ALGORITMICA GRAFURILOR
Săptămâna 6

C. Croitoru

croitoru@info.uaic.ro

FII

November 3, 2013

OUTLINE

1. **Arbori** (ag 13-14 allinone.pdf pag. 150 $\longrightarrow$ 182)
2. **Problemele pentru seminarul 6**

Teoreme de caracterizare

Teoremă. *Fie $G = (V, E)$ un graf.*
Următoarele afirmații sînt echivalente:
(i) *G este arbore (este conex și fără circuite).*
(ii) *G este conex și este minimal cu această proprietate.*
(iii) *G este fără circuite și este maximal cu această proprietate.*

Teoremă. *Următoarele afirmații sînt echivalente pentru un graf $G = (V, E)$ cu n vîrfuri:*
(i) *G este arbore.*
(ii) *G este conex și are $n - 1$ muchii.*
(iii) *G este fară circuite și are $n - 1$ muchii.*
(iv) *$G = K_n$ pentru $n = 1, 2$ și $G \neq K_n$ pentru $n \geq 3$ și adăugarea unei muchii la G produce exact un circuit.*

Generarea arborilor parţiali ai unui graf

generare-arbori-parţiali(*int i*);

// se generează toţi arborii parţiali ai lui G
avînd drept prime $i-1$ muchii,elementele
$T(1),\ldots,T(i-1)$
ale tabloului E (ordonate crescător).

variabile locale:

$j \in \{1,\ldots m\}$; S, listă de vîrfuri; $x \in V$;

if $i=n$ **then**

// $\{T(1),\ldots,T(n-1)\}$ formează un
arbore parţial ;

prelucrează T *(listează, memorează etc.)*

Generarea arborilor parţiali ai unui graf (continuare)

else
  if $i = 1$ **then**
    for $j := 1$ **to** $d_G(v_0)$ **do**
    $\{\ T[i] \leftarrow j;$
      A:
      generare-arbori-parţiali($i + 1$ *);*
      B:
    $\}$
  else
    for $j := T[i-1] + 1$ **to** $m - (n-1) + i$ **do**
      if $\langle \{T[1], \ldots, T[i-1]\} \cup \{j\} \rangle_G$ nu are circuite
        then
        $\{\ T[i] \leftarrow j;$
          A:
          generare-arbori-parţiali($i + 1$ *);*
          B: $\}$

Numărarea arborilor parţiali ai unui graf

Fie $G = (V, E)$ un multigraf cu $V = \{1, 2, \ldots, n\}$.
Cosiderăm $A = (a_{ij})_{n\times n}$ matricea de adiacenţă a lui G ($a_{ij} =$ multiplicitatea muchiei ij dacă $ij \in E$, altfel 0).
Fie $D = diag(d_G(1), d_G(2), ..., d_G(n))$.

Matricea $L[G] = D - A$ se numeşte **matricea de admitanţă a multigrafului** G sau **matricea Laplace a lui** G.

Teoremă (Kirchoff-Trent; Matrix Tree Theorem)
Dacă G este un multigraf cu mulţimea de vîrfuri $\{1, \ldots, n\}$ şi $L[G]$ matricea Laplace, atunci

$$|\mathcal{T}_G| = det(L[G]_{ii}) \quad \forall i \in \{1, \ldots, n\}.$$

$L[G]_{ij}$ notează minorul lui $L[G]$ obţinut prin îndepărtarea liniei i şi coloanei j.

Corolar $\left|\mathcal{T}_{K_n}\right| = n^{n-2}$ **(Cayley).**

Arbori parțiali de cost minim.

(P1) *Date* $G = (V, E)$ *graf și* $c : E \to \mathbf{R}$ *(*$c(e)$ *– costul muchiei* e*), să se determine* $T^* \in \mathcal{T}_G$ *astfel încît*

$$c(T^*) = \min\{c(T) \mid T \in \mathcal{T}_G\},$$

unde $c(T) = \sum_{e \in E(T)} c(e)$.

Algoritm generic

Inițial, $\mathcal{T}^0 = (T_1^0, T_2^0, \ldots, T_n^0)$, $T_i^0 = (\{i\}, \emptyset)$, $i = \overline{1, n}$ ($V = \{1, 2, \ldots, n\}$).

În pasul k ($k = \overline{0, n-2}$), din familia $\mathcal{T}^k = (T_1^k, T_2^k, \ldots, T_{n-k}^k)$ de $n - k$ arbori a.î $V(T_i^k)_{i=\overline{1,n-k}}$ este partiție a lui V, se construiește $\mathcal{T}^{k+1}$:

- se alege T_s^k unul din arborii familiei $\mathcal{T}^k$.
- dintre muchiile lui G cu o extremitate în T_s^k și cealaltă în $V - V(T_s^k)$, se alege una de cost minim, $e^* = v_s v_{j^*}$ unde $v_{j^*} \in V(T_{j^*}^k)$.
- $\mathcal{T}^{k+1} = (\mathcal{T}^k \setminus \{T_s^k, T_{j^*}^k\}) \cup T$, unde T este arborele obținut din T_s^k și $T_{j^*}^k$ la care adăugăm muchia e^*.

Teoremă. Dacă $G = (V, E)$ este un graf conex cu $V = \{1, 2, \ldots, n\}$ atunci T_1^{n-1} construit de algoritmul descris mai sus este arbore parțial de cost minim.

Algoritmul lui Prim (implementare Dijkstra)

Arborele T_s^k va fi întotdeauna arborele cu cele mai multe vîrfuri dintre arborii familiei curente.
La fiecare pas $k > 0$ avem un arbore $T_s = (V_s, E_s)$ cu $k+1$ vîrfuri, ceilalți $n-k-1$ avînd cîte un singur vîrf.
Fie $\alpha[1..n]$ cu componente din V și $\beta[1..n]$ cu componente reale a.î. :
$\forall j \in V - V_s, \beta[j] = c(\alpha[j]j) = \min\{c(ij) \mid i \in V_s, ij \in E\}$

1. $V_s :\leftarrow \{s\}$; ($s \in V$, oarecare)
$E_s \leftarrow \emptyset$;
for $v \in V \setminus \{s\}$ **do** $\{\ \alpha[v] := s; \beta[v] := c(sv)\}$;
2. **while** $V_s \neq V$ **do**
$\{$ determină $j^* \in V \setminus V_s$ a.î.
$\beta[j^*] = \min\{\beta[j] \mid j \in V - V_s\}$;
$V_s \leftarrow V_s \cup \{j^*\}$;
$E_s := E_s \cup \{\alpha[j^*]j^*\}$;
for $j \in V - V_s$ **do**
if $\beta[j] > c[j^*j]$ **then**
$\{\ \ \beta[j] \leftarrow c[j^*j];\ \ \alpha[j] :\leftarrow j^*\ \}$
$\}$

Algoritmul lui Kruskal

În metoda generală se va alege la fiecare pas drept arbore T_s^k **unul din cei doi arbori cu proprietatea că sînt "uniți" printr-o muchie de cost minim printre toate muchiile cu extremitățile pe arbori diferiți.**
Dacă notăm cu $T = E(\mathcal{T}^k)$, atunci algoritmul poate fi descris astfel:

1. Sortează $E = (e_1, e_2, \ldots, e_m)$ astfel încît:
   $c(e_1) \leq c(e_2) \leq \ldots \leq c(e_m)$.
   1.2 $T \leftarrow \emptyset$; $i \leftarrow 1$;
2. **while** $i \leq m$ **do**
   { **if** $\langle T \cup \{e_i\}\rangle_G$ nu are circuite **then**
   $T \leftarrow T \cup \{e_i\}$;
   $i++$ }

Pasul 1 necesită $O(m \log n)$ operații.
Pentru realizarea eficientă a testului din pasul 2 va fi necesar să reprezentăm la fiecare pas k, $V(T_1^k), V(T_2^k), \ldots, V(T_n^k)$ și să testăm dacă muchia e_i curentă are ambele extremități în aceeași mulțime.
Se vor folosi pentru reprezentarea acestor mulțimi, arbori (care nu sînt în general subarbori ai lui G).

Union-Find - Tarjan

pred$[1..n]$:

pred$[v]$= **vîrful dinaintea lui** v **de pe drumul la** v **de la rădăcina arborelui care memorează mulțimea la care aparține** v**;**
pred$[v] = 0 \Leftrightarrow v$ **este rădăcina arborelui;**
pred$[v] < 0 \Leftrightarrow v$ **este rădăcină a unui arbore și** $-pred[v]$ **este cardinalul mulțimii memorate în el".**

Adăugăm în pasul 1 inițializarea 1.3 și modificăm pasul 2 al algoritmului astfel:

1.3 **for** $v \in V$ **do** $pred[v] \leftarrow -1$;
2. **while** $i \leq m$ **do**
{ fie $e_i = vw$;
$x \leftarrow find(v)$; $y \leftarrow find(w)$;
if $x \neq y$ **then** $\{ union(x, y); T \leftarrow T \cup \{e_i\} \}$
$i{+}{+}$ }

Union-Find - Tarjan

Procedura union și funcția find sunt:

procedure $union(v, w : V)$**;**
  $//v$ și w sunt rădăcini, variabila locală întreagă t
  $t \leftarrow pred[v] + pred[w]$;
  if $pred[v] > pred[w]$ **then** $\{\ pred[v] \leftarrow w; pred[w] \leftarrow t\ \}$
  else $\{\ pred[w] \leftarrow v; pred[v] \leftarrow t\ \}$

function $find(v : V)$**;** $//$variabile întregi locale i, j, k;
  $i \leftarrow v$;
  while $pred[i] > 0$ **do** $i \leftarrow pred[i]$**;**
  $j \leftarrow v$;
  while $pred[j] > 0$ **do**
  $\{\ k \leftarrow pred[j];\ pred[j] \leftarrow i;\ j \leftarrow k;\ \}$
  return i

Complexitatea pasului 2 este $O(m \cdot \alpha(m, n))$, unde $\alpha(m, n)$ este inversa funcției lui Ackermann.

Problemele pentru seminarul 6

Se vor discuta (cel puțin) patru probleme dintre următoarele:

1. **Problema 1, Setul 13**
2. **Problemele 1,2,4 Setul 5**
3. **Problema 1, Setul 7**
4. **Problemele 1,2 Setul 17**
5. **Problema 4, Setul 20**