o coadă a nodurilor etichetate (iniţializată cu s, la fiecare început de creştere).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 obţine în $\mathcal{O}(m^2 n)$ time.

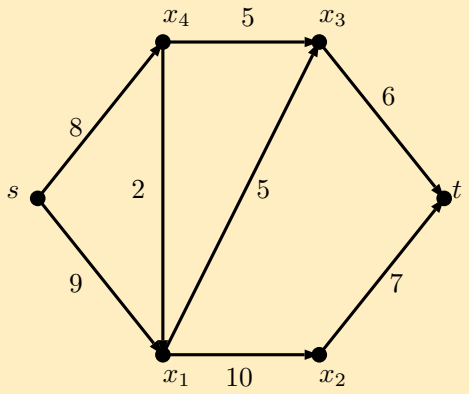
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Demonstraţie. Există $\mathcal{O}(mn)$ creşteri de flux (din Teorema 4), fiecare de complexitate $\mathcal{O}(m)$. $\square$

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

Exerciţii pentru seminarul din săptămâna viitoare

Exerciţiul 1. Considerăm reţeaua de transport de mai jos:



Determinaţi un flux maxim în reţeaua de mai jos utilizând algoritmul lui Edmonds-Karp pentru următoarele ordonări ale listelor de adiacenţă:

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

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

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

Exerciţiul 2. Considerăm o reţea $R = (G, s, t, ; c)$, unde $G = (V, E)$ are n noduri şi m arce, şi funcţia de capacitate are doar valori întregi ($c : E \to \mathbb{Z}_+$). Fie $C = \max_{e \in E} c(e)$.

(a) Arătaţi că valoarea maximă a unui flux în R este cel mult $m \cdot C$.

(b) Arătaţi că, pentru fiecare flux x din R şi pentru fiecare $K \in \mathbb{Z}_+$, putem găsi un drum de creştere P cu capacitatea reziduală, $r(P)$, cel puţin K - dacă un asemenea drum există - în $\mathcal{O}(m)$.

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

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

Exerciţiul 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$(K \geqslant 1)$ **{**

while$(x$ **are un drum de creştere** P **cu** $r(P) \geqslant K)$

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

$K \leftarrow K/2$;

}

return x;

}

Exerciţii pentru seminarul din săptămâna viitoare

Exerciţiul 2 (continuare).

1. Arătaţi că procedura **SC-MAX-FLOW**(R) returnează un flux de valoare maximă x în R.
2. Arătaţi că, după fiecare iteraţie while exterioară, valoarea maximă a unui flux în R este cel mult $v(x) + m \cdot K$.
3. Arătaţi că, $\forall\, K \in \mathbb{Z}_+$, există cel mult $2m$ iteraţii while interioare. În consecinţă procedura are complexitatea timp $\mathcal{O}(m^2 \log C)$.

Exerciţiul 3. Digraful $G = (V, E)$ reprezintă topologia unei reţele de procesoare.Fiecărui procesor, $v \in V$, îi cunoaştem încărcarea, $load(v) \in \mathbb{R}_+$. Folosind un flux maxim într-o anumită reţea determinaţi o strategie statică de echilibrare a încărcaării (static load balancing strategy) în G: indicaţi pentru fiecare procesor încărcarea ce trebuie trimisă şi cărui procesor aşa încât toate procesoarele să aibă aceeaşi încărcare.

Exerciţii 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

Exerciţiul 4. Fie $R = (G, s, t, c)$ o reţea şi $(S_i, T_i)(i = \overline{1,2})$ două secţiuni de capacitate minimă în R. Arătaţi că $(S_1 \cup S_2, T_1 \cap T_2)$ şi $(S_1 \cap S_2, T_1 \cup T_2)$ sunt de asemenenea secţiuni de capacitate minimă.

Exerciţiul 5. Fie $R = (G = (V, E), s, t, c)$ o reţea şi $c : E \to \mathbb{Z}_+$, $n = |V|$, $m = |E|$.

(a) Descrieţi un algoritm de complexitate timp $\mathcal{O}(n + m)$ pentru a determina o secţiune de capacitate minimă în R, având la îndemână un flux de valoare maximă x^*.

(b) Folosind un algoritm de flux maxim pentru o anumită funcţie de capacitate, arătaţi că se poate găsi o secţiune de capacitate minimă în R, (S_0, T_0), cu număr minim de arce.

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

Exerciţiul 6. Fie $G = (S, T; E)$ un graf bipartit. Demonstraţi teorema lui Hall (*există un cuplaj în G care saturează toate nodurile din S dacă şi numai dacă (H)* $\forall\, A \subseteq S, |N_G(A)| \geqslant |A|$) folosind teorema flux maxim - secţiune minimă pe o reţea particulară.

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

Exerciţiul 7. Fie $R = (G, s, t, c)$ o reţea care are toate capacităţile întregi pozitive şi pare. Arătaţi că există un flux maxim cu toate valorile întregi pozitive şi 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 *

Exerciţiul 8. Fie $R = (G, s, t, c)$ o reţea astfel ca $st, ts \notin E(G)$. Avem şi o funcţie majorant $m : E(G) \to \mathbb{R}_+$, $m(e) \leqslant c(e), \forall\, e \in E(G)$. Un flux φ în R se numeşte **flux legal în** R dacă $x(e) \geqslant m(e), \forall\, e \in E(G)$.

(a) Arătaţi că, pentru orice *flux legal* φ şi orice st-secţiune (S, T) avem

$$v(\varphi) \leqslant \sum_{i \in S, j \in T, ij \in E(G)} c(ij) - \sum_{i \in S, j \in T, ji \in E(G)} m(ji)$$

(b) Pornind de la R construim o reţea $\overline{R} = (\overline{G}, \overline{s}, \overline{t}, \overline{c})$, unde

- $V(\overline{G}) = V(G) \cup \{\overline{s}, \overline{t}\}$ (acestea sunt noduri noi);
- $E(\overline{G}) = E(G) \cup \{st, ts\} \cup \{\overline{s}v, v\overline{t} \,:\, v \in V(G)\}$;
- pentru orice $v \in V(G)$, $\overline{c}(\overline{s}v) = \sum_{uv \in E(G)} m(uv)$ şi $\overline{c}(v\overline{t}) = \sum_{vu \in E(G)} m(vu)$;
- $\overline{c}(st) = \overline{c}(ts) = \infty$ şi pentru orice $ij \in E(G)$, $\overline{c}(ij) = c(ij) - m(ij)$.

Exerciţii 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 *

Cuprins

1. Fluxuri în reţele
   - Prefluxuri
     - Schema generală a unui algoritm de tip preflux
     - Algoritmul lui Ahuja & Orlin
   - Aplicaţii combinatoriale
     - Cuplaje în grafuri bipartite
     - Recunoaşterea secvenţelor digrafice
     - Conexiunea pe muchii
     - Conexiunea pe noduri
2. Exerciţii pentru seminarul din săptămâna viitoare

C. Croitoru - Graph Algorithms *

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

Definiţie

Un **preflux** în $R = (G, s, t, c)$ este o funcţie $x : E(G) \to \mathbb{R}$ astfel încât

(i) $0 \leqslant x_{ij} \leqslant c_{ij}, \forall ij \in E$;

(ii) $\forall i \neq s,\ e_i = \sum_{ji \in E} x_{ji} - \sum_{ij \in E} x_{ij} \geqslant 0.$

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

e_i (pentru $i \in V \setminus \{s, t\}$) este numit **excesul** din nodul i. Dacă $i \in V \setminus \{s, t\}$ şi $e_i > 0$, 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 iniţial în R este transformat prin modificarea fluxurilor de pe arce **într-un flux** cu proprietatea că **nu există drumuri de creştere în R relativ la el**.

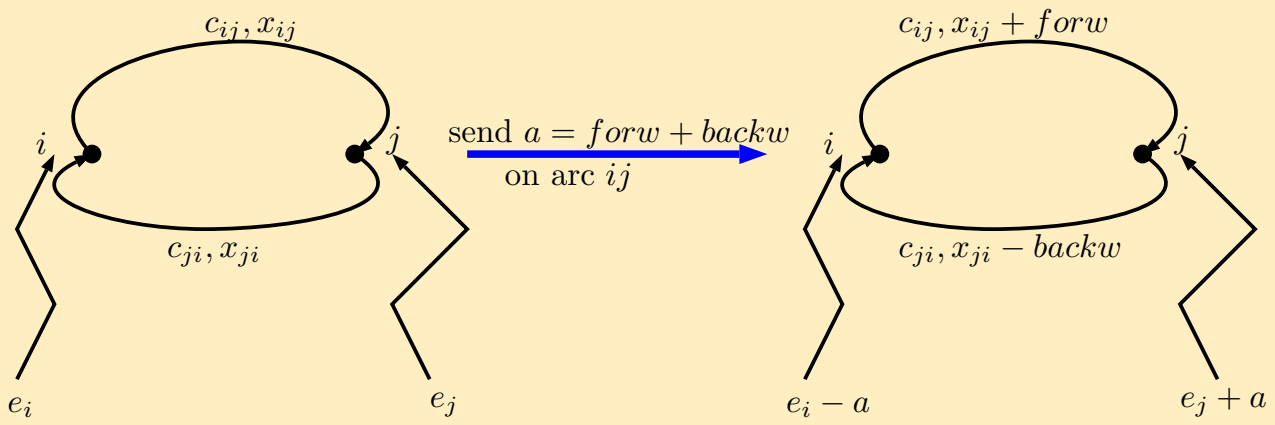
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

G este reprezentat folosind liste de adiacenţă. Vom presupune că dacă $ij \in E$, atunci $ji \in E$ de asemeni (altfel, adăugăm arcul ji de capacitate 0). Astfel, G devine digraf simetric.

Dacă x este un preflux în R şi $ij \in E$, 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 şi ji).



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

În cele de urmează, **a trimite** flux de la i la j înseamnă creşterea 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 **funcţie distanţă** în R relativ la prefluxul x este o funcţie $d : V \to \mathbb{Z}_+$ a. î.

(D_1) $d(t) = 0$,

(D_2) $\forall ij \in E, r_{ij} > 0 \Rightarrow d(i) \leqslant d(j) + 1$.

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

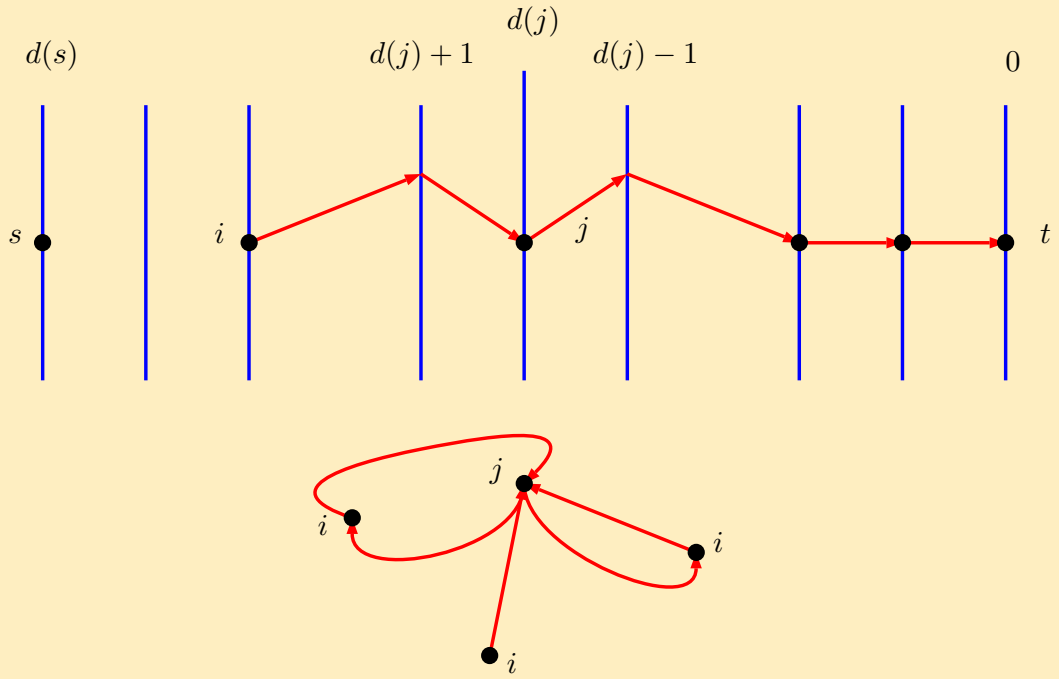
Remarci

- Dacă P este un A-drum relativ la prefluxul x în R de la i la t, atunci $d(i) \leqslant length(P)$ (arcele lui P au capacităţi reziduale pozitive şi apoi se aplică (D_2) în mod repetat). Urmează că (τ_i este lungimea minimă a unui A-drum de la i la t): $d(i) \leqslant \tau_i$.

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

Remarci

- Ca de obicei, notăm cu $A(i)$ lista de adiacenţă a nodului i.



Definiţie

Fie x un preflux în R şi d o funcţie distanţă relativ la x. Un arc $ij \in E$ este numit **admisibil** *dacă $r_{ij} > 0$ şi $d(i) = d(j) + 1$.*

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

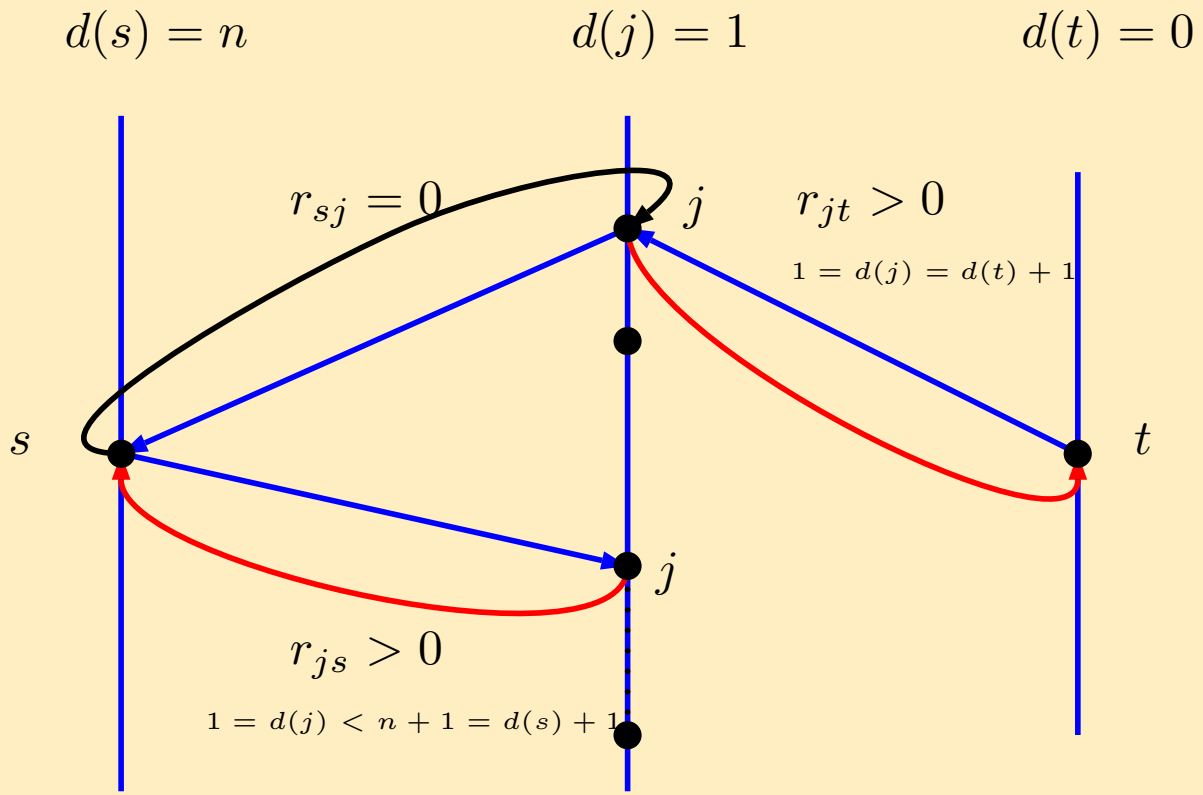
Dacă R este o reţea, considerăm următoarea procedură de iniţializare, care construieşte în $\mathcal{O}(m)$ un preflux x şi o funcţie distanţă d relativ la x.

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 $(i \in V - \backslash\{s, t\})$ **do**
  $d[i] \leftarrow 1;$

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

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

Se observă că după execuţia acestei proceduri, avem $r_{sj} = 0, \forall sj \in A(s)$.
Astfel condiţia (D_2) nu este not afectată de $d(s) = n$.
Pentru toate arcele ij, (D_2) este îndeplinită:



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

Alegerea $d(s) = n$ are semnificaţia: 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 puţin $d(s) = n$, 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" $\delta = \min\{e_i, r_{ij}\}$ (unităţi 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
$d[i] \leftarrow \min\{d[j] + 1 \ : \ ij \in A(i)$ şi $r_{ij} > 0\}$;

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 funcţie distanţă relativ la prefluxul x" este un invariant al algoritmului de mai sus. La fiecare apel $relabel(i)$, $d(i)$ creşte.

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

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

Demonstraţie. Am demonstrat deja că procedura de iniţializare construieşte un preflux x şi o funcţie distanţă d relativ la x. Arătăm că dacă perechea (d, x) satisface (D_1) şi (D_2) înaintea unei iteraţii while, atunci după această iteraţie cele două condiţii sunt de asemeni satisfăcute.
Avem două cazuri, în funcţie de care dintre procedurile push sau relabel este executată în iteraţia while curentă:
este apelată $push(i)$: singura pereche care poate viola (D_2) este $d(i)$ şi $d(j)$. Deoarece ij este admisibil, la apelul $push(i)$ avem $d(i) = d(j)+1$. După execuţia $push(i)$, arcul ji poate avea $r_{ji} > 0$ (fără a fi astfel înaintea apelului), dar condiţia $d(j) \leqslant d(i) + 1$ este evident satisfăcută.
este apelată $relabel(i)$: actualizarea lui $d(i)$ este astfel încât (D_2) este satisfăcută pentru fiecare arc ij cu $r_{ij} > 0$. Deoarece $relabel(i)$ este apelată când $d(i) < d(j) + 1$, $\forall ij$ cu $r_{ij} > 0$, urmează că, după apel, $d(i)$ creşte (cu cel puţin 1). $\square$

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

Pentru a arăta că algoritmul se opreşte, este necesar să arătăm că (în timpul execuţiei) dacă un nod i este activ atunci în lista lui de adiacenţă, $A(i)$, există cel puţin 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 $i_0 s$ A-drum relativ la x.

Demonstraţie. Dacă x este un preflux în R, atunci x poate fi descompus în $x = x^1 + x^2 + \cdots + x^p$, unde fiecare x^k are proprietatea că mulţimea $A^k = \{ij \ : \ ij \in E, x^k_{ij} \neq 0\}$ este

(a) mulţimea arcelor unui drum de la s la t, sau

(b) mulţimea de arce ale unui drum de la s la un nod activ, sau

(c) mulţimea arcelor unui circuit.

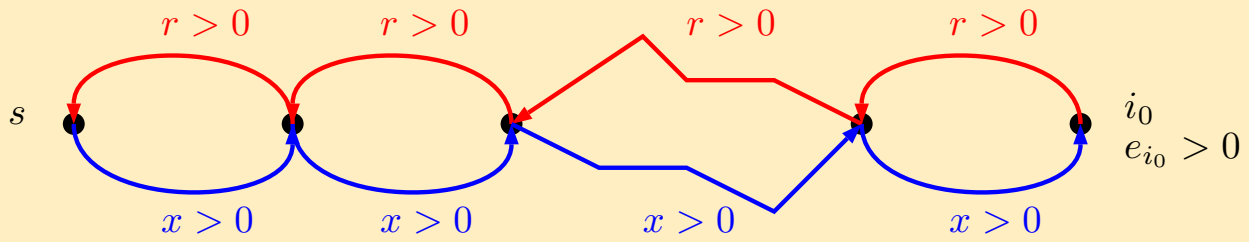
Prefluxuri - Schema generală a unui algoritm de tip preflux

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) şi (c), x^k este un flux. (Demonstraţia este algoritmică: construim mai întâi mulţimile de tipul (a), apoi pe cele de tipul (c) şi (b); la fiecare pas, căutăm inversul unui drum de tip (a) sau (c) (sau (b)); prefluxul obţinut este scăzut din cel curent; deoarece excesele nodurilor sunt nenegative, construcţia poate fi realizată, ori de câte ori prefluxul curent este nenul; construcţia este finită, deoarece numărul de arce pe care fluxul curent este 0, creşte la fiecare pas.)
Deoarece i_0 este un nod activ în R relativ la x, cazul (b) va aparea pentru nodul i_0 (cazurile (a) şi (c) nu afectează excesul din nodul i_0). Arcele inverse de pe acest drum au capacitatea reziduală pozitivă (vezi imaginea de mai jos), astfel ele formează A-drumul cerut. $\square$

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

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



Corolarul 1

$\forall i \in V,\ d(i) < 2n.$

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

Demonstraţie. 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 $length(P) \leqslant n - 1$. Din (D_2), urmează că, după reetichetare , $d(i) \leqslant d(s) + n - 1 = 2n - 1$ ($d(s) = n$ nu este modificat vreodată). $\square$

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

Corolarul 2

Numărul total de apeluri ale procedurii **relabel** nu este mai mare decât $2n^2$.

Demonstraţie. 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 şi distanţa iniţială d). $\square$

Corolarul 3

Numărul total de pompări saturate nu este mai mare decât nm.

Demonstraţie. 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 $d'(j) = d'(i)+1 \geqslant d(i)+1 = d(j)+2$.

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

Demonstraţie (continuare). Astfel o modificare a acestui flux pe arcul ij nu va apărea până când $d(j)$ nu va creşte cu 2. Urmează că un arc nu poate deveni saturat de mai mult de n ori şi (deoarece numărul total de arce este m), nu există mai mult de nm pompări saturate. □

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

Lema 3

(Goldberg şi 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

Demonstraţie. 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.

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

Demonstraţie. Din Lemele 1 şi 3 şi din Corolarul 3 al Lemei 2, algoritmul se opreşte în cel mult $2n^2m$ iteraţii while. Deoarece $d(s) = n$ nu se modifică niciodată, urmează că nu există vreun drum de creştere relativ la fluxul x obţinut, astfel x este de valoare maximă: dacă P ar fi un drum de creştere (în digraful suport al lui G), atunci înlocuind de-a lungul lui P fiecare arc invers cu simetricul său se obţine un A-drum de la s la t, deci $n = d(s) \leqslant d(t) + n - 1 = n - 1$. $\square$

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

În locul demonstraţiei 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 *

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

Să presupunem că $c_{ij} \in \mathbb{N}$ şi $U = \max_{ij \in E}(1 + c_{ij})$. Fie $K = \lceil \log_2 U \rceil$.

Ideea algoritmului: Avem $K + 1$ etape. În fiecare etapă p, cu p luând succesiv valorile $K, K-1, \ldots, 1, 0$, următoarele două condiţii sunt îndeplinite:

(a) la începutul etapei p, $e_i \leqslant 2^p$, $\forall i \in V \setminus \{s, t\}$.

(b) în timpul etapei p, procedurile **push-relabel** sunt folosite pentru a elimina nodurile active, i, cu $e_i \in \{2^{p-1} + 1, \ldots 2^p\}$.

Din definiţia lui K, în prima etapă ($p = K$), condiţia (a) este îndeplinită, şi dacă condiţia (b) va fi menţinută de-a lungul execuţiei algoritmului, după $K + 1$ etape, dacă se menţine 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.

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

Pentru a menţine (b) de-a lungul execuţiei 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 $i_1, i_2, \ldots, i_{l(p)}$ cu excese $e_i > 2^{p-1}$, ordonate crescător după d (aceasta poate fi făcută cu hash-sorting în timpul $O(n)$, deoarece $d(i) \in \{1, 2, \ldots, 2n-1\}$).
- nodul activ selectat pentru **push-relabel** în timpul etapei p va fi **primul nod din** $L(p)$. Urmează că, dacă se face o pompare (push) pe un arc admisibil ij, atunci $e_i > 2^{p-1}$ şi $e_j \leqslant 2^{p-1}$ (deoarece $d(j) = d(i) - 1$ şi 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 $2^p - e_j \geqslant 2^{p-1}$) urmează că o **pompare nesaturată** trimite cel puţin 2^{p-1} unităţi de flux.

După apelul *push*(i) excesul din nodul j (singurul pentru care excesul poate creşte) va fi $e_j + \min(e_i, r_{ij}, 2^p - e_j) \leqslant e_j + 2^p - e_j \leqslant 2^p$, deci condiţia (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 listele de adiacenţă $A(i)$ după cum urmează:

- fiecare element din listă conţine: nodul j, x_{ij}, r_{ij}, un pointer către elementul corespunzător lui i din lista de adiacenţă $A(j)$ şi 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**.

initialization; $K \leftarrow \lceil \log_2 U \rceil$; $\Delta \leftarrow 2^{K+1}$;
for $(p = \overline{K, 0})$ **do**
  construieşte $L(p)$; $\Delta \leftarrow \Delta/2$;
  while $(L(p) \neq \varnothing)$ **do**
    fie i primul nod din $L(p)$;
    caută în $A(i)$ primul arc admisibil sau până se ajunge la sfârşit;
    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" δ unităţi de flux de la i la j;
      if $(e_i \leqslant \Delta/2)$ **then**
        şterge i din $L(p)$;
      if $(e_j > \Delta/2)$ **then**
        adaugă j la începutul listei $L(p)$;
    else
      calculează $d[i] = \min\{d[j] + 1 \ : \ ij \in A(i)$ şi $r_{ij} > 0\}$
      repoziţionează i în $L(p)$;
      setează poziţia curentă (a pointerului) la începutul listei $A(i)$;

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

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 $\mathcal{O}(n^2 \log U)$.

Demonstraţie. Fie

$$F(p) = \sum_{i \in V, i \neq s,t} \frac{e_i \cdot d(i)}{2^p}.$$

La începutul etapei p, $F(p) < \sum_{i \in V} \frac{2^p \cdot (2n)}{2^p} = 2n^2$.

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, şi $d(i)$ creşte cu $h \geqslant 1$ unităţi. $F(p)$ va creşte cu cel mult h. Deoarece, $\forall i$, $d(i) < 2n$ urmează că $F(p)$ va creşte (până la finalul etapei p) cel mult până la $4n^2$.

Dacă, în etapa p, se execută *push*(i), atunci aceasta trimite $\delta \geqslant 2^{p-1}$ pe arcul admisibil ij cu $r_{ij} > 0$ şi $d(i) = d(j) + 1$. Astfel, după push, $F(p)$ va avea valoarea $F'(p) = F(p) - \dfrac{\delta \cdot d(i)}{2^p} + \dfrac{\delta \cdot d(j)}{2^p} = F(p) - \dfrac{\delta}{2^p} \leqslant F(p) - \dfrac{2^{p-1}}{2^p} = F(p) - 1/2$.

Această descreştere nu poate apărea de mai mult de $8n^2$ ori (deoarece $F(p)$ poate creşte cel mult până la $4n^2$ şi $F(p)$ este nenegativ). Evident, numărul pompărilor nesaturate este dominat de acest număr de descreşteri ale lui $F(p)$.

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

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

A. Determinarea unui cuplaj de cardinal maxim şi a unei mulţimi stabile de cardinal maxim într-un un graf bipartit.

Fie $G = (V_1, V_2; E)$ un graf bipartit cu n noduri şi m muchii.
Considerăm reţeaua $R = (G_1, s, t, c)$, unde

- $V(G_1) = \{s, t, \} \cup V_1 \cup V_2$;
- $E(G_1) = E_1 \cup E_2 \cup E_3$, cu

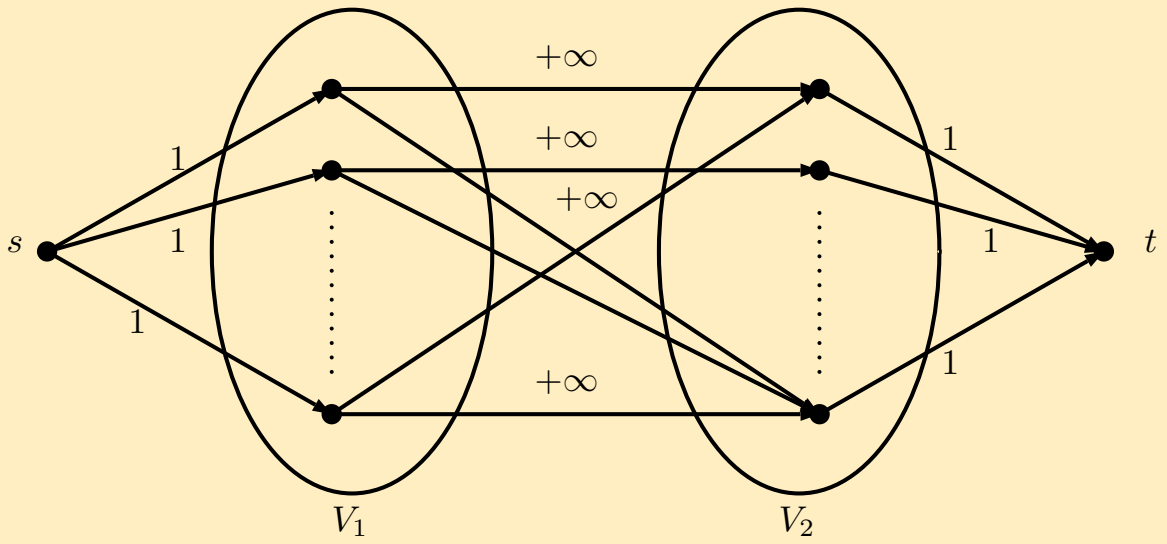
$$E_1 = \{sv_1 \ : \ v_1 \in V_1\}, E_2 = \{v_2 t \ : \ v_2 \in V_2\},$$

$$E_3 = \{v_1 v_2 \ : \ v_1 \in V_1, v_2 \in V_2\},$$

- $c : E(G_1) \to \mathbb{N}$ definită prin

$$c(e) = \begin{cases} 1, & \text{dacă } e \in E_1 \cup E_2 \\ \infty, & \text{dacă } e \in E_3 \end{cases}$$

G. Croitoru, Graph Algorithms * G. Croitoru, Graph Algorithms * G. Croitoru, Graph Algorithms



Dacă $x = (x_{ij})$ este un flux întreg în R, atunci mulţimea $\{ij : i \in V_1, j \in V_2$ şi $x_{ij} = 1\}$ coresponde unui cuplaj M^x în graful bipartit G, cu $|M^x| = v(x)$.

Reciproc, orice cuplaj $M \in \mathcal{M}_G$ corespunde unei mulţimi de arce neadiacente din G_1; dacă pe fiecare astfel de arc ij ($i \in V_1$, $j \in V_2$) considerăm $x_{ij}^M = 1$ şi $x_{si}^M = x_{jt}^M = 1$, şi adăugăm $x^M(e) = 0$ pe restul arcelor, atunci fluxul întreg x^M satisface $v(x^M) = |M|$.

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

Astfel, dacă rezolvăm problema fluxului maxim în R (începând cu fluxul nul), atunci obţinem în $\mathcal{O}(nm + n^2 \log n)$ un cuplaj de cardinal maxim în G.

Fie (S, T) secţiunea de capacitate minimă (obţinută în $\mathcal{O}(m)$ dintr-un flux maxim determinat). Din Teorema de flux maxim- secţiune minimă, $c(S, T) = \nu(G)$.

Deoarece $\nu(G) < \infty$, luând $S_i = S \cap V_i$ şi $T_i = T \cap V_i$ $(i = \overline{1,2})$, avem $|T_1| + |S_2| = \nu(G)$ şi $X = S_1 \cup T_2$ este o mulţime 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 mulţime 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 *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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. Recunoaşterea secvenţelor digrafice

Considerăm următoarea problemă:

Date $(d_i^+)_{i=\overline{1,n}}$ şi $(d_i^-)_{i=\overline{1,n}}$, există un digraf $G = (\{1, \ldots, n\}, E)$ astfel încât $d_G^+(i) = d_i^+$ şi $d_G^-(i) = d_i^-$, $\forall i = \overline{1, n}$?

Condiţii evident necesare pentru un răspuns afirmativ sunt:

$$d_i^+ \in \mathbb{N}, 0 \leqslant d_i^+ \leqslant n-1 \text{ şi } d_i^- \in \mathbb{N}, 0 \leqslant d_i^- \leqslant n-1, \forall i = \overline{1, n};$$

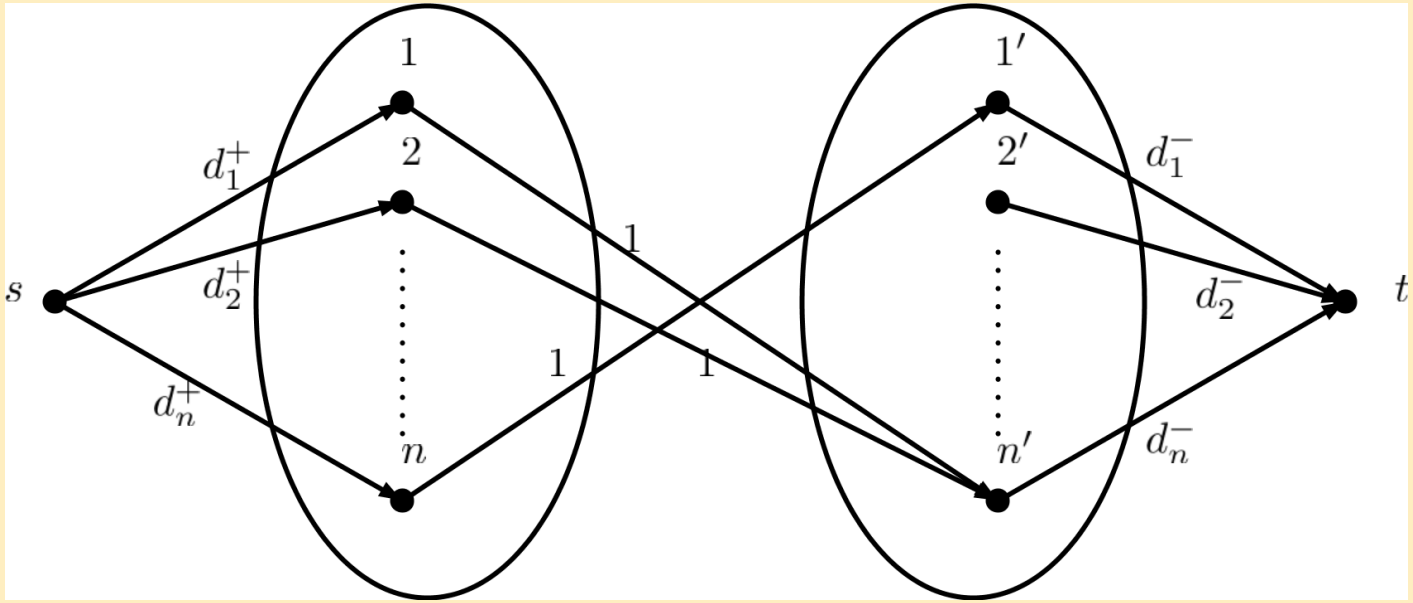
$$\sum_{i=1}^{n} d_i^+ = \sum_{i=1}^{n} d_i^- = m \text{ (unde } m = |E|).$$

În aceste ipoteze considerăm reţeaua bipartită $R = (G_1, s, t, c)$.

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

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

G_1 este obţinut din graful complet bipartit $K_{n,n}$ cu bipartiţia $(\{1,2,\ldots,n\},\{1',2',\ldots,n'\})$, prin ştergerea mulţimii de muchii $\{11',22',\ldots,nn'\}$ şi prin orientarea fiecărei muchii ij' $(\forall i \neq j \in \{1,2,\ldots,n\})$ de la i la j', şi prin adăugarea a două noi noduri s, t, şi a tuturor arcelor si, $i \in \{1,2,\ldots,n\}$ şi $j't$, $j \in \{1,2,\ldots,n\}$.
Funcţia de capacitate: $c(si) = d_i^+$, $c(j't) = d_j^-$, $c(ij') = 1$, $\forall i,j = \overline{1,n}$.



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

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 $x_{ij'} = 1$, şi in fiecare nod j' vor intra exact d_{i}^{-} arce, ij', pe care $x_{ij'} = 1$.

Digraful dorit, G, este construit luând $V(G) = \{1, 2, \ldots, n\}$ şi punând $ij \in E(G)$ dacă şi numai dacă $x_{ij'} = 1$.

Reciproc, dacă G există, atunci inversând construcţia anterioară vom obţine un flux întreg în R de valoare m (deci de valoare maximă).

Urmează că, recunoaşterea secvenţelor digrafice (şi 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 * C. Croitoru - Graph Algorithms *

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 mulţimi de muchii astfel încât nu există drum de la s la t în graful obţinut prin ştergerea 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

Demonstraţie. Fie G_1 digraful obţinut din G prin înlocuirea fiecărei muchii printr-o pereche de arce simetrice. Fie $c : E(G_1) \to \mathbb{N}$ o funcţie de capacitate definită prin $c(e) = 1, \forall e \in E(G_1)$.

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

Demonstraţie (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 $x^0_{ij} = 1$ 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 şi 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 şi în graful G.

Urmează că $v(x^0) = p_e(s, t)$, deoarece orice mulţime 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 *

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

Proof (continuare). Fie (S, T) o secţiune de capacitate minimă în R; avem $c(S, T) = v(x^0)$, din Teorema flux maxim-secţiune minimă. Pe de altă parte, $c(S, T)$ este numărul de arce cu o extremitate în S şi cealaltă în T (deoarece toate capacităţile arcelor sunt 1). Această mulţime de arce generează în G o mulţime de muchii de acelaşi cardinal astfel încât nu există vreun drum de la s to t în graful obţinut prin ştergerea ei din G.

Astfel am obţinut o mulţime de $c(S, T) = v(x^0) = p_e(s, t)$ muchii în G care deconectează pe s de t prin ştergerea lor din G. Urmează că $c_e(s, t) \leqslant p_e(s, t)$. Deoarece inegalitatea $c_e(s, t) \geqslant p_e(s, t)$ 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

$$\min_{s,t \in V(G), s \neq t} c_e(s, t) \ (*)$$

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 dacăe observăm că, pentru o pereche fixată (s, t), dacă (S, T) este o secţiune de capacitate minimă, atunci

$$\forall v \in S \text{ şi } \forall w \in T \; c_e(v, w) \leqslant c(S, T) \; (**)$$

În particular, dacă (s, t) este perechea pentru care minimul din $(*)$ se atinge, avem egalitate în $(**)$.
Dacă fixăm un nod $s_0 \in V$ şi rezolvăm cele $n-1$ probleme de flux maxim luând $t_0 \in V \setminus \{s_0\}$ vom obţine o pereche (s_0, t_0) cu $c(s_0, t_0) = \lambda(G)$ (t_0 nu va fi în aceeaşi clasă a bipartiţiei cu s_0 în (S, T)).
Concluzie: $\lambda(G)$ poate fi găsit în $\mathcal{O}(n \cdot (nm + n^2 c)) = \mathcal{O}(n^2 m)$.

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

C. Croitoru - 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 mulţimist-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

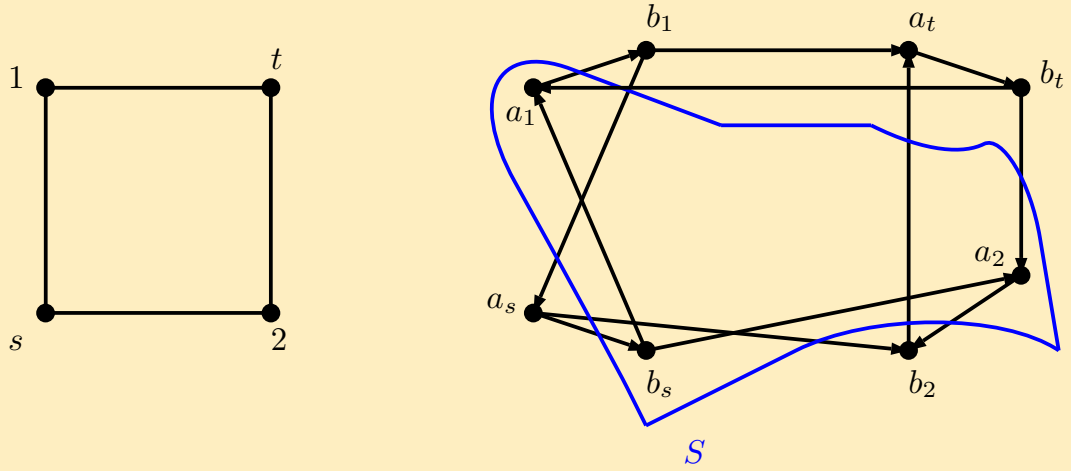
$$k(G) = \begin{cases} n - 1, & \text{dacă } G = K_n \\ \min\limits_{s,t \in V(G), s \neq t} c(s, t), & \text{dacă } G \neq K_n \end{cases} \quad (****)$$

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

Arătăm că egalitatea $(***)$ provine din Teorema flux maxim-secţiune minimă, aplicată unei reţele corespunzătoare.
Fie $G_1 = (V(G_1), E(G_1))$ digraful construit pornind de la G astfel:

- $\forall v \in V$, adăugăm $a_v, b_v \in V(G_1)$ şi $a_v b_v \in E(G_1)$;
- $\forall vw \in E$, adăugăm $b_v a_w, b_w a_v \in E(G_1)$.

Exemplu



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

Mai definim $c : E(G_1) \to \mathbb{N}$ prin

$$c(e) = \begin{cases} 1, & \text{dacă } e = a_v b_v \\ \infty, & \text{altfel} \end{cases}$$

Considerăm reţeaua $R = (G_1, b_s, a_t, c)$.
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 şi din nodurile $a_v (v \in V)$ pleacă exact câte un arc de capacitate 1.
Urmează că (din legea de conservare a fluxului) că $x^0_{ij} \in \{0, 1\}$, $\forall ij \in E(G_1)$. 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 corespond la $v(x^0)$ drumuri intern disjuncte din G.
Pe de altă parte, din orice mulţime 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 $v(x^0) = p(s, t)$.

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

Fie (S, T) o secţiune 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 $i = a_u$ şi $j = b_u$.

Astfel, secţiunea (S, T) coresponde unei mulţimi de noduri $A_0 \subseteq V$ astfel încât $c(S, T) = |A_0|$ şi A_0 este o mulţime st-separatoare.

Pe de altă parte, pentru orice mulţime st-separatoare, A, avem $|A| \geqslant p(s, t) = v(x^0)$. Deci

$$c(s, t) = |A_0| = c(S, T) = v(x^0) = p(s, t).$$

Demonstraţia 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.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 observaţie simplă ne oferă un algoritm mai eficient. Evident,

$$k(G) \leqslant \min_{v \in V} d_G(v) = \frac{1}{n}\left(n \cdot \min_{v \in V} d_G(v)\right) \leqslant \frac{1}{n}\sum_{v \in V} d_G(v) = \frac{2m}{n}.$$

Dacă A_0 este a mulţime separatoare în G cu $|A_0| = k(G)$, atunci $G \setminus A_0$ nu este conex şi există o partiţie of $V \setminus A_0$ (V', V'') astfel încât $\forall v' \in V'$ şi $\forall v'' \in V''$ 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 *

Aplicaţii combinatoriale - Conexiunea pe noduri

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

Exerciţii 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 -

Exerciţiul 1. Arătaţi că, utilizând un algoritm de flux maxim (într-o anumită reţea), se poate găsi, într-o matrice $0-1$, o mulţime de cardinal maxim de elemente egale cu 0, în care oricare două elemente nu se află pe aceeaşi linie sau pe aceeaşi coloană.

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

Exerciţiul 2. Fie S şi T mulţimi nevide, disjuncte şi finite. Se dă o funcţie $a : S \cup T \to \mathbb{N}$. 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 şi T sunt clasele bipartiţiei lui G). Arătaţi că această problemă poate fi rezolvată în timp polinomial ca o problemă de flux maxim într-o anumită reţea.

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

Exerciţii pentru seminarul din săptămâna viitoare

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

Exerciţiul 3. Fiecare student dintr-o mulţime S de cardinal $n > 0$ subscrie la o submulţime de 4 cursuri opţionale dintr-o mulţime C de cardinal $k > 4$. Descrieţi un algoritm (de complexitate timp polinomială) care să determine (dacă există) o alocare a studenţilor către cursurile opţional din C astfel încât fiecare student să fie asignat la exact 3 cursuri (din cele 4 la care a subscris) şi fiecare curs să aibă asignaţi cel mult $\lceil n/k \rceil$ studenţi.

Exerciţiul 4.

(a) Adevărat sau fals? Într-o reţea $R = (G, s, t, c)$ cu capacităţi distincte există un singur flux maxim. De ce?

(b) Descrieţi şi demonstraţi corectitudinea unui algoritm de complexitate (timp) polinomială care să decidă dacă într-o reţea dată există un singur flux maxim.

Exerciţii 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

Exerciţiul 5. O companie IT are n angajaţi $P_1, P_2, \ldots, P_n$ care trebuie să lucreze la m proiecte $L_1, L_2, \ldots, L_m$. Pentru orice angajat P_i avem o listă $\mathcal{L}_i$ a proiectelor la care poate lucra şi s_i numărul de projecte din $\mathcal{L}_i$ pe care le poate termina într-o săptămână ($s_i \leqslant |\mathcal{L}_i|$). Fiecare proiect va fi alocat unui singur angajat.
Cum se poate determina numărul minim de săptămâni necesare terminării tuturor proiectelor folosind fluxuri în reţele.

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

Exerciţiul 6. Planul de evacuare de urgenţă al unei clădiri este descris ca un grid $n \times n$; frontierele celulelor sunt rutele de eavcuare către exteriorul clădirii (grid-ului). O instanţă a problemei evacuării conţine dimensiunea n a grid-ului şi m puncte de plecare (colţurile celulelor).

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

Exerciţii pentru seminarul din săptămâna viitoare

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

Exerciţiul 6 (continuare). Instanţa primeşte un răspuns afirmativ dacă există m drumuri disjuncte către frontiera grid-ului plecând din cele m puncte de mai sus. Dacă drumurile acestea nu există, atunci instanţa primeşte un răspun negativ.
Determinaţi o reprezentare a acestei problem a evacuării ca o problemă de flux într-o reţea. Descrieţi un algoritm eficient pentru a recunoaşte instanţele pozitive ale problemei (care este complexitatea timp a algoritmului?).

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

Exerciţiul 7. Fie $G = (V, E)$ un graf cu n noduri $\{v_1, v_2, \ldots, v_n\}$ şi $c : E \to \mathbb{R}_+$ o funcţie de capacitate pe muchiile lui G. O tăietură în G este o bipartiţie (S, T) a lui V. Capacitatea unei tăieturi (S, T) este $c(S, T) = \sum_{e \in E, |e \cap S| = 1} c(e)$.

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 -

Exerciţiul 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) Arătaţi că se poate determina o tăietură minimă în timp polinomial rezolvând un număr polinomial de probleme de flux maxim în anumite reţele.

(b) Pentru $G = C_n$ (circuit indus de ordin $n \geqslant 3$) cu toate capacităţile 1, arătaţi că există $\frac{n(n-1)}{2}$ tăieturi minime.

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

Exerciţii pentru seminarul 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 -

Exerciţiul 8. Fie $G = (V; E)$ un digraf şi $w : V \to \mathbb{R}$ astfel ca $w(V) \cap \mathbb{R}_+$ şi $w(V) \cap \mathbb{R}_- \neq \varnothing$. O submulţime $A \subseteq V$ se numeşte *izolată* în G dacă nu există nici un arc care iese din A. *Ponderea* unei submulţimi $A \subseteq V$ este $w(A) = \sum_{v \in A} w(v)$. Descrieţi un algoritm de compexitate polinomială bazat pe un algoritm de flux maxim într-o anumită reţea care să determine o submulţime izolată de pondere maximă în G.

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

Exerciţiul 9. Fie $G = (S, T; E)$ un graf bipartit. Demonstraţi teorema lui Hall (*există un cuplaj în G care saturează toate nodurile din S dacă şi numai dacă (H)* $\forall A \subseteq S, |N_G(A)| \geqslant |A|$) folosind *teorema flux maxim - secţiune minimă* pe o reţea particulară.

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

Cuprins

1. Fluxuri în reţele
   - Fluxuri de cost minim
2. Reduceri în timp polinomial pentru probleme pe grafuri
   - Mulţimii stabile de cardinal maxim
   - Colorări ale nodurilor
   - Probleme Hamiltoniene
3. Exerciţii 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

Să presupunem că în reţeaua $R = (G, s, t, c)$, se dă în plus o **funcţie de cost**: $a : E \to \mathbb{R}$; $\forall ij \in E$, $a(ij) = a_{ij}$ este costul arcului ij (interpretat ca fiind costul trimiterii unei "unităţi" de flux pe arcul ij).

Dacă x este un flux în R, atunci **costul lui x este**

$$\mathbf{a(x) = \sum_{i,j} a_{ij} x_{ij}}.$$

Problema fluxului de cost minim

Date: R o reţea, $a : E \to \mathbb{R}$ o funcţie de cost, şi $v \in \mathbb{R}_+$,

Determină: un flux x^0 în R astfel încât

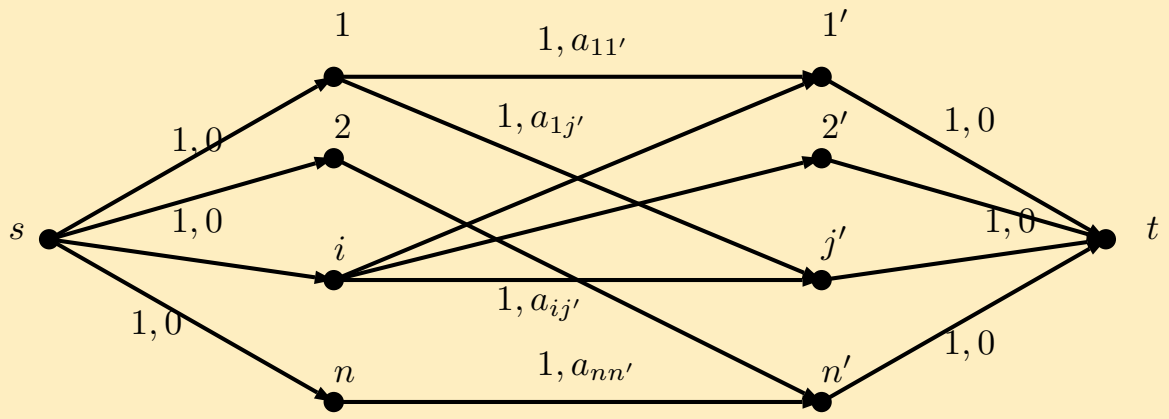
$$a(x^0) = \min\{a(x) \ : \ x \text{ flux în } R, v(x) = v\}.$$

Să observăm că, dacă v nu este mai mare decât valoarea fluxului maxim în R, atunci problema are întotdeauna soluţii ($a(x)$ este o funcţie liniară definită pe mulţimea nevidă şi compactă din $\mathbb{R}^{m+1}$ a tuturor fluxurilor de valoare v).

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

1. Problema asignării. Se dau n muncitori şi n slujbe. Costul repartizării muncitorului i slujbei j este a_{ij}. Să se asigneze fiecare muncitor câte unei slujbe astfel încât costul total să fie minim.

Considerăm reţeaua bipartită de mai jos, unde fiecare arc este etichetat cu capacitatea sa urmată de cost. Astfel, $c_{ij'} = 1$, $c_{si} = 1$, $a_{si} = 0$, $c_{j't} = 1$, şi $a_{j't} = 0$, $\forall i, j \in \{1, 2, \ldots, n\}$.



- 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 soluţia 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 $D_1, \ldots, D_n$ în cantităţile $d_1, \ldots, d_n$, respectiv, este cerut decătre clienţii $C_1, \ldots, C_m$ în cantităţile $c_1, \ldots, c_m$, respectiv. Se cunosc costurile de transport unitar, a_{ij} - de la depozitul D_i la clientul C_j, $\forall i \in \{1, \ldots, n\}$,$\forall j \in \{1, \ldots, m\}$.

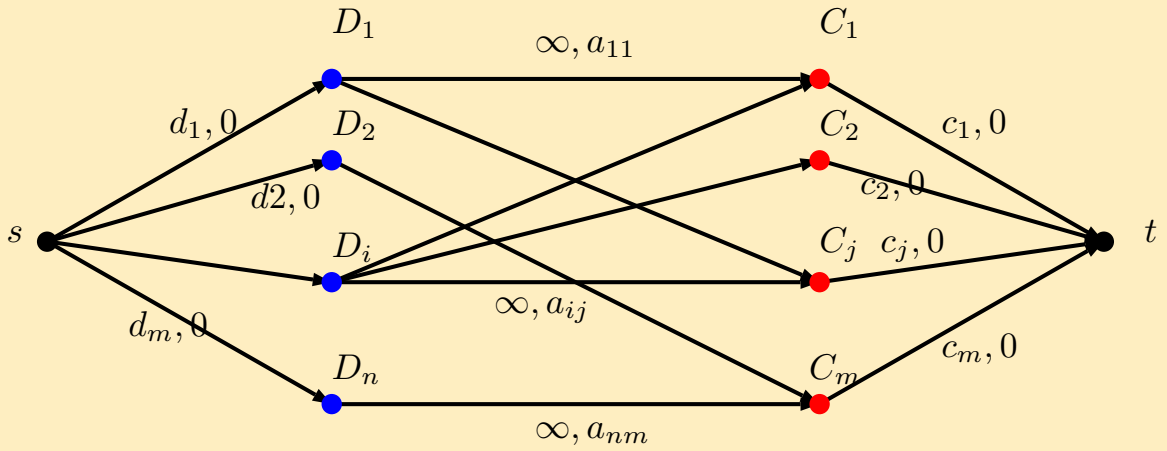
Să se determine o schemă de transport care să satisfacă cererea clienţilor cu un cost total al transportului minim.

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 * 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

Problema are soluţie numai dacă $\sum_{i=1}^{n} d_i \geqslant \sum_{j=1}^{m} c_j$. În acest caz, un flux de cost minim de valoare $v = \sum_{i=1}^{m} c_i$, în reţeaua de mai jos, rezolvă problema.



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

Definiţie

Fie x un flux în $R = (G, s, t, c)$ şi $a : E \to \mathbb{R}$ o funcţie 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 parcurgere alui C (este posibil ca ambele sensuri să ofere A-drumuri relativ la x).*

Remarci

- Dacă P este un drum de creştere relativ la x, atunci $x^1 = x \otimes r(P)$ este un flux de valoare $v(x^1) = v(x) + r(P)$ şi 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 $v(x^1) = v(x)$ şi cost $a(x^1) = a(x) + r(C) \cdot a(C)$. Urmează că **dacă $a(C) < 0$ atunci x^1 este un flux de aceeaşi valoare cu x dar de cost $a(x^1) < a(x)$.**

Teorema 1

Un flux x de valoare v este un flux de cost minim dacă şi numai dacă nu există un A-drum închis de cost negativ relativ la x în R.

Demonstraţie. "$\Rightarrow$" 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

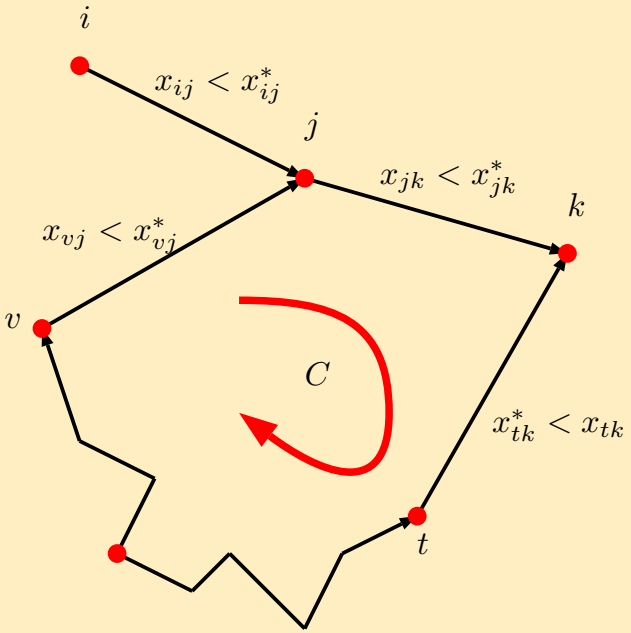
$$\Delta(x, x^*) = \min\{\Delta(x, x') \,:\, x' \text{ flux de cost minim de valoare } v\},$$

unde $\Delta(x, x') = |\{ij \in E \,:\, x_{ij} \neq x'_{ij}\}|$.

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

Dacă $\Delta(x, x^*) = 0$, atunci $x = x^*$, astfel x este un flux de cost minim. Altfel, $\Delta(x, x^*) > 0$ şi există ij astfel încât $x_{ij} \neq x^*_{ij}$. Să presupunem că $0 \leqslant x_{ij} < x^*_{ij} \leqslant c_{ij}$ (dacă $x_{ij} > x^*_{ij}$, raţionamentul este similar). Din legea conservării fluxului, există $jk \in E$ astfel încât $0 \leqslant x_{jk} < x^*_{jk} \leqslant c_{jk}$, sau există $kj \in E$ astfel încât $0 \leqslant x^*_{kj} < x_{kj} \leqslant c_{jk}$.

Deoarece numărul de noduri este finit, repetând acest raţionament, obţinem, C, un A-drum închis relativ la x 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

Dacă parcurgem C în sens contrar, obţinem un A-drum închis, C', relativ la x^*. Deoarece $a(C) \geqslant 0$ (din ipoteză), şi $a(C') = -a(C)$ urmează că $a(C) = 0$. (x^* este de cost minim; astfel, din implicaţia directă, $a(C') \geqslant 0$).

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' satisface $v(x') = v(x^*) = v$, $a(x') = a(x^*) + \delta(C') \cdot a(C') = a(x^*)$.

Astfel x' este un flux de cost minim de valoare v, dar $\Delta(x, x') < \Delta(x, x^*)$, în contradicţie cu alegerea lui x^*. Astfel $\Delta(x, x^*) = 0$, şi teorema este demonstrată. $\square$

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

Teorema 2

Dacă x este un flux de cost minim de valoare v şi P_0 este un drum de creştere relativ la x astfel încât

$$a(P_0) = \min\{a(P) \ : \ P \text{ drum de creştere relativ la } x\},$$

atunci $x^1 = x \otimes r(P_0)$ este un flux de cost minim de valoare $v(x^1) = v + r(P_0)$.

Demonstraţie. Omisă.

Un drum de creştere de cost minim poate fi găsit folosind un algoritm pentru drumuri de cost minim. Dacă x este un flux în R şi $a : E \to \mathbb{R}$ este o funcţie de cost, atunci luând $a_{ij} = \infty$ dacă $ij \notin E$ (când $xij = 0$), construim

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

$$\overline{a}_{ij} = \begin{cases} a_{ij}, & \text{dacă } x_{ij} < c_{ij} \text{ şi } x_{ji} = 0, \\ \min\{a_{ij}, -a_{ji}\}, & \text{dacă } x_{ij} < c_{ij} \text{ şi } x_{ji} > 0, \\ -a_{ji}, & \text{dacă } x_{ij} = c_{ij} \text{ şi } x_{ji} > 0, \\ +\infty, & \text{dacă } x_{ij} = c_{ij} \text{ şi } x_{ji} = 0. \end{cases}$$

Un st-drum de cost $\overline{a}$ minim corespunde unui drum de creştere de cost minim relativ la x în R, şi 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 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

fie x un flux de valoare $v' \leqslant v$; // 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 $\overline{a}$ negativ) **do**

  $x \leftarrow x \otimes r(C)$;

end while

while ($v(x) < v$) **do**

  determină un st-drum P de cost $\overline{a}$ minim;

  $x \leftarrow x \otimes \min\{r(P), v - v(x)\}$;

end while

Complexitatea timp al celui de-al doilea while este $\mathcal{O}(n^3 v)$ (dacă pornim cu fluxul nul şi capacităţile sunt întregi). Primul while poate fi implementat astfel încât să aibă $\mathcal{O}(nm^2 \log n)$ iteraţii.

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

Memento

- Fie $P_i : I_i \to \{da, nu\}$ ($i \in \{1, 2\}$) două **probleme de decizie**. **P_1 se reduce polinomial la P_2**, şi notăm aceasta prin $P_1 \leqslant_P P_2$, dacă există o funcţie calculabilă în timp polinomial $\Phi : I_1 \to I_2$, astfel încât $P_1(i) = P_2(\Phi(i)), \forall i \in I_1$.
- Funcţiile Φ vor fi definite folosind un algoritm care construieşte, pentru fiecare instanţă $i_1 \in I_1$, o instanţă $i_2 \in I_2$ în timp polinomial (în dimensiunea lui i_1), astfel încât $P_1(i_1) = da$ dacă şi numai dacă $P_2(i_2) = da$.
- Construcţia din spatele reducerii în timp polinomial arată cum prima problemă poate fi rezolvată eficient folosind un oracol pentru cea de-a doua.

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

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

- Relaţia $\leqslant_P$ este o relaţie tranzitivă pe mulţimea problemelor de decizie (deoarece, clasa funcţiilor polinomial calculabile este închisă la compunere).

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

Exemplu

SAT $\leqslant_P$ 3SAT

SAT

Instanţă: $U = \{u_1, u_2, \ldots, u_n\}$ o mulţime finită de variabile booleene;

$$C = C_1 \wedge C_2 \ldots \wedge C_m \text{ o formulă CNF peste } U:$$

$$C_i = v_{i1} \vee v_{i_2} \vee \ldots \vee v_{i_{k_i}}, i = \overline{1, m}, \text{ unde}$$

$$\forall i_j, \exists \alpha \in \{1, 2, \ldots, n\} \text{ a. î. } v_{i_j} = u_\alpha \text{ sau } v_{i_j} = \overline{u}_\alpha.$$

Întrebare: Există o asignare $t : U \to \{true, false\}$ a. î. $t(C) = true$?

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. 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 restricţia problemei **SAT** la mulţimea instanţelor 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).

$$\mathbf{NP} \neq \mathbf{NP} \cap \mathbf{coNP} = \mathbf{P}$$

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 *

Reduceri în timp polinomial - Problema mulţimii stabile de cardinal maxim

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

SM

Instanţă: $G = (V, E)$ un graf şi $k \in \mathbb{N}$.

Întrebare: Există o mulţime stabilă S în G astfel încât $|S| \geqslant k$?

Teorema 3

(Karp, 1972). 3SAT $\leqslant_P$ SM.

Demonstraţie. Fie $U = \{u_1, u_2, \ldots, u_n\}$, $(n \in \mathbb{N}^*)$, $C = C_1 \wedge C_2 \ldots \wedge C_m$ $(m \in \mathbb{N}^*)$ cu $C_i = v_{i1} \vee v_{i_2} \vee v_{i_3}$, $i = \overline{1, m}$, (unde $\forall i_j, \exists \alpha \in \{1, 2, \ldots, n\}$ a. î. $v_{i_j} = u_\alpha$ sau $v_{i_j} = \overline{u}_\alpha$) reprezentând datele unei instanţe a problemei **3SAT**.

Vom construi în timp polinomial (în $n + m$) un graf G şi $k \in \mathbb{N}$, astfel încât există o asignare t care satisface C dacă şi numai dacă există o mulţime stabilă S în G astfel încât $|S| \geqslant k$.

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

Demonstraţie (continuare).

Construcţia grafului G:

- $\forall i \in \{1, 2, \ldots, n\}$, fie grafurile disjuncte $T_i = (\{u_i, \overline{u}_i\}, \{u_i\overline{u}_i\})$.
- $\forall j \in \{1, 2, \ldots, m\}$, fie grafurile disjuncte $Z_j = (\{a_{j1}, a_{j2}, a_{j3}\}, \{a_{j1}a_{j2}, a_{j2}a_{j3}, a_{j3}a_{j1}\})$.
- $\forall j \in \{1, 2, \ldots, m\}$, fie mulţimea 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 mulţimii stabile de cardinal maxim

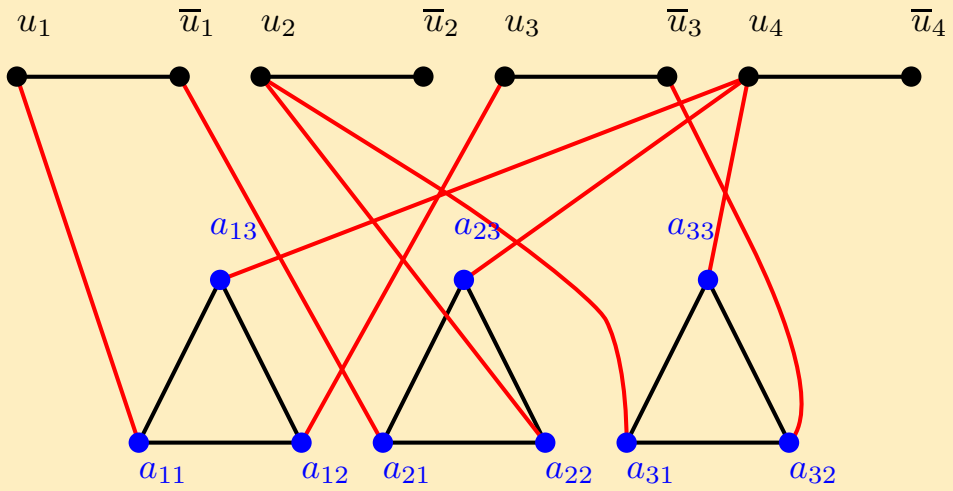
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Evident, construcţia lui G se poate face în timp polinomial relativ la dimensiunea $n + m$ a instanţei **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 mulţimii stabile de cardinal maxim

Să presupunem că răspunsul la întrebarea problemei **SM** pentru (G, k) instanţă este da.

$\exists S \in \mathcal{S}_G$ (familia tuturor mulţimilor stabile din G) astfel încât $|S| \geqslant k$. Deoarece fiecare mulţime stabilă poate avea cel mult un nod în fiecare $V(T_i)$ şi fiecare $V(Z_j)$, urmează că $|S| = k$ şi $|S \cap V(T_i)| = 1$, $|S \cap V(Z_j)| = 1$, $\forall i = \overline{1, n}$, $\forall j = \overline{1, m}$.

Fie $t : U \to \{true, false\}$ dată prin

$$t(u_i) = \begin{cases} true, & \text{dacă } S \cap V(T_i) = \{\overline{u}_i\} \\ false, & \text{dacă } S \cap V(T_i) = \{u_i\} \end{cases}$$

Atunci, $t(C_j) = true$, $\forall j = \overline{1, m}$, astfel $t(C) = true$, şi răspunsul la **3SAT** este da.

Într-adevăr, $\forall j = \overline{1, m}$, dacă $C_j = v_{j1} \vee v_{j1} \vee v_{j3}$ şi $S \cap V(Z_j) = \{a_{jk}\}$ $(k \in \{1, 2, 3\})$, atunci (deoarece $a_{jk} v_{jk} \in E$) urmează că $v_{jk} \notin S$.

Reduceri în timp polinomial - Problema mulţimii 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 $\overline{u}_\alpha \in S$, şi - din definiţia lui t - avem $t(u_\alpha) = true$, adică, $t(v_{jk}) = true$, ceea ce implică $t(C_j) = true$.
- Dacă $v_{jk} = \overline{u}_\alpha$, atunci $\overline{u}_\alpha \notin S$, deci $u_\alpha \in S$, şi - din definiţia lui t - avem $t(\overline{u}_\alpha) = true$, adică, $t(v_{jk}) = true$, ceea ce implică $t(C_j) = true$.

Reciproc, dacă răspunsul la întrebarea problemei **3SAT** este da, atunci $\exists t : U \to \{true, false\}$ astfel încât $t(C_j) = true$, $\forall j = \overline{1, m}$.

Fie $\overline{S}_1$ mulţimea stabilă $\overline{S}_1 = \bigcup_{i=1}^{n} V_i'$ cu n noduri, unde

$$V_i' = \begin{cases} \{\overline{u}_i\}, & \text{dacă } t(u_i) = true\} \\ \{u_i\}, & \text{dacă } t(u_i) = false\} \end{cases}$$

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

Atunci, deoarece $t(C_j) = true$, $\forall j = \overline{1, m}$, urmează că există $k_j \in \{1, 2, 3\}$ astfel încât $t(v_{jk_j}) = true$. Fie $\overline{S}_2 = \bigcup_{j=1}^{m} \{a_{jk_j}\}$. Evident, $\overline{S}_2$ este o mulţime stabilă în G având m noduri.

Fie $\overline{S} = \overline{S}_1 \cup \overline{S}_2$. Evident, $|\overline{S}| = n + m = k$ (astfel $|\overline{S}| \geqslant k$). Dacă arătăm că $\overline{S}$ este o mulţime stabilă, atunci răspunsul la SM pentru instanţă (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:

- $v = u_\alpha$, $w = a_{jk_j}$, $\alpha \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$, $k_j \in \{1, 2, 3\}$ şi $v_{jk_j} = u_\alpha$.
  Deoarece $t(v_{jk_j}) = true$, urmează că $t(u_\alpha) = true$, astfel $v = u_\alpha \notin \overline{S}_1$, contradicţie.

Reduceri în timp polinomial - Problema mulţimii 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 - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

- $v = \overline{u}_\alpha$, $w = a_{jk_j}$, $\alpha \in \{1, 2, \ldots, n\}$, $j \in \{1, 2, \ldots, m\}$, $k_j \in \{1, 2, 3\}$ şi $v_{jk_j} = \overline{u}_\alpha$.
  Deoarece $t(v_{jk_j}) = true$, urmează că $t(\overline{u}_\alpha) = true$, astfel $t(u_\alpha) = false$. Astfel, $v = \overline{u}_\alpha \notin \overline{S}_1$, contradicţie. $\square$

O demonstraţie similară poate fi făcută pentru a arăta că **SAT** $\leqslant_P$ **SM**, singura diferenţă este că Z_i sunt grafuri complete cu k_i 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. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - 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

Instanţă: $G = (V, E)$ graf şi $p \in \mathbb{N}^*$.

Întrebare: Există o p-colorare (a nodurilor) lui G?

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

Teorema 4

3SAT $\leqslant_P$ COL.

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

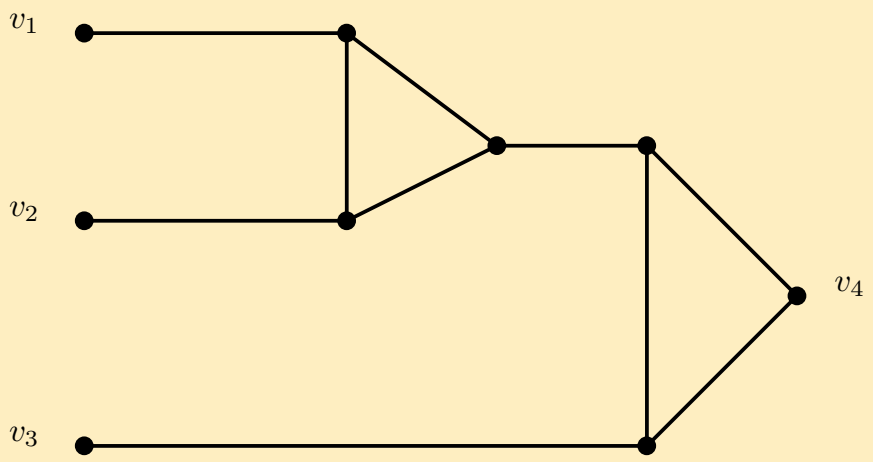
Această teoremă arată că problema colorării nodurilor este **NP**-hard. Demonstraţia dată mai jos arată că problema, care se obţine din **COL** prin restricţia la instanţele cu $p = 3$, care poate fi numită 3-**COL**, este **NP**-hard, de asemeni.

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

G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms

Lema 1

Fie H graful

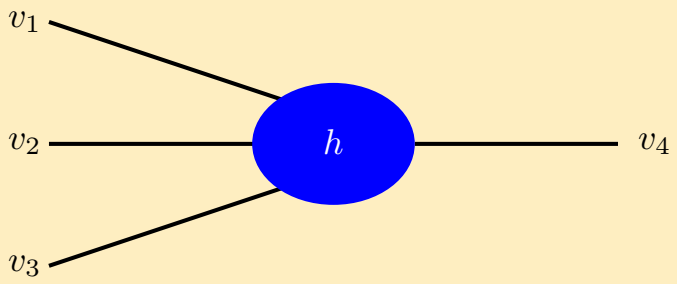


a) Dacă c este o 3-colorare a lui H a. î. $c(v_1) = c(v_2) = c(v_3) = a \in \{1, 2, 3\}$, atunci în mod necesar $c(v_4) = a$ (forcing).

b) Dacă $c : \{v_1, v_2, v_3\} \to \{1, 2, 3\}$ satisface $c(\{v_1, v_2, v_3\}) \neq \{a\}$, $(a \in \{1, 2, 3\})$, atunci c poate fi extinsă la o 3-colorare c a lui H cu $c(v_4) \neq a$.

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

Demonstraţie. Se examinează lista 3-colorărilor lui H.
Vom utiliza reprezentarea simplificată a grafului H:

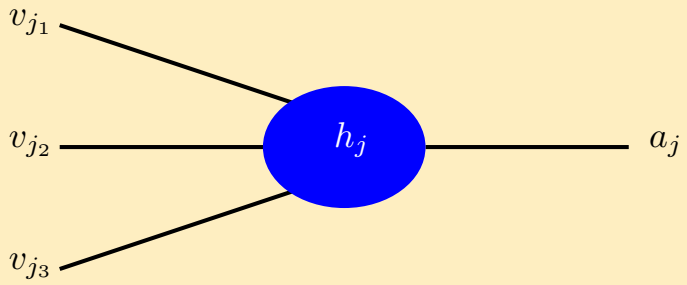


Demonstraţia Teoremei 4. Considerăm datele unei instanţe a **3SAT**: $U = \{u_1, \ldots, u_n\}$, $(n \in \mathbb{N}^*)$, o mulţime de variabile booleene, şi $C = C_1 \wedge \ldots \wedge C_m$, $(m \in \mathbb{N}^*)$ formulă CNF cu $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$, $\forall j = \overline{1, m}$, unde $\forall i = \overline{1, 3}$, $\exists \alpha$ a. î. $v_{j_i} = u_\alpha$ sau $v_{j_i} = \overline{u}_\alpha$.
Vom construi un graf G cu proprietatea că G este 3-colorabil dacă şi numai dacă răspunsul la **3SAT** pentru această instanţă este da, adică, există o asignare $t : U \to \{true, false\}$ astfel încât $t(C) = true$.

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

Construcţia necesită un timp polinomial, şi constă din următorii paşi:

- Considerăm grafurile disjuncte (V_i, E_i), $\forall i = \overline{1, n}$, unde $V_i = \{u_i, \overline{u}_i\}$ şi $E_i = \{u_i\overline{u}_i\}$.
- Pentru $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$, $\forall j = \overline{1, m}$, considerăm grafurile:



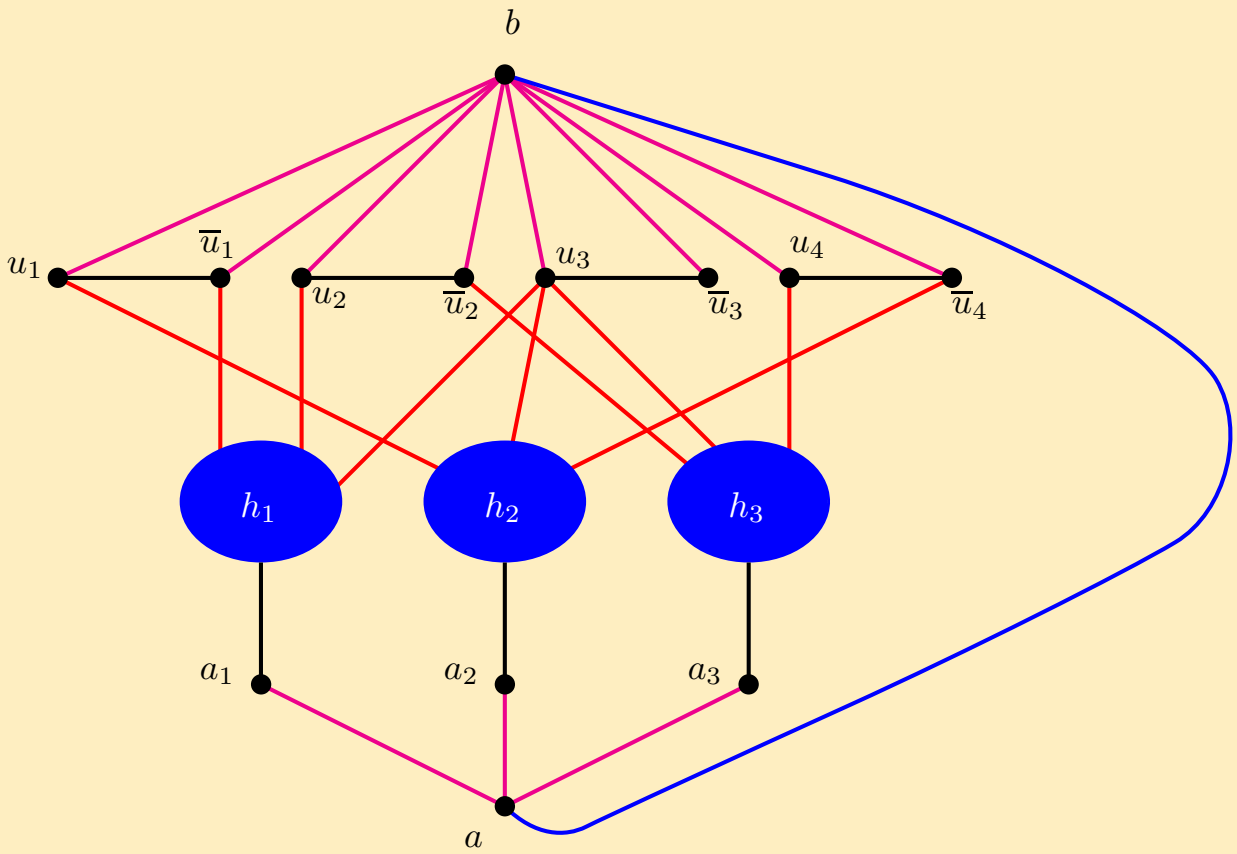
  unde, v_{j_k} $(k = \overline{1, 3})$ sunt nodurile corespunzătoare literalilor v_{j_k}, grafurile h_j sunt disjuncte, şi a_j sunt noduri distincte.
- Considerăm un nod nou a, şi toate muchiile aa_j, $\forall j = \overline{1, m}$.
- Considerăm un nod nou b, toate muchiile bu_i, $b\overline{u}_i$, $\forall i = \overline{1, n}$ şi ba.

Graful G are un număr de liniar de noduri relativ la $n + m$.

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

Exemplu

$U = \{u_1, u_2, u_3, u_4\}$, $C = (\overline{u}_1 \vee u_2 \vee u_3) \wedge (u_1 \vee u_3 \vee \overline{u}_4) \wedge (\overline{u}_2 \vee u_3 \vee u_4)$.
Graful G este



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

Să presupunem că răspunsul la **3SAT** pentru instanţă considerată este da

Astfel $\exists t : U \to \{true, false\}$ a. î. $t(C) = true$, adică, $t(C_j) = true$, $\forall j = \overline{1, m}$. Vom arăta că G este 3-colorabil.

Colorăm mai întâi nodurile u_i şi $\overline{u}_i$, $\forall i \overline{1, n}$.

$$\begin{cases} c(u_i) = 1 \text{ şi } c(\overline{u}_i) = 2, \text{dacă } t(u_i) = true \\ c(u_i) = 2 \text{ şi } c(\overline{u}_i) = 1, \text{dacă } t(u_i) = false \end{cases}$$

Observăm că, dacă v este un literal, atunci $c(v) = 2$ dacă şi numai dacă $t(v) = false$.

Deoarece t este o asignare pentru care satisface C, $t(C_j) = true$, $\forall j = \overline{1, m}$. Urmează că $c(\{v_{j_1}, v_{j_2}, v_{j_3}\}) \neq \{2\}$, $\forall j = \overline{1, m}$.

Din Lema 1 b), putem extinde c la o 3-colorare, în fiecare graf h_j, astfel încât $c(a_j) \neq 2$, adică $c(a_j) \in \{1, 3\}$.

Luând $c(a) = 2$ şi $c(b) = 3$, obţinem o 3-colorare a 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

Reciproc, să presupunem că G este 3-colorabil.
Putem presupune că $c(b) = 3$ şi $c(a) = 2$ (altfel, redenumim culorile).
Urmează că $\{c(u_i), c(\overline{u}_i\} = \{1, 2\}$, $\forall i = \overline{1, n}$ şi $c(a_j) \in \{1, 3\}$, $\forall j = \overline{1, m}$.
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 $t : U \to \{true, false\}$ prin

$$t(u_i) = \begin{cases} true, & \text{dacă } c(u_i) = 1 \\ false, & \text{dacă } c(u_i) = 2 \end{cases},$$

Obţinem o asignare cu proprietatea că $t(C_j) = true$, $\forall j = \overline{1, m}$.
Astfel răspunsul la **3SAT** pentru instanţa dată este da.

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

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 conţine 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?

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 şi P_2 sunt echivalente polinomial dacă $P_1 \leqslant_P P_2$ şi $P_2 \leqslant_P P_1$.

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

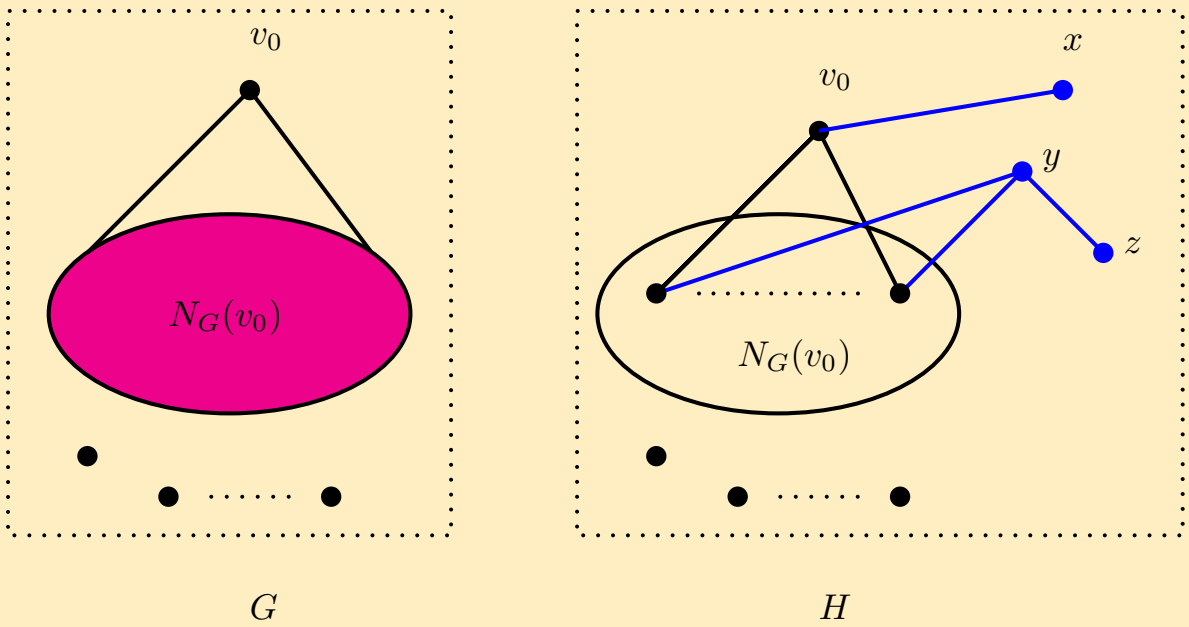
Demonstraţia Teoremei 5.

CH $\leqslant_P$ TR Fie G un graf şi $v_0 \in V(G)$. Construim în timp polinomial un graf H astfel încât G este Hamiltonian dacă şi numai dacă H este trasabil.

Fie $V(H) = V(G) \cup \{x, y, z\}$ şi $E(H) = E(G) \cup \{xv_0, yz\} \cup \{wy : w \in V(G) \cap N_G(v_0)\}$.

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

Demonstraţia Teoremei 5 (continuare).

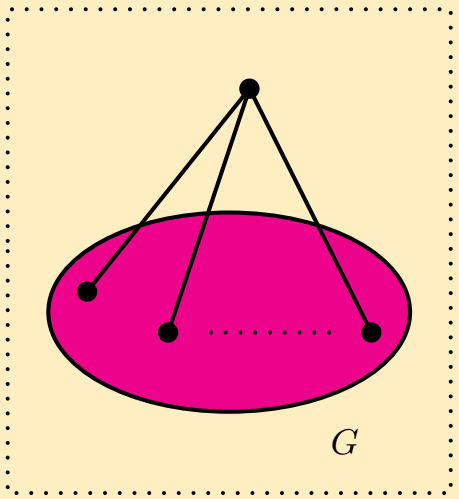


Atunci, H este trasabil dacă şi numai dacă are un drum Hamiltonian, P, cu extremităţile x şi z (care au gradul 1 în H). P există în H dacă şi numai dacă în G există un drum Hamiltonian cu o extremitate în v_0 şi cealaltă un vecin al lui v_0, adică, dacă şi numai dacă G este Hamiltonian.

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

Demonstraţia Teoremei 5 (continuare).

TR $\leqslant_P$ CH Fie G un graf. Considerăm $H = G + K_1$. Atunci, H este Hamiltonian dacă şi numai dacă G are un drum Hamiltonian.



$G + K_1$

Echivalenţa problemelor **DCH** şi **DTR** se dovedeşte similar.

CH $\leqslant_P$ DCH Fie G un graf. Fie D digraful obţinut din G prin înlocuirea fiecărei muchii cu o pereche simetrică de arce.

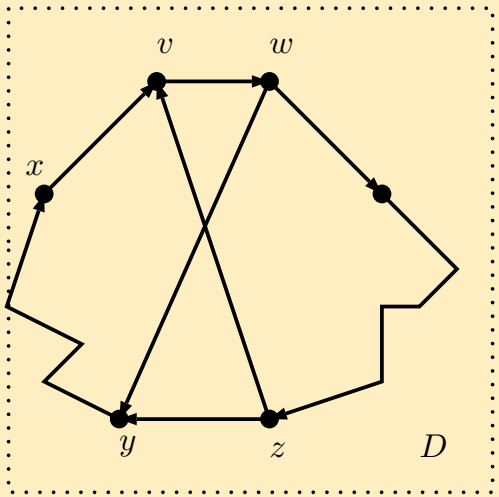
Evident fiecare circuit în G dă un circuit în D şi reciproc, fiecare circuit în D provine dintr-un circuit în G.

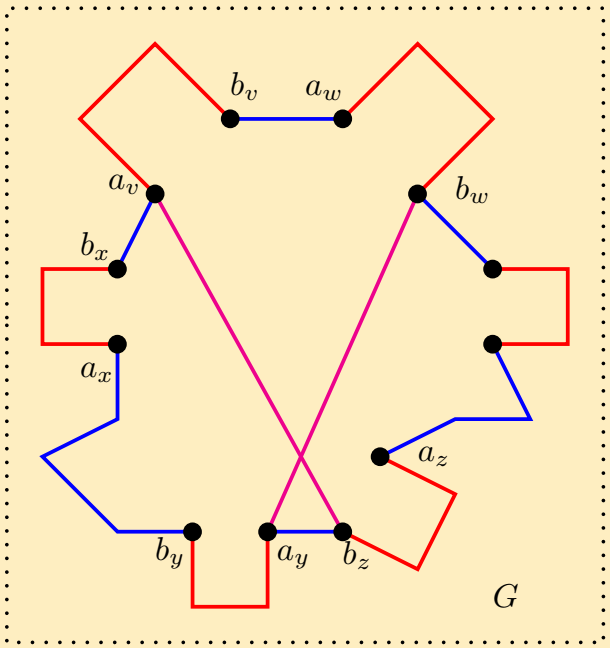
DCH $\leqslant_P$ CH fie D un digraf. Fiecare nod $v \in V(D)$ este înlocuit printr-un graf neorientat $P_3(v)$ cu extremităţi a_v şi b_v:

$$P_3(v) = (\{a_v, b_v, c_v, d_v\}, \{a_v c_v, c_v d_v, d_v b_v\}).$$

Fiecare arc $vw \in E(D)$ este înlocuit prin muchia (neorientată) $b_v a_w$.

Fie G graful obţinut (în timp polinomial) astfel:





C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 şi, reciproc, fiecare circuit în G corespunde unui circuit al lui D. Urmează că D este Hamiltonian dacă şi 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, şi $length(C) = 3 \cdot length(C') + length(C') = 4 \cdot length(C')$. Urmează că orice circuit al lui G este par, deci G este un graf bipartit.

Astfel demonstraţia de mai sus (**DCH** $\leqslant_P$ **CH**) este de fapt **DCH** $\leqslant_P$ **BCH**.

Deoarece **BCH** $\leqslant_P$ **CH** este evidentă, Teorema 5 este complet demonstrată. $\square$

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

Exerciţii pentru seminarul 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

Exerciţiul 1. Arătaţi că următoarea problemă este **NP**-completă

INT

Instanţă: $n, m \in \mathbb{N}^*$, $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$.

Întrebare: Există o asignare $x \in \mathbb{Z}^n$ a. î. $Ax \leqslant b$?

(Inegalitatea $\leqslant$ dintre doi vectori este pe componente.) *Hint:* You can try **SM** $\leqslant_P$ **INT**.

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

Exerciţiul 2. Considerăm următoarea problemă de decizie

P

Instanţă: $G = (V, E$ un digraf şi $p \in \mathbb{N}$.

Întrebare: Există $A \subseteq V$ astfel încât $|A| \leqslant p$ şi $G - A$ nu conţine circuite?

Arătaţi că **SM** $\leqslant_P$ **P**.

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

Exerciţii pentru seminarul din săptămâna viitoare

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

Exerciţiul 3. Un hipergraf k-uniform este o pereche $H = (V, E)$, unde $V \neq \emptyset$ este o mulţime finită, $k \in \mathbb{N}^* \setminus \{1\}$, şi $E \subseteq \mathcal{P}_k(V) = \{A \subseteq E : |A| = k\}$. Este uşor 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 funcţie $c : V \to \{1, 2, \ldots, k\}$ 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

Instanţă: H un hipergraf k-uniform.

Întrebare: H este simplu?

(a) Arătaţi că problema 3-**SIMPLE** este **NP**-completă.

(b) Arătaţi că problema 2-**SIMPLE** este în **P**.

Exerciţii pentru seminarul din săptămâna viitoare

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

Exerciţiul 4. Considerăm următoarea problemă de decizie:

AGM

Instanţă: G un graf, $k \in \mathbb{N}$.

Întrebare: G are un arbore parţial T astfel încât $\Delta(T) \geqslant k$?

Arătaţi că $\mathbf{AGM} \in \mathbf{P}$.

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

Exerciţiul 5. Fie $D = (V, E)$ un digraf fără bucle. O mulţime 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.

(a) Arătaţi că un quasi-kernel poate fi construit în $\mathcal{O}(n + m)$, unde $n = |V|$ şi $m = |E|$.

(b) Arătaţi că 3-**SAT** se poate reduce în timp polinomial la problema determinării dacă într-un digraf dat există un quasi-kernel care conţine un nod dat.

Exerciţii 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

Exerciţiul 6. Considerăm următoarea problemă de decizie: **LPL**

Instanţă: G un graf, $k \in \mathbb{N}$.

Întrebare: Are G un drum P astfel încât $length(P) \geqslant k$?

Arătaţi că **LPL** este **NP**-completă.

Exerciţiul 7. Un kernel într-un digraf $G = (V, E)$ este o mulţime 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

Instanţă: G un digraf.

Întrebare: Are G un kernel?

Arătaţi că următoarea construcţie conduce la o reducere polinomială a lui **SAT** la **NUCLEU** (i.e. **SAT** $\leqslant_P$ **NUCLEU**):

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

Exerciţiul 7 (continuare). Pentru fiecare conjucţie de clauze, F, instanţă a lui **SAT**, definim un digraf G (o instanţă 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

$$v_x, v_x v_{\overline{x}}, v_{\overline{x}}, v_{\overline{x}} v_x, v_x;$$

- pentru fiecare clauză C şi fiecare literal u care apare în C adăugăm a G trei arce

$$v_u v_C^1, v_u v_C^2, v_u v_C^3.$$

Exerciţii pentru seminarul din săptămâna viitoare

Exercise 8. Considerăm următoarea problemă de decizie

2SAT

Instanţă: $\mathcal{C}$ o familie de clauze fiecare cu doi literali.

Întrebare: Există o asignare a valorilor de adevăr către variable astfel ca toate clauzele din $\mathcal{C}$ să fie satisfăcute?

Define G digraful (implicaţiilor): $V(G) =$ mulţimea literalilor folosiţi în $\mathcal{C}$ şi $E(G) = \{\overline{v}_j w_j, \overline{w}_j v_j \ : \ C_j = v_j \vee w_j,\ j = \overline{1, m}\}$ (fiecare clauză introduce în G două arce). Arătaţi că $\mathcal{C}$ este satisfiabilă dacă şi numai dacă x_i şi $\overline{x}_i$ aparţin la componente tari conexe diferite ale lui G, $\forall i = \overline{1, n}$. Arătaţi că această proprietate poate fi verificată în $\mathcal{O}(n + m)$.

Exercise 9. Arătaţi că următoarea problemă este **NP**-completă.

MAX-2SAT

Instanţă: $\mathcal{C}$ o familie de clauze fiecare cu cel mult doi literali şi $k \in \mathbb{N}$.

Întrebare: Există o asignare a valorilor de adevăr către variable astfel ca cel puţin k dintre clauze să fie satisfăcute?

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

Instanţă: $\mathcal{C}$ o familie de clauze fiecare cu doi literali.

Întrebare: Există o asignare a valorilor de adevăr către variable astfel ca în fiecare clauză să avem şi un literal adevărat şi unul fals?

Arătaţi că următoarea construcţie conduce la o reducere polinomială a lui **3SAT** la **NAE 3SAT** (i.e. **3SAT** $\leqslant_P$ **NAE 3SAT**):

- păstrăm variabilele booleene ale instanţei **3SAT**, $U = \{u_1, u_2, \ldots, u_n\}$ şi 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 şi înlocuim C_j cu două clauze:

$$C_j^1 = v_{j_1} \vee v_{j_2} \vee y_j, C_j^2 = v_{j_3} \vee x \vee \overline{y}_j.$$

Exerciţiul 11. Considerăm următoarea problemă de decizie

MAX-CUT

Instanţă: $G = (V, E)$ un graf, $c : E \to \mathbb{R}$ o funcţie de pondere şi $k \in \mathbb{R}$.

Întrebare: există o tăietură în G de pondere $\geqslant k$?

Arătaţi că următoarea construcţie duce la o reducere polinomială a problemei **NAE 3SAT** la **MAXCUT** (i.e. **NAE 3SAT** $\leqslant_P$ **MAXCUT**):

- considerăm o instanţă a problemei **NAE 3SAT** cu clauzele $\mathcal{C} = \{C_1, \ldots, C_m\}$ peste mulţimea de variabile booleene $U = \{u_1, u_2, \ldots, u_n\}$; putem presupune că fiecare clauză conţine trei literali diferiţi,
- $V(G) = \{u_i, \overline{u}_i \ : \ i = \overline{1, n}\}$ şi adăugăm la $E(G)$ muchiile $u_i\overline{u}_i$ de pondere $10m$,
- pentru fiecare clauză $C_j = v_{j_1} \vee v_{j_2} \vee v_{j_3}$ adăugăm la $E(G)$ trei muchii $v_{j_1}v_{j_2}$, $v_{j_2}v_{j_3}$ şi $v_{j_1}v_{j_3}$, fiecare de pondere 1,
- $k = 10nm + 2m$.

Exerciţii 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 *

Cuprins

1. Reduceri în timp polinomial pentru probleme pe grafuri
   - Probleme Hamiltoniene
   - Problema comis-voiajorului
2. Abordări ale problemelor NP-hard
   - TSP metrică: Algoritmul lui Christofides
   - Colorarea nodurilor: Algoritmul de colorare greedy
3. Exerciţii pentru seminarul din următoarea săptămână

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

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

Teorema 1

(Karp, 1972) SM $\leqslant_P$ CH.

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

Demonstraţie. Fie $G = (V, E)$ şi $j \in \mathbb{N}$ o instanţă a problemei **SM**. Vom construi în timp polinomial (relativ la $n = |V|$) un graf H astfel încât există o mulţime stabilă S în G cu $|S| \geqslant j$ dacă şi numai dacă H este Hamiltonian.

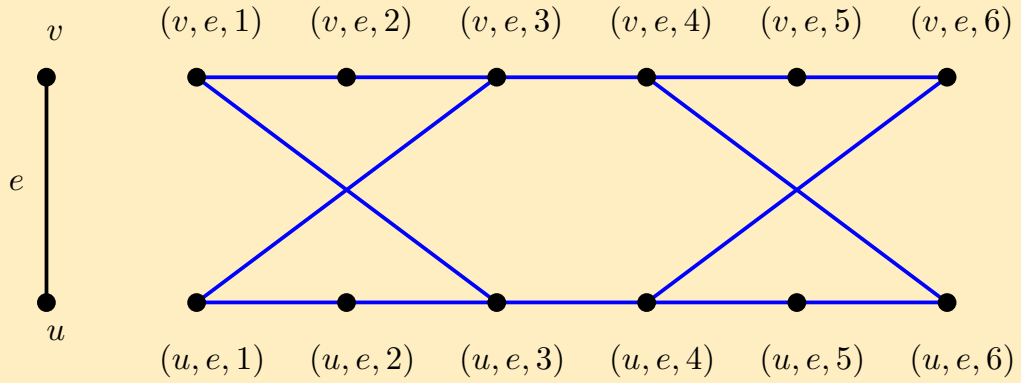
Fie $k = |V| - j$. Să presupunem că $k > 0$ (pentru a evita cazurile banale)

(i) Fie $A = \{a_1, a_2, \ldots, a_k\}$ o mulţime of k de noduri distincte.

(ii) Pentru fiecare muchie $e = uv \in E(G)$, considerăm graful $G'_e = (V'_e, E'_e)$ cu $V'_e = \{(w, e, i) : w \in \{u, v\}, i = \overline{1, 6}\}$ şi $E'_e = \{(w, e, i)(w, e, i + 1) : w \in \{u, v\}, i = \overline{1, 5}\} \cup \{(u, e, 1)(v, e, 3), (u, e, 3)(v, e, 1), (u, e, 4)(v, e, 6), (u, e, 6)(v, e, 4)\}$.

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

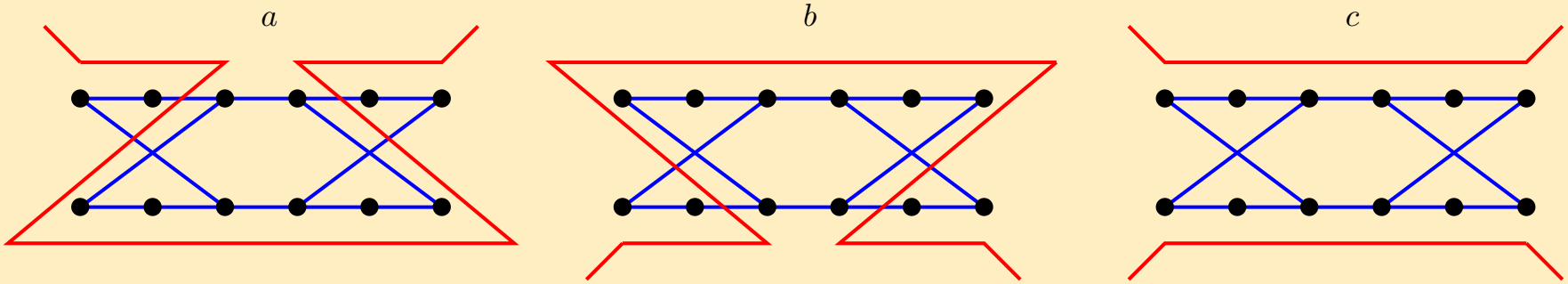
Demonstraţie (continuare).



Graful are proprietatea că, dacă este un subgraf indus al unui graf Hamiltonian, H, şi nici unul dintre nodurile (w, e, i) cu $w \in \{u, v\}$ şi $i = \overline{2,5}$ nu are alţi vecini în H, atunci singurele posibilităţi de parcurgere a lui ${G'}_e$ pe un circuit Hamiltonian sunt:

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

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

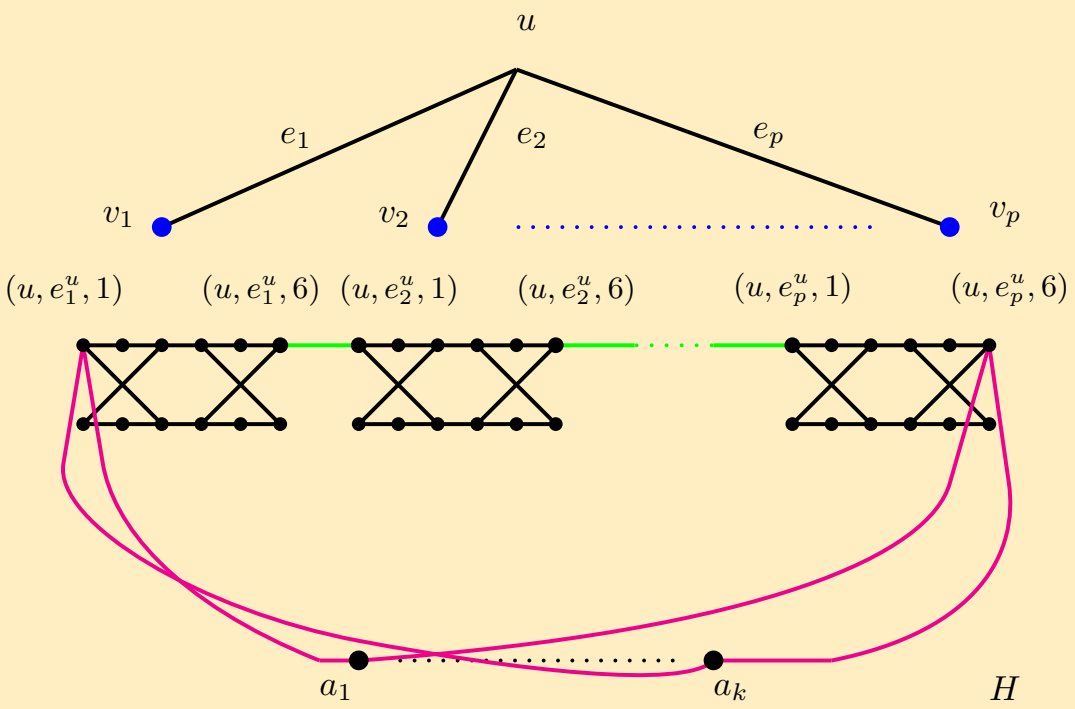


Demonstraţie (continuare). Astfel, dacă circuitul Hamiltonian "intră" în G'_e printr-un nod corespunzând lui u ($(u, e, 1)$ sau $(u, e, 6)$) atunci "părăseşte" graful G'_e de asemeni printr-un nod corespunzând lui u.

(iii) Pentru fiecare nod $u \in V$, considerăm (într-o ordine arbitrară) muchiile incidente în G cu u: $e_1^u = uv_1, e_2^u = uv_2, \ldots, e_p^u = uv_p$ ($p = d_G(u)$). Fie $E_u'' = \{(u, e_i^u, 6)(u, e_{i+1}^u, 1) : i = \overline{1, p-1}\}$ şi $E_u''' = \{a_i(u, e_1^u, 1), a_i(u, e_p^u, 6) : i = \overline{1, k}\}$

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

Demonstraţie (continuare). Graful H are $V(H) = A \cup \left(\bigcup_{e \in E} V'_e \right)$ şi $E(H) = \left(\bigcup_{e \in E} E'_e \right) \cup \left(\bigcup_{u \in V} (E''_u \cup E'''_u) \right)$. Evident, poate fi construit din G în timp polinomial (în n).



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

Demonstraţie (continuare). Acum arătăm că există o mulţime stabilă în G cu cel puţin j noduri dacă şi numai dacă H este Hamiltonian.
"$\Leftarrow$" Dacă H este Hamiltonian, atunci există C un circuit Hamiltonian în H. Deoarece A este o mulţime stabilă în H, A descompune circuitul C în exact k drumuri intern disjuncte: $D_{a_{i_1} a_{i_2}}, D_{a_{i_2} a_{i_3}}, \ldots, D_{a_{i_k} a_{i_1}}$.
Fie $D_{a_{i_j} a_{i_{j+1}}}$ un astfel de drum $(j+1 = 1+(j \ (mod \ k)))$. Din construcţia 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 asemeni printr-un nod corespunzător lui v_{i_j}.

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

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

Demonstraţie (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 şi 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 $t = 1$ şi $t' = d_G(v_{i_j})$, $x = 1$, $x' = 6$ sau $t = d_G(v_{i_j})$, $t' = 1$, $x = 6$, $x' = 1$.

Urmează că nodurile v_{i_j} sunt distincte.

Fie $V^* = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. 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.

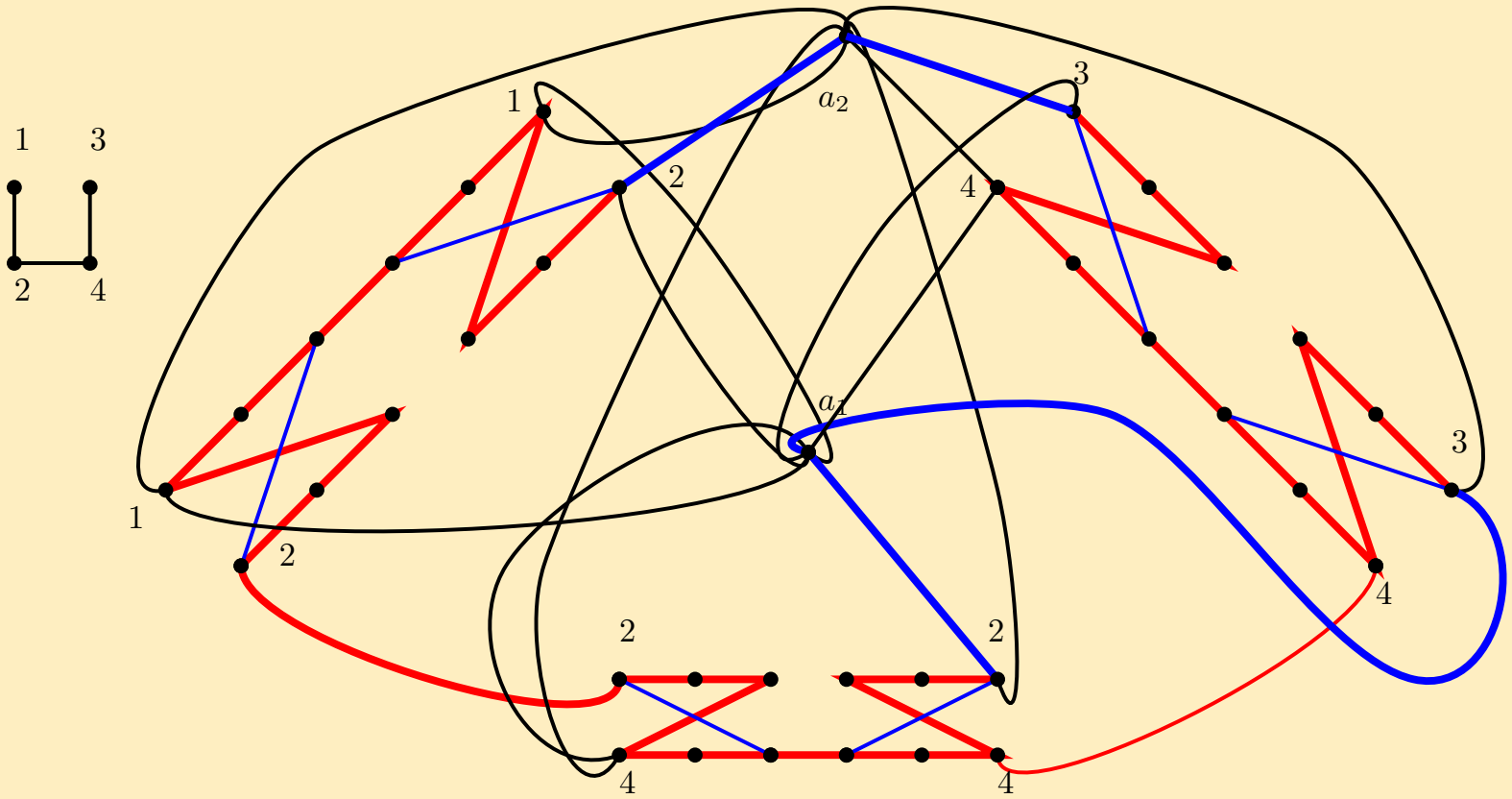
Deci $S = V \setminus V^*$ este o mulţime stabilă în G şi $|S| = j$.

Astfel, am arătat că, dacă H este Hamiltonian, atunci în G există o mulţime stabilă cu j noduri şi răspunsul la **SM** pentru instanţa G, j este da.

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

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

Demonstraţie (continuare).



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

Demonstraţie (continuare). " $\Rightarrow$ " Să presupunem că răspunsul la **SM** pentru instanţa G, j este da, astfel există S_0 o mulţime stabilă în G cu $|S_0| \geqslant j$. Există $S \subseteq S_0$ cu $|S| = j$. Fie $V^* = V \setminus S = \{v_1, v_2, \ldots, v_k\}$.

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 $u \in V^*$ şi $v \notin V^*$.
- drumul din cazul (a) în G'_e (desenat mai sus) dacă $u \notin V^*$ şi $v \in V^*$.

La reuniunea tuturor acestor drumuri adăugăm muchiile $a_i(v_i, e_1^{v_i}, 1)$, $(v_i, e_1^{v_i}, 6)(v_i, e_2^{v_i}, 1), \ldots, (v_i, e_{p-1}^{v_i}, 6)(v_i, e_p^{v_i}, 1)$, $(v_i, e_p^{v_i}, 6)a_{i+1}$, (cu $p = d_G(v_i)$), pentru $i = \overline{1, k}$.

Ceee ce se obţine este un circuit Hamiltonian în H. $\square$

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 şi $d : E \to \mathbb{R}_+$ o funcţie de pondere nenegativă pe muchiile sale, să se determine un circuit Hamiltonian, H_0, a. î. suma ponderilor pe muchiile lui H_0 să fie minimă (printre toate circuitele Hamiltoniene ale lui G).

Fie graful G poate fi o reţea constând dintr-o mulţime, V, de oraşe împreună c cu o mulţime, E, de rute directe între oraşe, funcţia d oferind, pentru fiecare muchie $uv \in E$, $d(uv) =$ distanţa pe ruta directă între oraşele u şi v. Fixând un oraş de start v_0, circuitul Hamiltonian H_0 reprezintă cel mai scurt traseu care vizitează toate oraşele exact odată (cu excepţia lui v_0) pe care îl poate parcurge un comis-voiajor plecând din v_0 şi 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 *

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

Considerăm următoarea formulare echivalentă a acestei probleme.

TSP Dat $n \in \mathbb{N}$ $(n \geqslant 3)$ şi $d : E(K_n) \to \mathbb{R}_+$, 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 complexităţii timp a acestei probleme, vom considera $d(e) \in \mathbb{N}$, pentru fiecare muchie e.

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

Instanţă: $n \in \mathbb{N}(n \geqslant 3)$, $d : E(K_n) \to \mathbb{N}$ şi $B \in \mathbb{N}$.

Întrebare: Există un circuit Hamiltonian, H_0, în K_n, a. î. $d(H_0) \leqslant B$?

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

Teorema 2

CH $\leqslant_P$ DTSP.

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

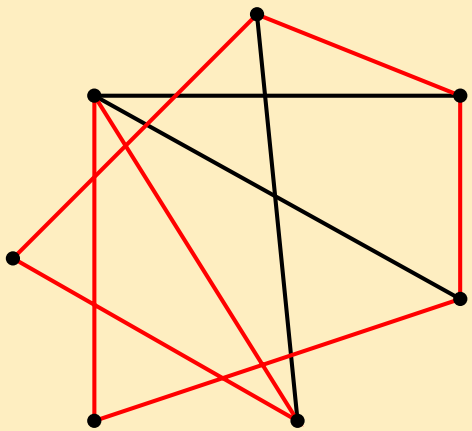
Demonstraţie. Fie $G = (V, E)$ ($|V| = n$) o instanţă a problemei **CH**. Construim în timp polinomial o instanţă, $d : E(K_n) \to \mathbb{N}$ şi $B \in \mathbb{N}$, a problemei **DTSP** astfel încât există un circuit Hamiltonian în K_n de pondere totală cel mult B dacă şi numai dacă G este graf Hamiltonian.

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

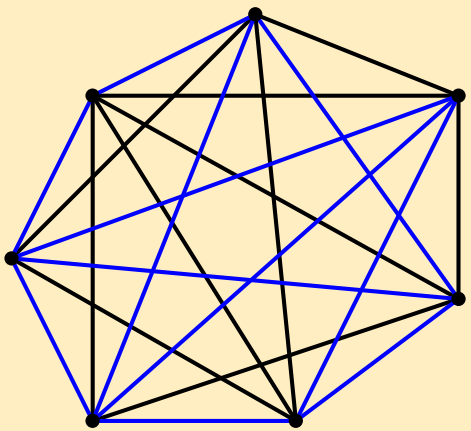
Fie

$$d(vw) = \begin{cases} 1, & \text{dacă } vw \in E(G) \\ 2, & \text{dacă } vw \in E(\overline{G}) \end{cases} \quad \text{şi } B = n.$$

Atunci, există în K_n un circuit Hamiltonian de pondere $\leqslant n$ dacă şi numai dacă există un circuit Hamiltonian C în K_n astfel încât $\forall e \in E$ $d(e) = 1$, adică, dacă şi numai dacă G are un circuit Hamiltonian:



G



weight 2
weight 1

Urmează că **TSP** este o problem ă **NP**-hard. □

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 instanţă **TSP**, un circuit Hamiltonian $H_{\mathcal{A}}$ al lui K_n, o aproximare a soluţiei optime H_o. Calitatea aproximării poate fi exprimată folosind următorul raport:

$$R_{\mathcal{A}}(n) = \sup_{d:E(K_n)\to\mathbb{R}_+, d(H_o)\neq 0} \frac{d(H_{\mathcal{A}})}{d(H_o)}$$

$$R_{\mathcal{A}} = \sup_{n\geqslant 3} R_{\mathcal{A}}(n).$$

Evident, algoritmul de aproximare $\mathcal{A}$ este util numai dacă $R_{\mathcal{A}}$ este finită. Din nefericire, dacă funcţia de pondere d este arbitrară, condiţia ca $R_{\mathcal{A}}$ să fie finită este la fel de dificilă ca şi rezolvarea exactă a **TSP**. Mai precis, avem următorul rezultat:

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

Teorema 3

Dacă există un algoritm de aproximare în timp polinomial A pentru **TSP** a. î. $R_{\mathcal{A}} < \infty$, atunci problema **CH** poate fi rezolvată în timp polinomial.

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

Demonstraţie. Fie A un algoritm de aproximare în timp polinomial cu $R_{\mathcal{A}} < \infty$. Există $k \in \mathbb{N}$ astfel încât $R_{\mathcal{A}} \leqslant k$.
Fie $G = (V, E)$ un graf arbitrar, instanţă a **CH**. Dacă $n = |V|$, atunci considerăm $d : E(K_n) \to \mathbb{N}$ definită prin

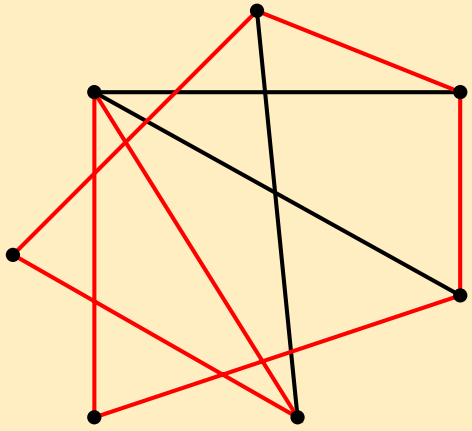
$$d(uv) = \begin{cases} 1, & \text{dacă } uv \in E(G) \\ kn, & \text{dacă } uv \notin E(G) \end{cases}.$$

Evident, G este Hamiltonian dacă şi numai dacă H_o, soluţia optimă a acestei instanţe a **TSP**, satisface $d(H_o) = n$.

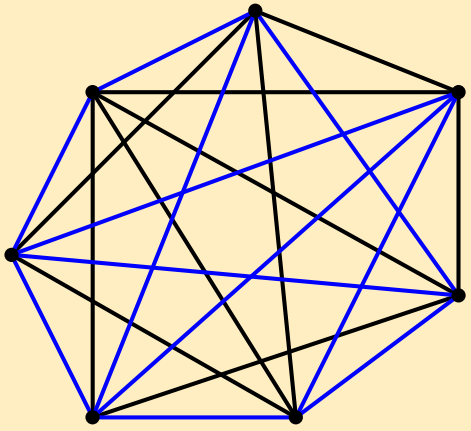
Aplicăm $\mathcal{A}$ pentru a rezolva aproximativ această instanţă a **TSP**.

- Dacă $d(H_{\mathcal{A}}) \leqslant kn$, atunci $d(H_{\mathcal{A}}) = n$ şi $H_{\mathcal{A}} = H_0$.
- Dacă $d(H_{\mathcal{A}}) > kn$, atunci $d(H_0) > n$. Într-adevăr, presupunând că $d(H_0) = n$, avem $\dfrac{d(H_{\mathcal{A}})}{d(H_0)} \leqslant k$, astfel $d(H_{\mathcal{A}}) \leqslant kd(H_0) = kn$, contradicţie.

Urmează că G este Hamiltonian dacă şi numai dacă $d(H_{\mathcal{A}}) \leqslant kn$, şi, deoarece A rulează în timp polinomial, urmează că **CH** poate fi rezolvată în timp polinomial. $\square$



G



weight $7k$
weight 1

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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 $\mathbf{P} \neq \mathbf{NP}$, atunci nu există algoritm de aproximare în timp polinomial $\mathcal{A}$ 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 *

Teorema 4

(Christofides,1976) Dacă in **TSP** funcţia de pondere d satisface

$$\forall u, v, w \in V(K_n) \text{ distincte, } d(uv) \leqslant d(uw) + d(wv),$$

atunci există un algoritm de aproximare în timp polinomial $\mathcal{A}$ cu $R_{\mathcal{A}} = 3/2$.

Demonstraţie. Fie $\mathcal{A}$ următorul algoritm:

- Determină T^0, mulţimea de muchii a unui arbore parţial 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 indus în K_n de mulţimea de noduri de grad impar din arborele parţial de cost minim T^0 (acest pas ia un timp polinomial folosind orice algoritm pentru determinarea unui cuplaj 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

Demonstraţie (continuare)

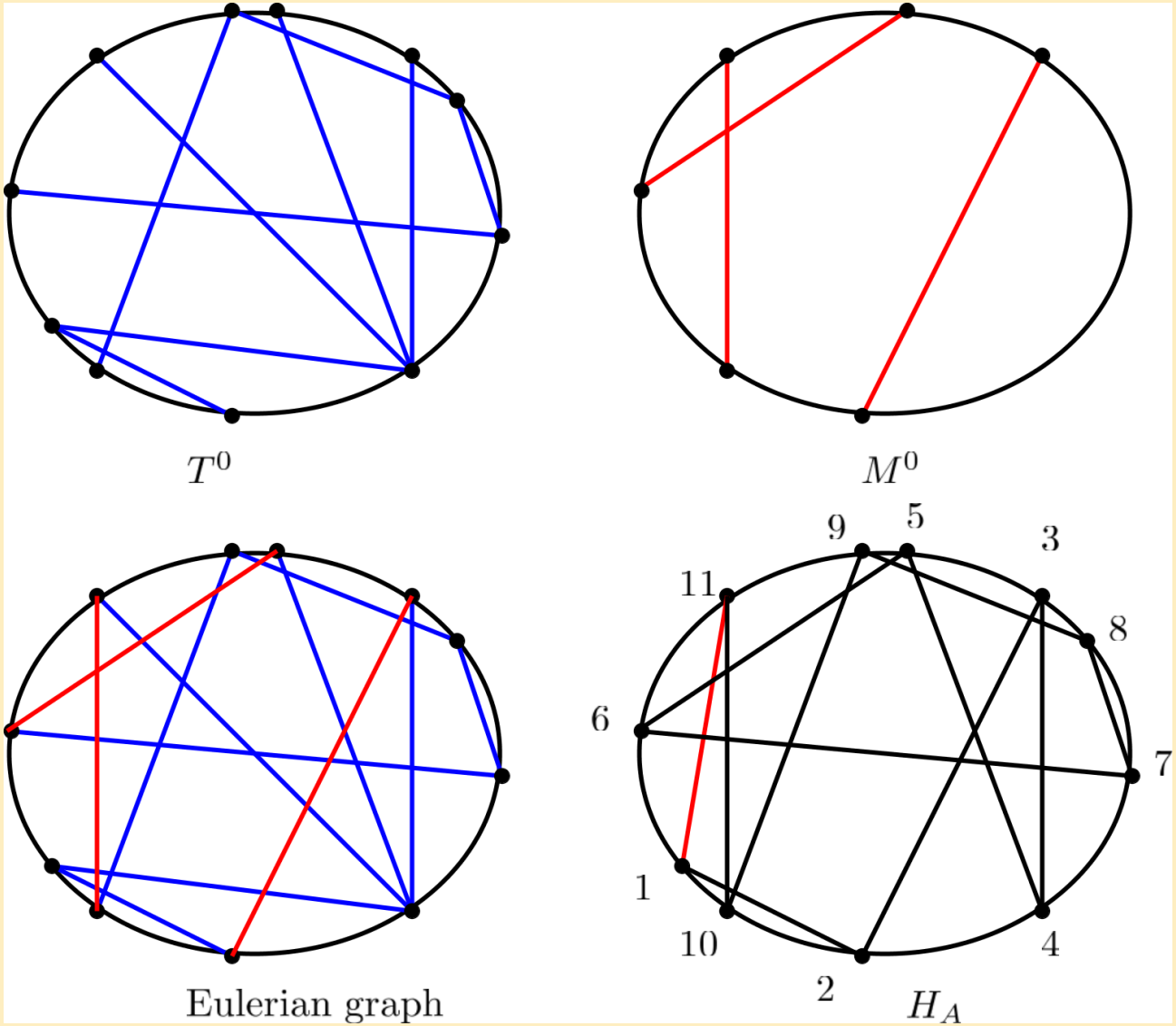
- În multigraful obţinut din $\langle T^0 \cup M^0 \rangle_{K_n}$, prin duplicarea muchiilor din $T^0 \cap M^0$ (este conex şi are toate nodurile de grad par) determină un parcurs Eulerian închis, $(v_{i_1}, v_{i_2}, \ldots, v_{i_1})$. Eliminăm toate apariţiile multiple ale nodurilor interne pentru a obţine un circuit Hamiltonian $H_{\mathcal{A}}$ in K_n cu mulţimea de muchii $H_{\mathcal{A}} = \{v_{j_1} v_{j_2}, v_{j_2} v_{j_3}, \ldots, v_{j_n} v_{j_1}\}$ (amândouă construcţiile se fac în $\mathcal{O}(n^2)$).

Fie $H_{\mathcal{A}}$ soluţia aproximativă a **TSP** dată de algoritmul lui Christofides. Fie $m = \lfloor n/2 \rfloor$ şi H_o o soluţie optimă. Arătăm (după Cornuejols & Nemhauser) că

$$\forall n \geqslant 3, d(H_{\mathcal{A}}) \leqslant \frac{3m-1}{2m} d(H_o).$$

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

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



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

Demonstraţie (continuare) Fie $H_o = \{v_1v_2, v_2v_3, \ldots, v_nv_1\}$ (dacă e necesar, redenumim nodurile).

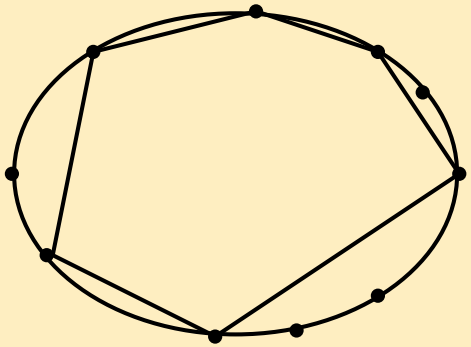
Fie $W = \{v_{i_1}, v_{i_2}, \ldots, v_{i_{2k}}\}$ mulţimea nodurilor de grad impar din $\langle T^0 \rangle_{K_n}$, $i_1 < i_2 < \ldots < i_{2k}$. Fie $H = \{v_{i_1}v_{i_2}, v_{i_2}v_{i_3}, \ldots, v_{i_{2k-1}}v_{i_{2k}}, v_{i_2k}v_{i_1}\}$ circuitul generat de W în K_n. Aplicând în mod repetat inegalitatea triunghiulară, obţinem $d(H) \leqslant d(H_o)$, (ponderea fiecărei corzi, $d(v_{i_j}v_{i_{j+1}})$ este majorată de suma ponderilor de pe muchiile lui H_o care unesc extremităţile corzii $v_{i_j}v_{i_{j+1}}$).

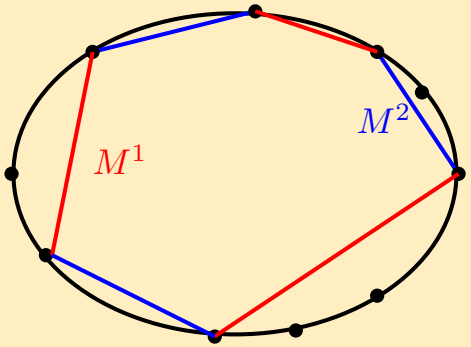
Deoarece H este un circuit par, poate fi scris ca reuniunea a două cuplaje perfecte în $[W]_{K_n}$, $M^1 \cup M^2$. Să presupunem că $d(M^1) \leqslant d(M^2)$.

Din modul de alegere al lui M^0, avem $d(M^0) \leqslant d(M^1) \leqslant (1/2)[d(M^1) + d(M^2)] = (1/2)d(H) \leqslant (1/2)d(H_o)$. Fie $\alpha \in \mathbb{R}_+$ a. î. $d(M^0) = \alpha d(H_o)$. Evident, $0 < \alpha \leqslant 1/2$.

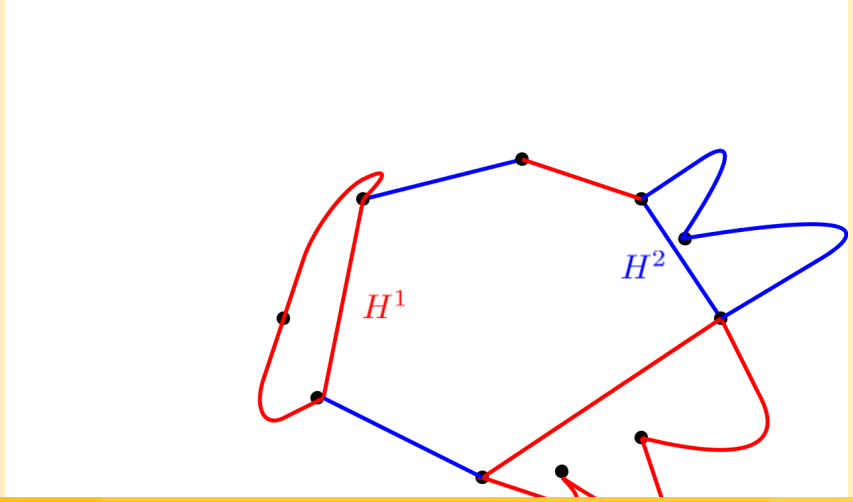
Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie (continuare)





Descompunem H_o în $H^1 \cup H^2$ luând în H^i muchiile din H_o care conectează extremităţile fiecărei corzi din M^i: $(v_{i_j} v_{i_{j+1}} \in M^i \Rightarrow v_{i_j} v_{i_j+1}, \ldots v_{i_{j+1}-1} v_{i_{j+1}} \in H^i)$.



G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms

Demonstraţie (continuare) Din inegalitatea triunghiulară, $d(H^i) \geqslant d(M^i)$, $i = 1, 2$. Cel puţin 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 $d(H^1) \geqslant d(M^1) \geqslant d(M^0) = \alpha d(H_o)$, urmează că există $e \in H^1$ astfel încât $d(e) \geqslant (\alpha/m) d(H_o)$.

Fie T un arbore parţial de cost minim obţinut din H_o prin ştergerea unei muchii de pondere maximă. Avem $d(T) = d(H_o) - \max\limits_{e \in E(H_o)} d(e) \leqslant d(H_o) - (\alpha/m) d(H_o)$. Deoarece T^0 este arbore parţial de cost minim în K_n urmează că $d(T^0) \leqslant d(H_o)(1 - \alpha/m)$.

Folosind inegalitatea triunghiulară obţinem

$$d(H_{\mathcal{A}}) \leqslant d(T^0) + d(M^0) \leqslant d(H_o)\left(1 - \frac{\alpha}{m}\right) + \alpha d(H_o) =$$

$$= \left(1 + \frac{\alpha(m-1)}{m}\right) d(H_o) \overset{\alpha \leqslant 1/2}{\leqslant} \frac{3m-1}{2m} d(H_o), \forall n \geqslant 3. \ \square$$

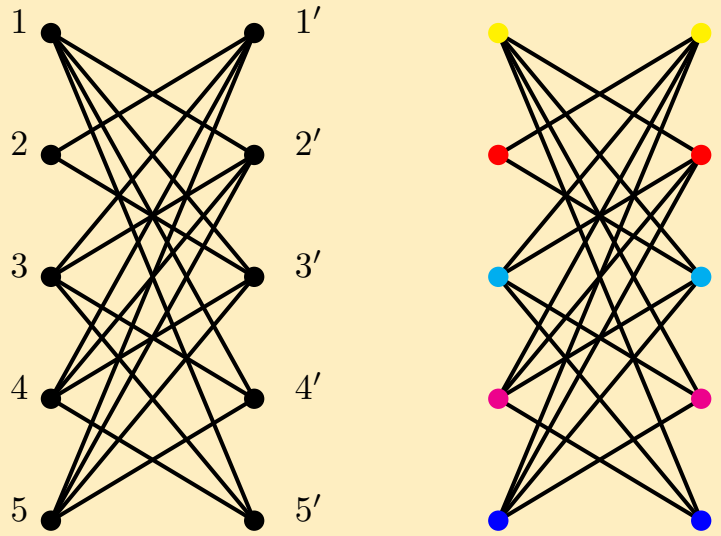
Fie $G = (V, E)$ un graf, $V = \{1, 2, \ldots, n\}$ şi fie π o permutare a lui V. Construim o colorare a nodurilor $c : V \to \{1, \ldots, \chi(G, \pi)\}$.

$c(\pi_1) \leftarrow 1$; $\chi(G, \pi) \leftarrow 1$; $S_1 \leftarrow \{\pi_1\}$;
for $(i = \overline{2, n})$ **do**
  $j \leftarrow 0$;
  repeat
    $j++$; $v \leftarrow$ primul nod (în π), 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

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

Să observă m că numărul de culori folosite de algoritm nu este mai mare decât $1 + \Delta(G)$. Urmează că $\chi(G) \leqslant 1 + \Delta(G)$.

$\chi(G, \pi)$ numărul de culori returnate de algoritmul de colorare greedy, poate fi cu oricât mai mare decât $\chi(G)$. De exemplu, fie G graful obţinut din graful complet bipartit $K_{n,n}$, cu mulţimea de noduri $\{1, 2, \ldots, n\} \cup \{1', 2', \ldots, n'\}$, prin ştergerea muchiilor $11', 22', \ldots, nn'$.



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

Dacă $\pi = (1, 1', 2, 2', \ldots, n, n')$, atunci algoritmul de colorare greedy returnează $c(1) = c(1')$, $c(2) = c(2') = 2$, $c(n) = c(n') = n$. Astfel, $\chi(G, \pi) = n$, pe când $\chi(G) = 2$.

Pe de altă parte, pentru orice graf G, există o permutare π a nodurilor sale astfel încât $\chi(G, \pi) = \chi(G)$.

Într-adevăr, fie $S_1, S_2, \ldots, S_{\chi(G)}$ clasele de colorare ale unei colorări optimale, astfel încât fiecare S_i este o mulţime 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, $\chi(G, \pi) = \chi(G)$.

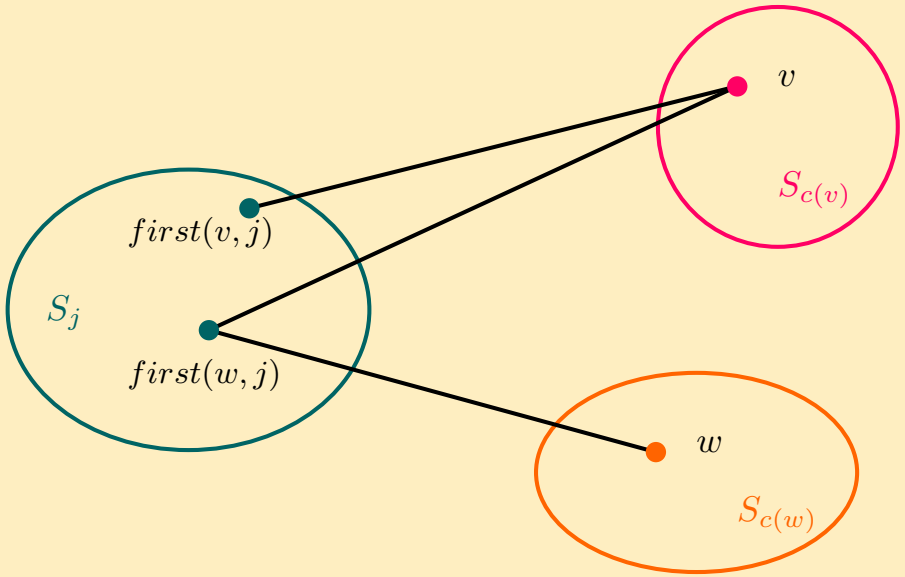
O condiţie suffcientă pentru corectitudinea algoritmului de colorare greedy:

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, $\forall vw \in E$ şi $\forall j < \min\{c(v), c(w)\}$ 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 *

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

Condiţia de mai sus poate fi testată în $\mathcal{O}(m)$ ca un ultim pas al algoritmului. Dacă este îndepinită, atunci avem un certificat pentru optimalitatea numărului găsit de culori.

Teorema de mai sus se poate demonstra arătând că dacă condiţia enunţată are loc, atunci $\chi(G,\pi) = \omega(G) \leqslant \chi(G)$. Pe de altă parte, se ştie 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 condiţia 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 *

Exerciţii pentru seminarul din următoarea săptămână

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Exerciţiul 1. Regiunile (inclusiv cea infinită) formate de n cercuri în plan pot fi colorate cu două culori astfel ca două regiuni care au în comun un arc de cerc (nedegenerat) să fie colorate diferit.

Exerciţiul 2. Fie $G = (V, E)$ un graf fără circuite impare disjuncte. Arătaţi că G este 5-colorabil.

Exerciţiul 3. Pentru un graf dat H definim **gradul mediu** ca $ad(H) = \frac{2|E(H)|}{|V(H)|}$. Pentru un graf G, **gradul mediu maxim** este

$$mad(G) = \max\{ad(H) \ : \ H \text{ subgraf indus al lui } G\}$$

Fie $k \geqslant 3$ un întreg. Arătaţi că, dacă un graf G are numărul cromatic strict mai mare decât k iar gradul său mediu maxim cel mult k, atunci G conţine un subgraf indus k-regulat.

Exerciţiul 4.

(a) Arătaţi că orice graf poate colorat (pe noduri) cu $\Delta(G) + 1$ culori.

(b) Considerăm următorul algoritm recursiv pentru colorarea nodurilor unui graf 3-colorabil cu n noduri:

color(G) {
if ($\Delta(G) \leqslant \sqrt{n}$) **then**
colorează toate nodurile lui G cu $(\Delta(G) + 1)$ culori noi;
else
fie $x_0 \in V(G)$ a. î. $d_G(x_0) = \Delta(G)$;
colorează x_0 cu o culoare nouă;
colorează nodurile lui $[N_G(x_0)]_G$ cu două culori noi;
$G \leftarrow G - (\{x_0\} \cup N_G(x_0))$;
color(G);
end if }

Arătaţi că algoritmul de mai sus foloseşte $\mathcal{O}(\sqrt{n})$ culori.

Exerciţiul 5. Considerăm următoarea problemă:

Set Cover

Instanţă: O mulţime nevidă X, o familie de submulţimi ale lui X: $\mathcal{F} = \{X_1, \ldots, X_n\}$ şi $k \in \mathbb{N}^*$.

Întrebare: X poate fi acoperit cu cel mult k submulţimi din $\mathcal{F}$?

Considerăm următoarea heuristică (de tip greedy) pentru rezolvarea acestei probleme

$\mathcal{F}' \leftarrow \emptyset$;
while ($\exists x \in X$ neacoperit de $\mathcal{F}'$) **do**
  fie X_i din $\mathcal{F}$ cu un număr maxim de elemente neacoperite;
  $\mathcal{F}' \leftarrow \mathcal{F}' \cup \{X_i\}$;
end while
return $\mathcal{F}'$;

Arătaţi că dacă m este cardinalul unei subfamilii optime a lui $\mathcal{F}$, atunci algoritmul de mai sus returnează o subfamilie cu cel mult $m \ln n$ submulţimi ($n = |X|$).

Exerciţii pentru seminarul din următoarea săptămână

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Exerciţiul 6. Fie $G = (V, E)$ un graf cu n noduri şi m muchii. O ordonare $\{x_1, x_2, \ldots, x_n\}$ a nodurilor lui G este **k-mărginită** dacă în digraful $\vec{G}$, obţinut din G prin înlocuirea fiecărei muchii $x_i x_j$ cu arcul $x_{\min\{i,j\}}, x_{\max\{i,j\}}$, avem $d^+_{\vec{G}}(x) \leqslant k, \forall\, x \in V$.

(a) Descrieţi un algoritm care să testeze în $\mathcal{O}(n + m)$ dacă G, are o ordonare k-mărginită, pentru un $k \in \mathbb{N}^*$.

(b) Folosiţi algoritmul de mai sus pentru a determina în $\mathcal{O}((n + m)\log n)$ numărul

$$o(G) = \min\{k \in \mathbb{N} \,:\, G \text{ are o ordonare } k\text{-mărginită.}\}$$

(c) Arătaţi că orice graf G are o colorare a nodurilor cu $o(G)+1$ culori.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciţii pentru seminarul din următoarea săptămână

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Exerciţiul 7. Un algoritm greedy pentru colorarea nodurilor graflui $G = (V, E)$ este următorul: mai întâi, alege o D-ordonare a nodurilor $V = \{v_1, v_2, \ldots, v_n\}$, i. e., $d_G(v_1) \geqslant d_G(v_2) \geqslant \ldots \geqslant d_G(v_n)$, apoi v_i primeşte cea mai mică culoare nefolosită de vecinii săi deja colorаţi. Considerăm următoarea problemă de decizie

3GCOL

instanţă: $G = (V, E)$ un graf.

întrebare: Are G o D-ordonare a. î. euristica de mai sus foloseşte cel mult 3 culori?

Arătaţi că **3COL** $\leqslant_P$ **3GCOL**.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Exerciţiul 8. Fie $G = (V; E)$ un graf cu $V = \{1, 2, \ldots, n\}$ şi $\omega(G) = 2$. Definim un graf nou $M(G)$ considerând reuniune disjunctă a lui G cu $K_{1,n}$ (a cărui bipartiţie este $(\{0\}, \{1', 2', \ldots, n'\})$) şi adăugând toate muchiile $\{i'j, ij' \ : \ ij \in E(G)\}$.

(a) Arătaţi că $\omega(M(G)) = 2$ şi $\chi(M(G)) = \chi(G) + 1$.

(b) Arătaţi că, pentru orice $p \in \mathbb{N}^*$, există un graf K_3-free cu numărul cromatic p.

Exerciţiul 9. Fie $\mathcal{S}$ o societate formată din n indivizi. Fiecare individ, $i \in \mathcal{S}$, cunoaşte o submulţime $c(i) \subseteq \mathcal{S} \setminus \{i\}$ formată din alţi indivizi. Doi indivizi diferiţi $i, i' \in \mathcal{S}$ nu pot face parte din acelaşi juriu dacă unul dintre ei îl cunoaşte pe celălalt (putem avea jurii formate dintr-un singur membru).

Arătaţi că dacă fiecare individ cunoaşte cel mult k alţi indivizi ($|c(i)| \leqslant k$, $\forall i \in \mathcal{S}$), atunci există o familie de cel mult $(2k + 1)$ jurii disjuncte care acoperă întreaga societate.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Cuprins

1 Grafuri planare
- Proprietăţi elementare
- Desenarea grafurilor 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 Algorithms

Fie $G = (V, E)$ un graf şi $\mathcal{S}$ o suprafaţă (e.g., plan, sferă) din $\mathbb{R}^3$. O **reprezentare** a lui G pe $\mathcal{S}$ este un graf $G' = (V', E')$ astfel încât:

a) $G \cong G'$;

b) V' este o mulţime puncte distincte ale lui $\mathcal{S}$;

c) Orice muchie $e' \in E'$ este o curbă simplă (arc Jordan) conţinută în $\mathcal{S}$ unindu-i extremităţile;

d) Orice punct din $\mathcal{S}$ este fie un nod al lui G' fie este conţinut în cel mult o muchie a lui G'.

Dacă $\mathcal{S}$ este un plan, atunci G este un **graf planar** şi G' este o **reprezentare planară** a lui G.
Dacă $\mathcal{S}$ este un plan şi G' este un graf satisfăcând constrângerile b), c) şi d) de mai sus, atunci G' se numeşte **graf plan**.

Lemă

Un graf este planar dacă şi numai dacă are o reprezentare pe o sferă.

Demonstraţie. Dacă G este planar, fie G' o reprezentare planară a lui G în planul π. Luăm un punct x în π şi considerăm o sferă $\mathcal{S}$ tangentă la π în x. Fie y punctul diametral opus lui x în $\mathcal{S}$. Considerăm $\varphi : \pi \to \mathcal{S} \setminus \{y\}$ dată prin $\varphi(M) =$ punctul diferit de y în care dreapta My intersectează sfera, $\forall M \in \pi$. φ este o bijecţie şi astfel $\varphi(G')$ este o reprezentare a lui G pe $\mathcal{S}$.

Reciproc, dacă G are o reprezentare pe o sferă $\mathcal{S}$: luăm un punct y în $\mathcal{S}$[a], considerăm x, punctul diametral opus lui y pe $\mathcal{S}$, construim un plan tangent π la $\mathcal{S}$ în x, şi definim $\psi : \mathcal{S} \to \pi$ by $\psi(M) =$ punctul în care dreapta yM intersectează planul π, pentru orice $M \in \mathcal{S}$. Imaginea prin ψ a reprezentării lui G pe sferă, $\psi(G)$, este reprezentarea planară dorită a lui G. $\square$

[a] y se alege a. î. să nu se afle pe vreo muchie sau în vreun nod al lui G.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Fie G un **graf plan**. Dacă ştergem punctele lui G din plan (nodurile şi 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 conţinută în acea regiune), care sunt numite **feţele** lui G.

Exact una dintre aceste feţe este nemărginită şi este numită faţa **exterioară**.

Fiecare faţă este caracterizată de mulţimea muchiilor care-i formează **frontiera**. Fiecare circuit al lui G împarte planul în exact două regiuni conexe, astfel fiecare muchie a unui circuit aparţine la exact două frontiere (la exact două feţe).

Un graf planar poate avea diferite reprezentări planare.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

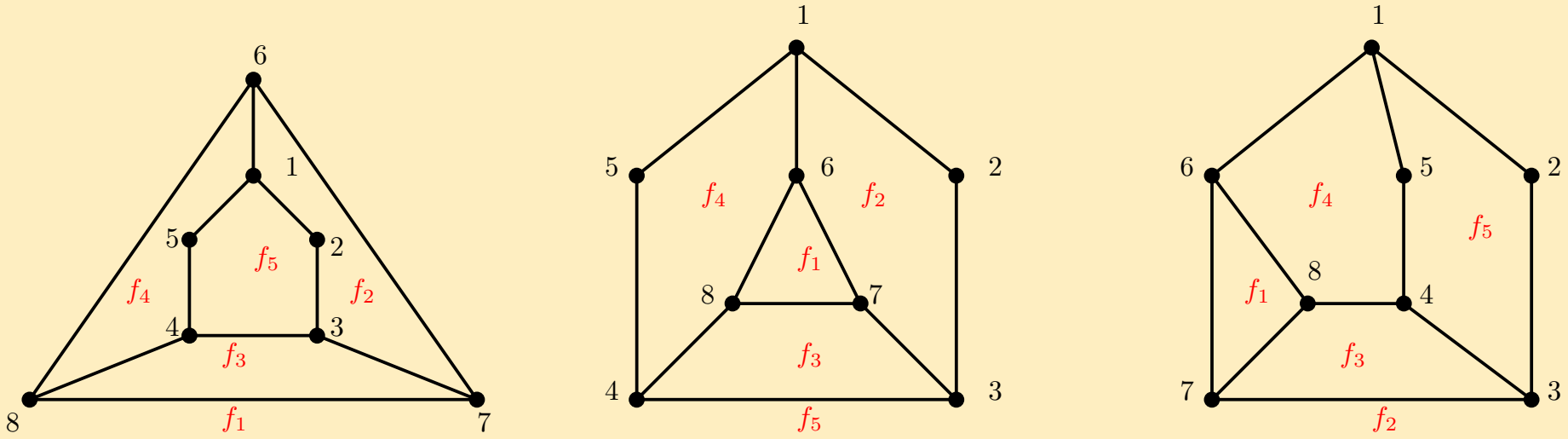
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Lemă

Orice reprezentare planară a unui graf planar poate fi transformată într-o reprezentare planară diferită în care o faţă fixată a primei reprezentări să devină faţa exterioară a celei de-a doua.

Demonstraţie. Fie G' o reprezentare planară a lui G şi F o faţă a lui G'. Fie G^0 o reprezentare a lui G' pe o sferă şi F^0 faţa a lui G^0 corespunzând lui F. Alegem un punct y în interiorul lui F^0, x punctul său diametral opus pe sferă, şi π planul tangent în x la sferă.
$G'' = \psi(G^0)$ este o reprezentare a lui G în planul π având ca faţă exterioară $\psi(F^0)$. $\square$

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms



C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

(Formula lui Euler) Fie $G = (V, E)$ un graf conex plan cu n noduri, m muchii şi f feţe. Atunci,

$$\mathbf{f = m - n + 2.}$$

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Demonstraţie. Inducţie după f.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie (continuare). Dacă $f = 1$, atunci G nu are circuite şi, deoarece este conex, este un arbore. Urmează că $m = n - 1$ şi teorema este adevărată.

În pasul inductiv să presupunem că teorema are loc pentru orice graf conex plan cu mai puţin de $f(\geqslant 2)$ feţe. Fie e o muchie de pe un circuit al lui G (există un astfel de circuit, deoarece $f \geqslant 2$). Atunci e aparţine frontierei a exact două feţe a lui G. Urmează că $G_1 = G - e$ este un graf conex plan cu n noduri, $m - 1$ muchii şi $f - 1$ feţe. 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 şi n este numărul de noduri, atunci $m = \mathcal{O}(n)$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Corolarul 1

Fie $G = (V, E)$ un graf conex planar cu $n \geqslant 3$ noduri şi m muchii. Atunci,

$$\mathbf{m \leqslant 3n - 6.}$$

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Demonstraţie. Fie G' o reprezentare planară a lui G. Dacă G' are doar o faţă, atunci G este un arbore, $m = n - 1$, şi pentru $n \geqslant 3$ inegalitatea are loc.

Dacă G' are cel puţin două feţe, atunci fiecare faţă F a lui G' are în frontiera sa muchiile unui circuit C_F, şi fiecare astfel de muchie aparţine la exact două feţe. Orice circuit al lui G' are cel puţin trei muchii, astfel

$$2m \geqslant \sum_{F \text{ faţă a lui } G'} length(C_F) \geqslant \sum_{F \text{ faţă a lui } G'} 3 = 3f = 3(m-n+2),$$

Adică inegalitatea dorită. $\square$

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Remarcă

Graful K_5 nu este planar (numărul său de noduri este $n = 5$, numărul său de muchii este $m = 10$ şi $10 > 3 \cdot 5 - 6$).

Corolarul 2

Fie $G = (V, E)$ un graf bipartit, planar şi conex cu $n \geqslant 3$ noduri şi $m \geqslant 3$ muchii. Atunci,

$$\mathbf{m \leqslant 2n - 4.}$$

Demonstraţie. Aceeaşi demonstraţie ca pentru Corolarul 1, dar folosind faptul că orice circuit al lui G' are cel puţin patru muchii. □

Remarcă

Graful $K_{3,3}$ nu este planar (numărul său de noduri este $n = 6$, numărul său de muchii este $m = 9$ şi $9 > 2 \cdot 6 - 4$).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Corolarul 3

Dacă $G = (V, E)$ este un graf conex planar, atunci există $v_0 \in V$ astfel încât

$$\mathbf{d_G(v_0) \leqslant 5.}$$

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

Demonstraţie. Putem să presupunem că G are cel puţin două muchii (altfel e banal). Fie G' o reprezentare planară a lui G cu n noduri şi m muchii. Dacă notăm cu n_i numărul de noduri de grad i $(1 \leqslant i \leqslant n-1)$ atunci

$$\sum_{i=1}^{n-1} i \cdot n_i = 2m \leqslant 2(3n-6) = 6\left(\sum_i n_i\right) - 12 \Rightarrow \sum_i (i-6)n_i + 12 \leqslant 0.$$

Dacă am avea $i \geqslant 6$, pentru orice i, toţi termenii din această sumă sunt $\geqslant 0$, deci există $i_0 \leqslant 5$ astfel încât $n_{i_0} > 0$. $\square$

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Fie $G = (V, E)$ un graf şi $v \in V$ astfel încât $d_G(v) = 2$ şi $vw_1, vw_2 \in E$, $w_1 \neq w_2$.
Fie $h(G) = (V \setminus \{v\}, E \setminus \{vw_1, vw_2\} \cup \{w_1 w_2\})$.

Lemă

G este planar dacă şi numai dacă $h(G)$ este planar.

Demonstraţie. "$\Leftarrow$" Presupunem că $h(G)$ e planar.
Dacă $w_1 w_2 \notin E$, atunci pe curba simplă care uneşte punctele corespunzând lui w_1 şi 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 feţele reprezentării planare a lui $h(G)$ şi "unim" acest punct nou cu punctele corespunzând lui w_1 şi w_2 prin curbe simple care nu le intersectează pe cele deja existente.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie (continuare) "$\Rightarrow$" Reciproc, presupunem că G este planar.

În reprezentarea sa planară, ştergem punctul corespunzând lui v şi cele două curbe corespunzând muchiilor vw_1 şi vw_2 sunt înlocuite cu reuniunea lor; dacă $w_1w_2 \in E$, atunci curba simplă care-i corespunde este ştearsă. $\square$

Notăm cu $h^*(G)$ graful obţinut din G prin aplicarea repetată a transformării h până se obţine un graf fără noduri de grad 2.

Urmează că G este planar dacă şi numai dacă $h^*(G)$ este planar.

Două grafuri G_1 şi G_2 sunt **homeomorfe** dacă $h^*(G_1) \cong h^*(G_2)$.

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Teoremă

(Kuratowski, 1930) Un graf este planar dacă şi numai dacă nu conţine subgrafuri homeomorfe cu K_5 sau cu $K_{3,3}$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

(Fary, 1948, independent Wagner & Stein) Orice graf planar are o reprezentare planară cu toate muchiile segmente de dreaptă (**reprezentare Fary**).

Problemă: Să se determine o reprezentare Fary cu punctele reprezentând nodurile de coordonate întregi şi aria suprafeţei 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 $[0, 2n-4] \times [0, n-2]$ şi cu toate muchiile segmente de dreaptă.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie algoritmică. Vom schiţa o desenare în $\mathcal{O}(n \log n)$. Fără a restrânge generalitatea, vom presupune că G este **maximal planar**: $\forall e \in E(G)$, $G + e$ nu este planar (adăugăm muchii la G pentru a-l face maximal planar şi când aceste muchii (segmente) vor fi desenate le facem invizibile). Observăm că orice faţă a unui graf maximal planar este un triunghi şi are $3n-6$ muchii, unde n este numărul său de noduri.

Lema 1

Fie G un graf planar şi 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 $w \neq u, v$ şi nu există nicio coardă interioară a lui C' cu o extremitate în w.

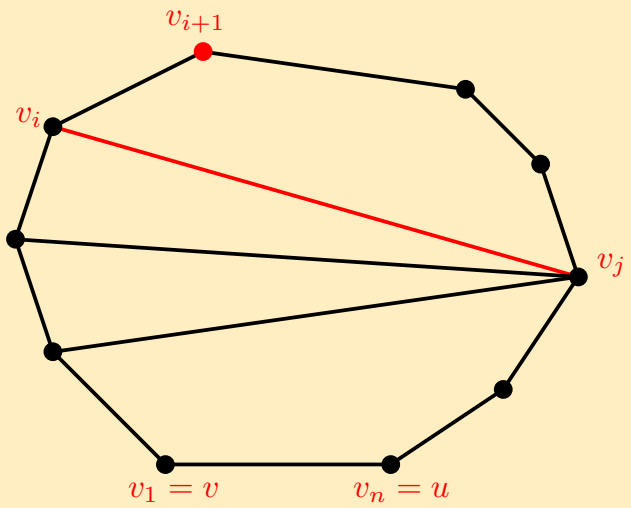
Demonstraţie. Fie $v_1, v_2, \ldots, v_n$ nodurile lui C' întâlnite într-o parcurgere de la u la v ($v = v_1$, $u = v_n$).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie (continuare). Dacă C' nu are corzi interioare, atunci lemma este adevărată. Altfel, alegem o pereche (i, j) astfel încât $v_i v_j$ este o coardă interioară a lui C' şi

$j - i = \min\{k - l \ : \ k > l + 1, v_k v_l \in E(G'), v_k v_l$ coardă interioară a lui C

Atunci, $w = v_{i+1}$ nu este incident cu o coardă interioară: $v_{i+1} v_p$ cu $i + 1 < p < j$ nu poate fi o coardă interioară - din modul de alegere a perechii (i, j), şi $v_{i+1} v_l$ cu $l < i$ sau $l > j$ nu este o coardă interioară deoarece ar trebui să intersecteze $v_i v_j$. $\square$



C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Lema 2

Fie G un graf maximal planar cu $n \geqslant 4$ noduri şi G' o reprezentare planară a lui G având faţa exterioară triunghiul u, v, w. Atunci, există o etichetare $v_1, v_2, \ldots, v_n$ a nodurilor lui G' astfel încât $v_1 = u$, $v_2 = v$, $v_n = w$ şi, pentru fiecare $k \in \{4, \ldots, n\}$, avem:

(i) Subgraful indus $G'_{k-1} = [\{v_1, \ldots, v_{k-1}\}]_G$ este 2-conex şi faţa sa exterioară este determinată de circuitul C'_{k-1} conţinând uv.

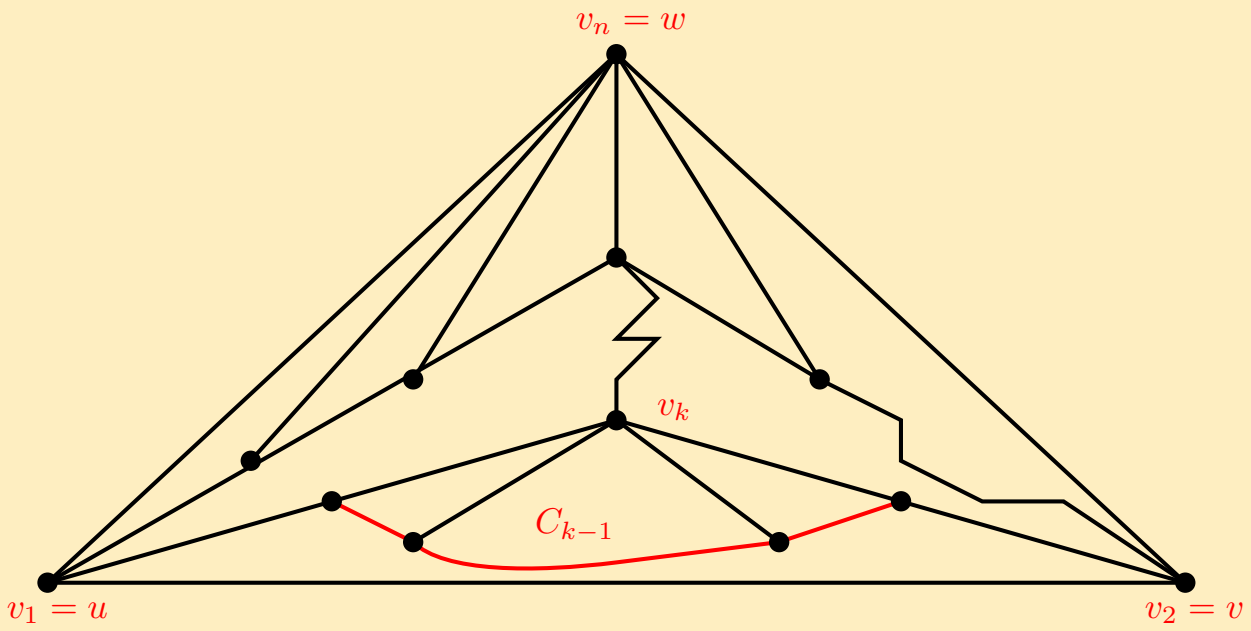
(ii) În subgraful indus G'_k nodul v_k este în faţa exterioară a lui G'_{k-1} şi $N_{G'_k}(v_k) \cap \{v_1, \ldots, v_{k-1}\}$ este un drum de lungime $\geqslant 1$ pe circuitul $C'_{k-1} - uv$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstraţie. Fie $v_1 = u$, $v_2 = v$, $v_n = w$, $G'_n = G$, $G'_{n-1} = G - v_n$.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms * G. Croitoru - Graph Algorithms



Demonstraţie (continuare). Observăm că $N_{G'_n}(w)$ este un circuit conţinând uv (după o sortare a lui $N_{G'_n}(w)$ pe coordonata x şi folosind planaritatea maximală). Urmează că i) şi ii) au loc pentru $k = n$.
Dacă v_k a fost deja ales ($k \leqslant n$) atunci în $G'_{k-1} = G' - \{v_n, \ldots, v_k\}$, vecinii lui v_k determină un circuit C'_{k-1} conţinând uv şi formând frontiera feţei exterioare a lui G'_{k-1}.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstraţie (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 construcţie, v_{k-1} nu este incident cu vreo coardă externă a lui C'_{k-1} (din planaritatea maximală). Urmează că G'_{k-2} conţine un circuit C'_{k-2} cu proprietăţile (i) şi (ii). $\square$

Demonstraţia Teoremei (Fraysseix, Pach, Pollack). Fie G un graf maximal planar cu n noduri, G' o reprezentare planară cu nodurile etichetate $v_1, \ldots, v_n$ ca în Lema 2, şi u, v, w faţa sa exterioară.
Vom construi o reprezentare Fary a lui G având nodurile puncte de coordonate întregi.
Presupunem că în pasul k ($\geqslant 3$) al construcţiei avem o astfel de reprezentare a lui G_k şi sunt îndeplinite următoarele trei condiţii:

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie (continuare).

(1) v_1 are coordonatele $(0, 0)$; v_2 are coordonatele $(i, 0)$, $i \leqslant 2k - 4$.

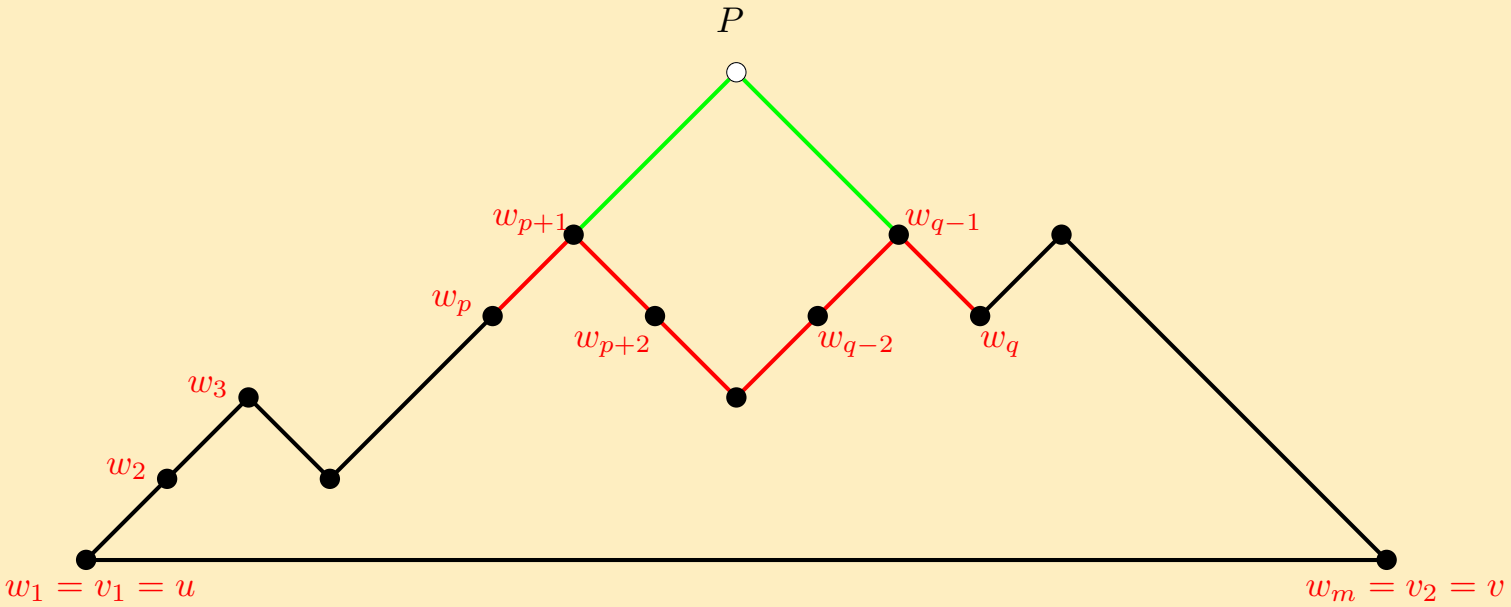
(2) Dacă $w_1, w_2, \ldots, w_m$ sunt nodurile circuitului care corespunde feţei exterioare a lui G_k, într-o parcurgere de la v_1 la v_2 ($w_1 = v_1$, $w_m = v_2$), atunci
$$x_{w_1} < x_{w_2} < \ldots < x_{w_m}.$$

(3) Muchiile $w_1w_2, w_2w_3, \ldots, w_{m-1}w_m$ sunt segmente de dreaptă paralele cu una dintre cele două bisectoare ale axelor de coordonate.

Condiţia (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 şi w_j au coordonate întregi).

Construcţia lui G'_{k+1}. Fie $w_p, w_{p+1}, \ldots, w_q$ vecinii din G'_k ai lui v_{k+1} în G'_{k+1} ($1 \leqslant p < q \leqslant m$).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms



Demonstraţie (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, $p \leqslant i \leqslant q$ astfel încât toare sunt distincte, atunci putem lua $v_{k+1} = P$ pentru a obţine o reprezentarea Fary a lui G_{k+1} cu toate noduri de coordonate întregi, satisfăcând condiţiile (1) - (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

Dacă segmentul $w_p w_{p+1}$ este paralel cu prima bisectoare, atunci translatăm către dreapta cu 1 toate nodurile lui G_k care au $x \geqslant x_{w_{p+1}}$. Facem o altă translaţie la dreapta cu 1 a tuturor nodurilor lui G_k având coordonata $x \geqslant x_{w_q}$.

Acum, toate segmentele $P'w_i$, cu $p \leqslant i \leqslant q$, sunt distincte, segmentele $w_i w_{i+1}$ cu $i = \overline{q, m-1}$ au pantele ± 1 şi de asemenea $w_p P'$ şi $P' w_q$ au pantele ± 1 (unde P' este intersecţia paralelei la prima bisectoare prin w_p cu paralela la cea de-a doua bisectoare prin w_q).

Luăm $v_{k+1} = P'$ şi pasul k al construcţiei 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 * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

(Tarjan & Lipton, 1979) *Fie G un graf planar cu n noduri. Există o partiţie (A, B, S) a lui $V(G)$ astfel încât:*

- *S separă A de B în G: $G - S$ nu conţine muchii de la A la B,*
- $|A| \leqslant (2/3)n$, $|B| \leqslant (2/3)n$,
- $|S| \leqslant 4\sqrt{n}$.

Această partiţie poate fi găsită în $\mathcal{O}(n)$.

Ideea demonstraţiei. Fie G un graf conex plan. Executăm o parcurgere **bfs** dintr-un nod oarecare s, etichetând fiecare nod v cu nivelul corespunzător din arborele bfs obţinut. Fie $L(t)$, mulţimea tuturor nodurilor de pe nivelul t, pentru $0 \leqslant t \leqslant l + 1$. Ultimul nivel $L(l + 1)$ este - din raţiuni tehnice - vid (ultimul nivel este în fapt l).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstraţie (continuare). Fiecare nivel intern este un separator în G (avem muchii doar între nivele consecutive). Fie t_1 nivelul de mijloc, adică nivelul care conţine cel de-al $\lfloor n/2 \rfloor$-lea nod întâlnit în timpul parcurgerii. Mulţimea $L(t_1)$ satisface:

$$\left| \bigcup_{t<t_1} L(T) \right| < \frac{n}{2} \text{ şi } \left| \bigcup_{t>t_1} L(T) \right| < \frac{n}{2}.$$

Dacă $|L(t_1)| \leqslant 4\sqrt{n}$, teorema este demonstrată.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.

Lemă

Există nivelele $t_0 \leqslant t_1$ şi $t_2 \geqslant t_1$ astfel încât $|L(t_0)| \leqslant \sqrt{n}$, $|L(t_2)| \leqslant \sqrt{n}$ şi $t_2 - t_0 \leqslant \sqrt{n}$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstraţie. Considerăm t_0 cel mai mare întreg care satisface $t \leqslant t_1$ şi $|L(t)| \leqslant \sqrt{n}$ (există un astfel de nivel deoarece $|L(0)| = 1$). Există t_2 un cel mai mic întreg care satisface $t > t_1$ şi $|L(t_2)| \leqslant \sqrt{n}$ (se observă că $|L(l+1)| = 0$).

Orice nivel dintre t_0 şi t_2 are mai mult de $\sqrt{n}$ noduri, deci numărul acestor nivele nu poate depăşi $\sqrt{n}$ (altfel, numărul de noduri ar fi $> n$). $\square$

Demonstraţie (continuare la Teorema separatorilor). Fie

$$C = \bigcup_{t<t_0} L(t), D = \bigcup_{t_0<t<t_2} L(t), E = \bigcup_{t>t_2} L(t).$$

- $|D| \leqslant (2/3)n$. Teorema are loc cu $S = L(t_0) \cup L(t_2)$, A mulţimea de cardinal maxim dintre C, D şi E, iar B reuniunea celor două mulţimi rămase (C şi E au cel mult $n/2$ elemente).

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie (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 obţine un separator de cardinal cel mult $4\sqrt{n}$, pentru A luăm reuniunea mulţimii de cardinal maxim dintre C şi E cu partea mai mică rămasă din D, şi pentru B luăm reuniunea celor două mulţimi rămase.

Separatorul pentru (graful indus de) D poate fi construit astfel: ştergem toate nodurile lui G care nu sunt din D mai puţin s care este unit cu toate nodurile de pe nivelul t_0+1. Graful obţinut se notează cu D şi este planar şi conex. Are un arbore parţial 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 obţine separatorul dorit. Detaliile (foarte interesante) sunt omise. $\square$

O aplicaţie 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 un graf G cu număr mic de noduri (pentru o constant c, putem verifica toate cele $\mathcal{O}(3^c) = \mathcal{O}(1)$ funcţii 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, $\mathcal{O}(n)$, partiţia (A, B, C) a nodurilor sale, cu $|A|, |B| \leqslant (2n/3)$ şi $|C| \leqslant \sqrt{n}$.

Se testează fiecare $3^{|C|} = 2^{\mathcal{O}(\sqrt{n})}$ funcţie posibilă de la C la $\{1, 2, 3\}$ dacă este o 3-colorare a subgrafului indus de C şi dacă această colorare poate fi extinsă la o 3-colorare a subgrafului indus de $A \cup C$ în G şi de asemeni la o 3-colorare a subgrafului indus de $B \cup C$ în G (recursiv).

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

O aplicaţie 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Timpul de execuţie, $T(n)$, al acestui algoritm, satisface recursia

$$T(n) = \begin{cases} \mathcal{O}(1), & \text{dacă } n \leqslant c, \\ \mathcal{O}(n) + 2^{\mathcal{O}(\sqrt{n})}\left(\mathcal{O}(\sqrt{n}) + 2T(2n/3)\right), & \text{dacă } n > c. \end{cases}$$

Urmează că $T(n) = 2^{\mathcal{O}(\sqrt{n})}$ (este posibil ca acele constante din spatele notaţiei $\mathcal{O}(\cdot)$ 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 *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Cuprins

1 Tree decomposition
- Tree width
- Small tree decompositions
- Proprietăţi ale tree descompunerilor
- Tree descompuneri cu rădăcină
- Aplicaţii

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Definiţie

O **tree decomposition** a unui graf $G = (V, E)$ este o pereche $\mathcal{T} = (T, \{V_t : t \in T\})$, unde T este un arbore iar $\{V_t : t \in V(T)\}$ este o familie de submulţimi de noduri ale lui G, $V_t \subseteq V, \forall\, t \in T$ astfel ca:

- *(Acoperirea nodurilor)* $V = \bigcup_{t \in V(T)} V_t$;
- *(Acoperirea muchiilor) Pentru orice muchie* $e \in E$, *ambele capete ale lui* e *sunt conţinute într-o mulţime* V_t, *pentru un anumit* $t \in V(T)$.
- *(Coerenţa) Fie* t_1, t_2, t_3 *trei noduri din* T *astfel ca* t_2 *se află pe drumul dintre* t_1 *şi* t_3 *in* T. *Atunci, dacă* $v \in V$ *se află în* V_{t_1} *şi* V_{t_3}, v *se află şi în* V_{t_2}.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Remarci

Proprietatea de coerenţă poate fi reformulată astfel:

- (Coerenţa') Fie $t_1, t_2, t_3 \in V(T)$ aşa încât t_2 aparţine drumului dintre t_1 şi t_2 în T. Atunci $V_{t_1} \cap V_{t_3} \subseteq V_{t_2}$.
- (Coerenţa") Pentru orice $x \in V$, subgraful lui T indus de $\{t \in V(T) : x \in V_t\}$ este (un subarbore al lui T) conex.

Mulţimile V_t ale descompunerii sunt numite **bags**.

Definiţie

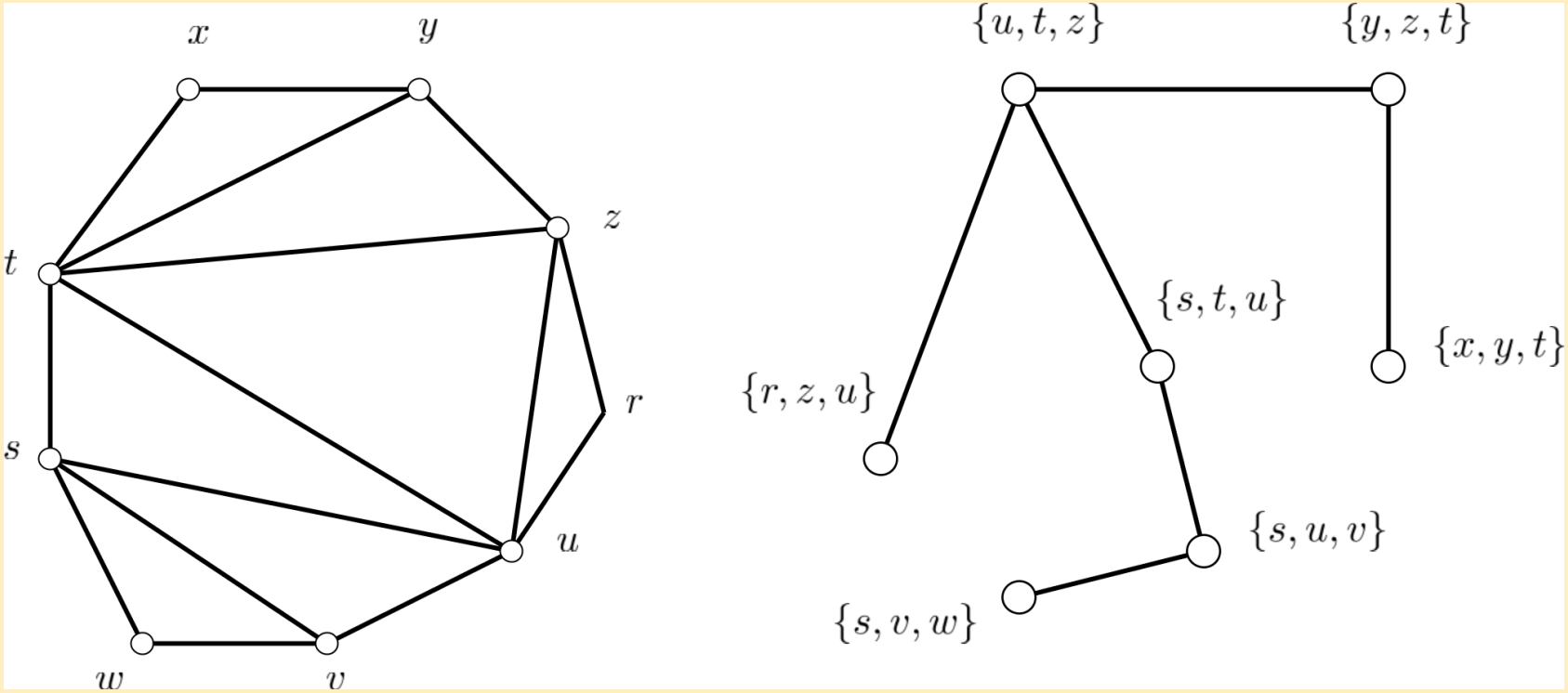
Fie $\mathcal{T} = (T, \{V_t : t \in T\})$ o tree-descompunere a lui G, **lăţimea** *(**width**) lui $\mathcal{T}$ este*

$$width(\mathcal{T}) = \max_{t \in V(T)} (|V_t| - 1).$$

Definiţie

Tree-width *a unui graf G, este cea mai mică lăţime a unei tree-descompuneri a lui G:*

$$tw(G) = \min\{width(\mathcal{T}) \ : \ \mathcal{T} \text{ tree-descompunere a lui } G\}.$$



C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Remarcă

$tw(G) = 0$ dacă şi numai dacă $E(G) = \varnothing$.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C.

Propoziţie

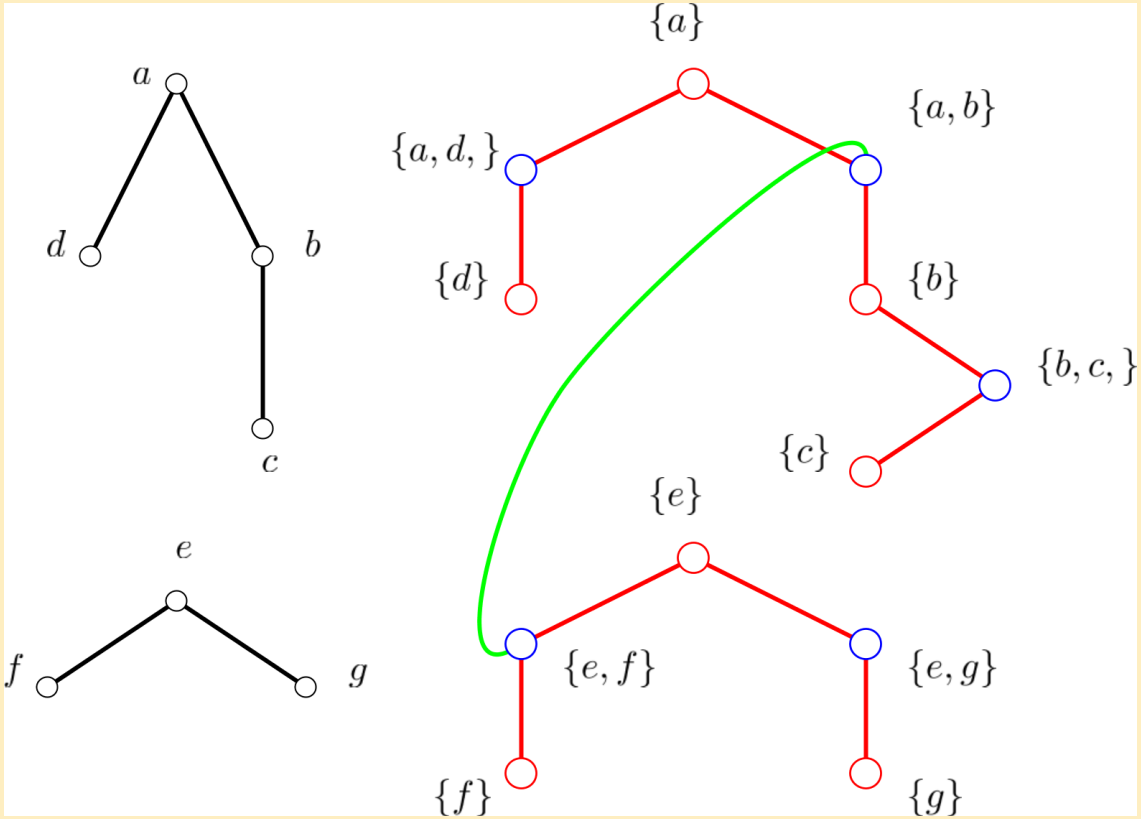
Dacă G este o pădure cu $E(G) \neq \varnothing$, atunci $tw(G) = 1$.

Demonstraţie. Din observaţia de mai sus $tw(G) \geqslant 1$. Dacă G este un arbore, atunci

- fie T obţinut din G prin redenumirea fiecărui nod $v \in V(G)$ prin t_v,
- inserează pe fiecare muchie $t_u t_v$ ($uv \in E(G)$) un vârf nou t_{uv},
- setează $V_{t_u} = \{u\}$ pentru orice t_u asociat lui $u \in V(G)$ şi $V_{t_{uv}} = \{u, v\}$ pentru orice $t_{uv} \in V(T)$ asociat lui $uv \in E(G)$.
- $(T, \{V_t : t \in V(T)\})$ este o tree-descompunere a lui G cu lăţimea 1.

Demonstraţie (continuare). O tree-descompunere a unei păduri cu p componente se poate obţine adăugând arbitrar $p-1$ muchii la tree-descompunerile componentelor (fără a crea circuite).

□



Small tree decompositions

Definiţie

O tree-descompunere, $\mathcal{T} = (T, \{V_t : t \in V(T)\})$, este **mică** *(***small***) dacă nu există noduri distincte $t_1, t_2 \in V(T)$ astfel ca $V_{t_1} \subseteq V_{t_2}$.*

Propoziţie

Dată o tree-descompunere a lui G, o tree-descompunere small a lui G de aceeaşi lăţime poate fi construitıa în timp polinomial.

Demonstraţie. Fie $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ o tree-descompunere a lui G cu $V_{t_1} \subseteq V_{t_2}$ pentru $t_1, t_2 \in V(T)$, $t_1 \neq t_2$. Putem presupune că $t_1 t_2 \in E(T)$ (altfel, putem determina două noduri adiacente cu această proprietate considerând drumul de la t_1 la t_2).
Contractând $t_1 t_2$ într-un nod nou t_{12} cu $V_{t_{12}} = V_{t_2}$, se obţine o tree-descompunere a lui G mai mică (conţine mai puţine perechi de noduri (t_1', t_2') cu $V_{t_1'} \subseteq V_{t_2'}$).

Demonstraţie (continuare). Repetăm această reducere până se obţine o tree-descompunere small.

□

Propoziţie

Dacă $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ *este o tree-descompunere small a lui* G*, atunci* $|T| \leqslant |G|$.

Demonstraţie. Prin inducţie după $n = |G|$. Dacă $n = 1$, atunci $|T| = 1$.
În pasul inductiv, pentru $n \geqslant 2$, considerăm o frunză t_1 a lui T cu vecinul t_2. $(T - t_1, \{V_t : t \in V(T - t_1)\})$ este o tree-descompunere small a lui $G' = G \setminus (V_{t_1} \setminus V_{t_2})$. Din ipoteza inductivă $|T - t_1| \leqslant |G'|$, astfel

$$|T| = |T - t_1| + 1 \leqslant |G'| + 1 \leqslant |G|.$$

□

Remarci

- dacă graful H se obţine din G prin contractarea unei muchii uv în z, atunci $tw(H) \leqslant tw(G)$: într-o tree-descompunere a lui G, inserăm z în fiecare bag care conţine pe u sau v şi ştergem apoi pe u şi v din orice bag pentru a obţine o tree-descompunere a lui H.
- dacă H este un subgraf al lui G, atunci $tw(H) \leqslant tw(G)$.

Definiţie

H este un **minor** *al unui graf G dacă se poate obţine din G prin ştergerea si contractarea succesivă a unor muchii din G.*

Corolar

Dacă H este un minor al grafului G, atunci $tw(H) \leqslant tw(G)$.

Demonstraţie. Folosind observaţiile de mai sus.

□

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - 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 $TW_k = \{G \ : \ tw(G) \leqslant k\}$.

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

TW (Tree-Width - versiunea de decizie)
Instanţă: G un graf şi $k \in \mathbb{N}$.
Întrebare: $G \in TW_k$?

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

Problema Tree-Width (versiunea de decizie) este o problemă **NP**-*completă.*

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Demonstraţie. Omisă.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Tree-width este FPT (fixed-parameter tractable)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Lemă

Pentru orice $k \in \mathbb{N}^*$, TW_k este o familie închisă la minori.

Teoremă

(Bodlaender) Pentru orice k fixat, problema determinării dacă G este în TW_k sau nu poate fi rezolvată în timpul $\mathcal{O}(f(k) \cdot n)$.

Proofs. Omisă. ($f(k)$ este o funcţie exponenţială în k.)

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru

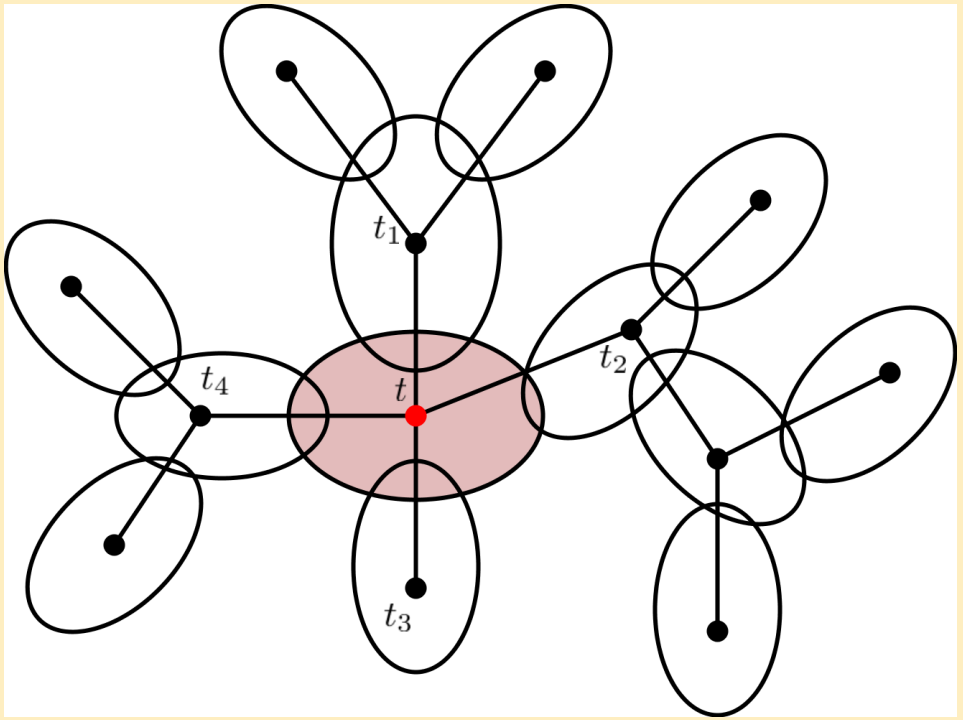
Notaţie: Fie $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ o tree-descompunere of G. Dacă T' este un subgraf al lui T, $G_{T'}$ este subgraful lui G indus de mulţimea de noduri $\bigcup_{t \in V(T')} V_t$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

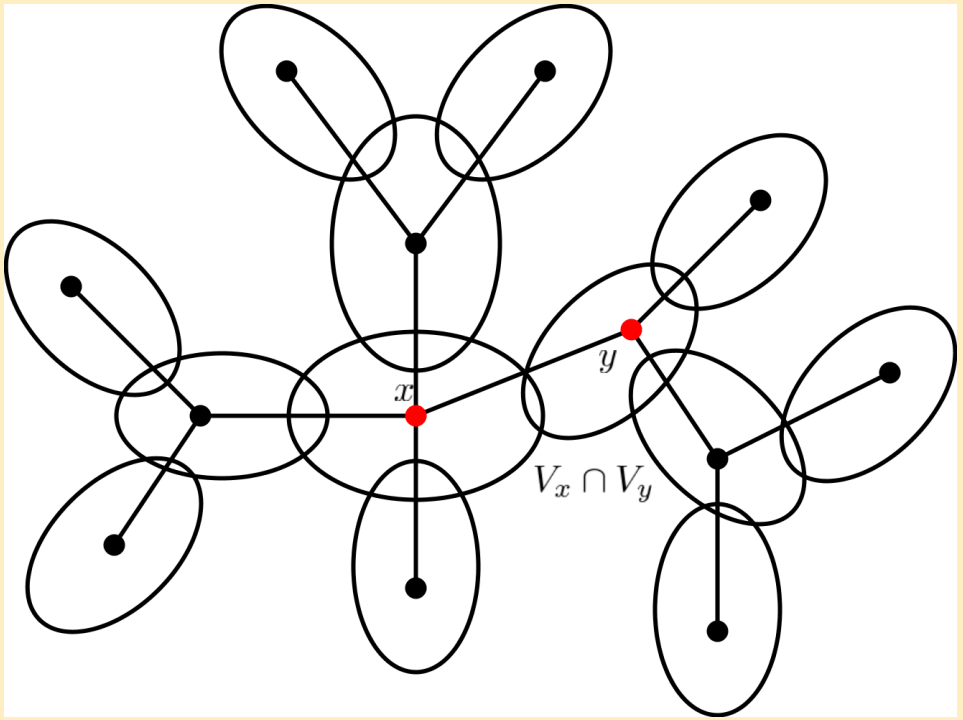
(Proprietatea separării nodurilor.) Presupunem că $T - t$ are componentele conexe $T_1, T_2, \ldots, T_p$. Atunci subgrafurile $G_{T_1} - V_t, G_{T_2} - V_t, \ldots, G_{T_p} - V_t$ nu au noduri în comun şi nici muchii între ele.

Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -



Teoremă

(Proprietatea separării muchiilor.) *Fie X şi Y componentele conexe ale lui T după ştergerea unei muchii $xy \in E(T)$. Atunci, ştergerea nodurilor din $V_x \cap V_y$ deconectează G în două subgrafuri $H_X = G_X - V_x \cap V_y$ şi $H_Y = G_Y - V_x \cap V_y$. Cu alte cuvinte H_X şi H_Y nu au noduri în comun şi nici muchii între ele.*



Proprietăţi ale tree descompunerilor

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Alte proprietăţi:

- Fie G un graf conex cu $tw(G) = k$, atunci sau $|G| = k + 1$ sau G are o mulţime separatoare de noduri de cardinal k.
- dacă $tw(G) = 1$, atunci G este o pădure.
- $tw(P_r \times P_s) = \min\{r, s\}$.
- $tw(K_n) = n - 1$.

Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru -

Definiţie

O **tree-descompunere cu rădăcină** *a lui G este o tree-descompunere $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ a lui G, în care un anumit nod r al lui T este declarat rădăcină.*

Notaţii: Fie t un nod dintr-o tree-descompunere cu rădăcină $\mathcal{T} = (T, \{V_t : t \in V(T)\})$.

- T_t este subarborele lui T cu rădăcina în t.
- $G[t]$ este subgraful lui G indus de nodurile din $\bigcup_{x \in V(T_t)} V_x$ (i. e., $G[t] = G_{T_t}$).

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Aplicaţii - Colorarea nodurilor

- Reamintim: o p-colorare a nodurilor unui graf $G = (V, E)$ este o funcţie $c : V \to \{1, 2, \ldots, p\}$ astfel ca $c(u) \neq c(v)$ pentru orice $uv \in E$.
- Fie H' şi H'' două subgrafuri ale lui G, cu p-colorările c' şi c'', respectiv. c'' este c'-compatibilă dacă pentru orice $v \in V(H') \cap V(H'')$ avem $c'(v) = c''(v)$.
- Fie $\mathcal{T} = (T, \{V_t : t \in V(T)\})$ tree-descompunere cu rădăcină a lui G. Pentru orice $t \in T$ şi orice p-colorare c a lui G_t, definim

$$Prev_t(c) = \begin{cases} 1, & \text{dacă } G[t] \text{ are o } p\text{-colorare } \overline{c}, c\text{-compatibilă} \\ 0, & \text{altfel.} \end{cases}$$

Propoziţie

$Prev_u(c) = 1$ dacă şi numai dacă pentru orice copil v al lui u, există o colorare $\overline{c}$ a lui G_v, c-compatibilă cu $Prev_v(\overline{c}) = 1$

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstraţie. "$\Longrightarrow$" dacă γ este colorare a lui $G[u]$, c-compatibilă, cum G_v este un a subgraf al lui $G[u]$, atunci restricţia lui γ la G_v ne oferă colorarea dorită, $\overline{c}$.

"$\Longleftarrow$" Presupunem că u are exact doi copii v şi w, şi că avem două colorări $\overline{c}'$ şi $\overline{c}''$, c-compatibile, respectiv (demonstraţia este similară pentru mai mulţi copii).

Deoarece $(T, \{V_t : t \in V(T)\})$ este o tree-descompunere, $V(G[v]) \cap V(G[w]) \subseteq V_u$, urmează că $\overline{c}'$ este $\overline{c}''$-compatibilă.

Combinând $\overline{c}'$ şi $\overline{c}''$ obţinem $\overline{c} : V(G[u]) \to \{1, 2, \ldots, p\}$. Cum $(T, \{V_t : t \in V(T)\})$ este o tree-descompunere, nu există muchii $xy \in E(G)$ cu $x \in V(G[v]) - V_u$ şi $y \in V(G[w]) - V_u$, deci $\overline{c}$ este o p-colorare a lui $G[u]$.

$\square$

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms

Teoremă

Dacă G, un graf de ordin n, are o tree-descompunere small $(T, \{V_t : t \in V(T)\})$ cu lăţimea w, atunci putem decide dacă G este p-colorabil în timpul $\mathcal{O}(p^{w+1} \cdot n^{\mathcal{O}(1)})$.

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Demonstraţie. Transformăm $(T, \{V_t : t \in V(T)\})$ într-o a tree-descompunere cu rădăcină (r). Pentru orice $v \in V(T)$ şi orice p-colorare c a lui G_v, determinăm $Prev_v(c)$: plecând din frunzele lui T, şi apoi folosind proprietatea de mai sus pentru celelalte noduri, într-o ordine corespunzătoare.

$G = G[r]$ este p-colorabil dacă şi numai dacă $Prev_v(c) = 1$ pentru o colorare c. Testarea dacăc este G_v colorare şi determinarea lui $Prev_v(c)$ pot fi făcute în timpul $\mathcal{O}(n^{\mathcal{O}(1)})$, astfel complexitatea timp totală este dată în principal de numărul de candidaţi pentru c, adică $p^{|V_v|}$.

Complexitate: $|V(T)| \cdot p^{w+1} \cdot n^{\mathcal{O}(1)} = \mathcal{O}(p^{w+1} \cdot n^{\mathcal{O}(1)})$.

Alte aplicaţii - Abordări similare (programare dinamică mai avansată)

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

Teoremă

Dacă G, un graf de ordin n, are o tree-descompunere small $(T, \{V_t : t \in V(T)\})$ cu lăţimea w, cardinalul minim al unei acoperiri cu noduri a lui G poate fi determinată în timpul $\mathcal{O}(2^{w+1} \cdot n^{\mathcal{O}(1)})$.

Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * 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

Dacă G, un grafde ordin n cu ponderi pe noduri, are o tree-descompunere small $(T, \{V_t : t \in V(T)\})$ cu lăţimea w, o mulţime stabila de pondere maximă a lui G poate fi determinată în timpul $\mathcal{O}(4^{w+1} \cdot w \cdot n)$.

- Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms *

Sfârşit!