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

Algoritmica grafurilor - Cursul 6

Graph Algorithms * C. Croitoru - Graph Algorithms *

Noiembrie 2018

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

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

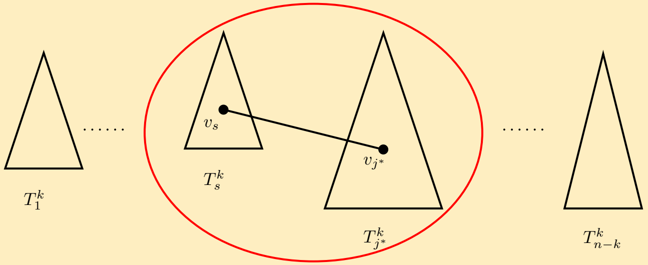
C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms
* C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

- 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 (roşie) 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

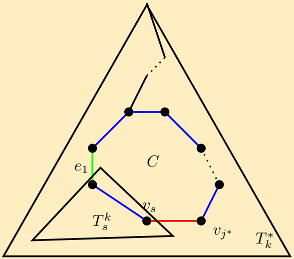
Demonstraţie (continuare). Dacă proprietatea roşie are loc pentru $0 \leqslant k \leqslant n-2$, atunci există un MST al lui G, T_k^*, astfel încât $E(\mathcal{T}^k) \subseteq E(T_k^*)$. Din construcţie, $E(\mathcal{T}^{k+1}) = E(\mathcal{T}^k) \cup \{e^*\}$. Dacă $e^* \in E(T_k^*)$, atunci luăm $T_{k+1}^* = T_k^*$ 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

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.

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

$$\text{(S)} \qquad \forall j \in V \setminus V_s, \beta[j] = c(\alpha[j]j) = \min_{i \in V_s, ij \in E} c(ij)$$

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 -

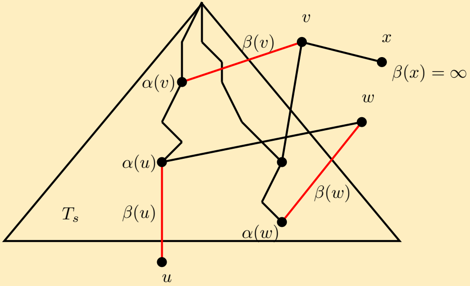
Algoritmul lui Prim

$V_s \leftarrow \{s\}$; $E_s \leftarrow \varnothing$; // 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. $\beta[j^*] = \min\limits_{j \in V \setminus V_s} \beta[j]$;
  $V_s \leftarrow V_s \cup \{j^*\}$; $E_s \leftarrow E_s \cup \{\alpha[j^*]j^*\}$;
  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

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 descrescător 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**
            $T \leftarrow T \cup \{e_i\}$;
        $i++$;

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

- 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**.

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

- 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**
    $union(v, w)$;
    $T \leftarrow T \cup \{e_i\}$;
  $i++$;

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 (correspunzând 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 * 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 (correspunzând familiei $\mathcal{T}^0$):

  for $(v \in V)$ **do**
      $pred[v] \leftarrow 0$;
- 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**
      $i \leftarrow pred[i]$;
  return 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 -

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

  procedure $union(root_1, root_2 : V)$
  $pred[root_1] \leftarrow root_2;$

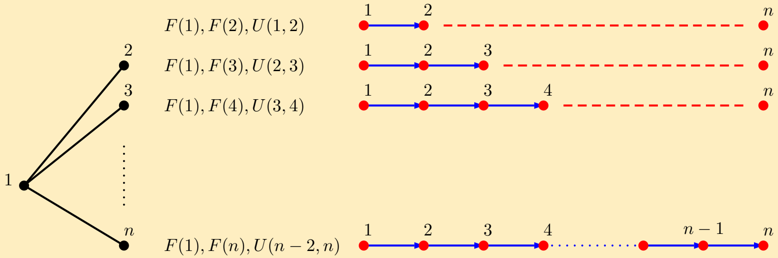
- Bucla while a algoritmului este modificată astfel:

  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 *

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

Dacă executăm bucla while în această formă pentru graful $G = K_{1,n-1}$ cu lista sortată 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 * 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 * C. Croitoru - Graph Algorithms * C. Croitoru - 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 acestei 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. □

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

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

$$h(v) \leqslant \log\left(-pred[find(v)]\right) \leqslant \log n, \forall v \in V.$$

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;

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, şi care are-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.

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:

P₁ Dat un graf $G = (V, E)$, să se determine $M^* \in \mathcal{M}_G$ astfel încât

$$|M^*| = \max_{M \in \mathcal{M}_G} |M|.$$

Notăm cu $\nu(G) = \max_{M \in \mathcal{M}_G} |M|$.

Problema cuplajului maxim este strâns legată de **problema acoperirii minime cu muchii**.

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

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

$$d_F(v) \geqslant 1, \forall v \in V(G).$$

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

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

(2) $$|F^*| \leqslant |F| = n - |M^*|.$$

"$\geqslant$" 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 * C. Croitoru - Graph Algorithms * C. Croitoru - Graph Algorithms * C. Croitoru - Graph

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

Să observăm ca tocmai am demonstrat că două probleme **P$_1$** şi **P$_2$** sunt polinomial echivalente deoarece cuplajul M şi acoperirea cu muchii F construite sunt soluţii optime pentru cele două probleme, respectiv.

Algorithms * C. Croitoru - Graph Algorithms *

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?

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

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 $n - 1$.

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_{vw}^{T} singurul vw-drum din T. Arătaţi că un arbore parţial $T^* = (V, E^*)$ este de cost minim dacă şi numai dacă

$$\forall\, e = vw \in E \setminus E^*, \forall\, e' \in E(P_{vw}^{T^*}),\ \text{avem } c(e) \geqslant c(e').$$

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 $G - e$.

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

$$s[T(H, A, w)] = \sum_{e \in E_T} w(e) =$$

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

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

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

Exerciţiul 5 (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 6. 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.

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

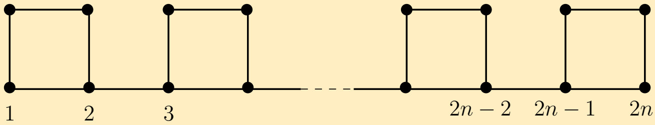
Exerciţiul 8. Fie $G = (V, E)$ un graf conex şi $c : E \to \mathbb{R}$ o funcţie de cost injectivă. Fie T^* un arbore parţial de cost minim al lui G şi T_0 un arbore parţial cu cel de-al doilea cel mai mic cost în G.

(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 9. Determinaţi numărul de cuplaje maxime ale următorului 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 -

Exerciţiul 10. 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.

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

Exerciţiul 11. 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 *