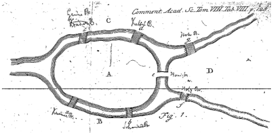
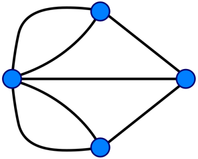




Algoritmica grafurilor

XI-XII. Cuplaje în grafuri. Alg. pct. articulație, punți, comp biconectate

Mihai Suciu

Facultatea de Matematică și Informatică (UBB)
Departamentul de Informatică

Mai, 16, 2019

Continut

1. Cuplaje in grafuri
   - Grafuri bipartite ponderate
   - Metoda maghiara
2. Puncte de articulatie, punti si componente biconectate
   - Puncte de articulație
   - Punti
   - Componente biconectate

Cuplaje în grafuri - recapitulare C10

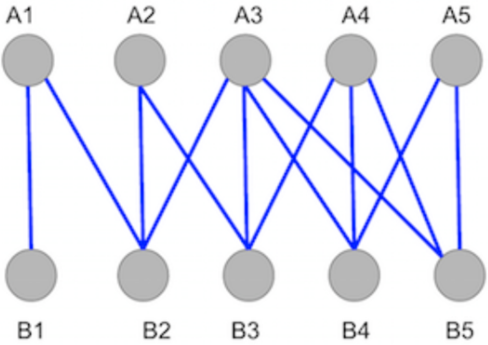
- Un cuplaj în G este o mulțime de muchii M în care nici o pereche de muchii nu are un vârf comun. vârfurile adiacente la muchiile din M se numesc vârfuri *saturate de M* (sau *M-saturate*). Celelalte vârfuri se numesc *M-nesaturate*.

Tipuri de cuplaje:

- un cuplaj perfect al lui G este un cuplaj care saturează toate vârfurile lui G,
- un cuplaj maxim al lui G este un cuplaj care are cel mai mare număr posibil de muchii,
- un cuplaj maximal al lui G este un cuplaj care nu poate fi lărgit prin adăugarea unei muchii.

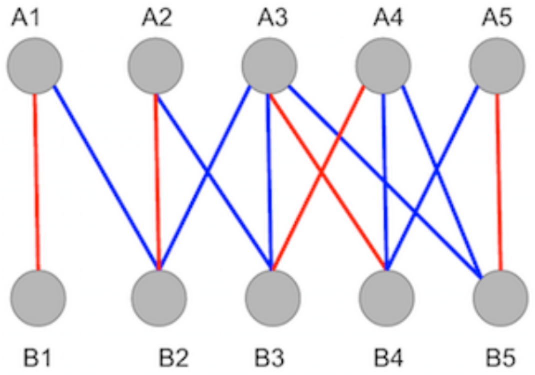
Cuplaje în grafuri

Un graf bipartit:



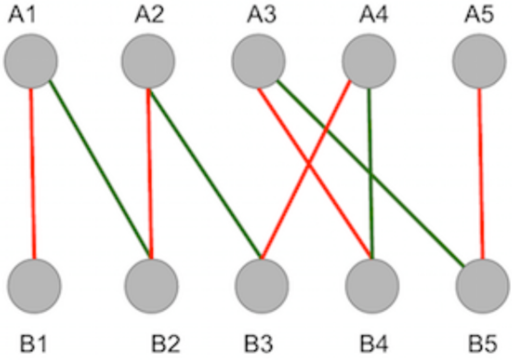
Cuplaje în grafuri

Un cuplaj aleator în G



Cuplaje în grafuri - lanț M-alternant

Un *lanț M-alternant* (*cale M-alternantă*) este un lanț în G în care toate muchiile alternează între muchii din M și muchii ce nu aparțin cuplajului M.



Cuplaje în grafuri - M-lanț de creștere

Un *M-lanț de creștere* (*M-cale de creștere*) este un lanț M-alternant care are ambele capete M-nesaturate.

Teorema lui Berge

Un cuplaj M al unui graf $G = (V, E)$ este maxim **dacă și numai dacă** G nu conține M-lanțuri de creștere.

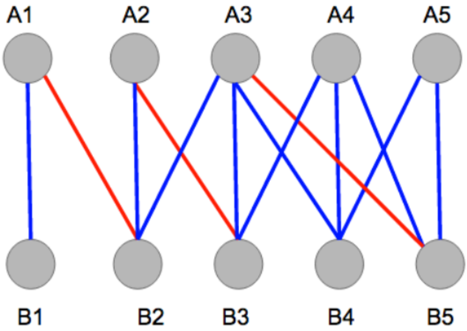
Demonstrație.

... □

Cuplaje în grafuri

Este cuplajul M de mai jos maxim?

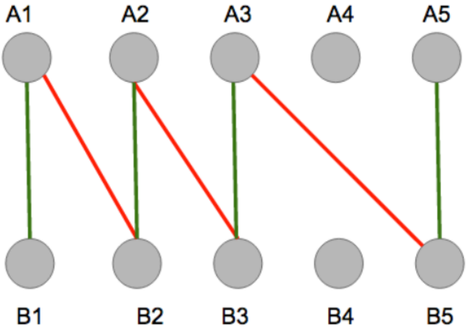
- $M = \{(A1, B2), (A2, B3), (A3, B5)\}$



Cuplaje în grafuri

Nu, deoarece conține un M-lanț de creștere.

- M-lanț de creștere $(B1, A1, B2, A2, B3, A3, B5, A5)$



Cuplaj maxim în grafuri bipartite

Teorema lui Hall

fie G un graf bipartit cu mulțimile partite X și Y. X poate fi cuplat în Y dacă și numai dacă $|N(S)| \geq |S|$ pentru toate submulțimile S ale lui X.

- intuitiv: fiecare nod trebuie să aibă suficienți vecini
- dacă G este bipartit cu $V = X \cup Y$ spunem că X poate fi cuplat în Y dacă există un cuplaj al lui G care saturează nodurile din X

Cum se găsește un cuplaj maxim într-un graf bipartit?

- algoritmi de flux maxim

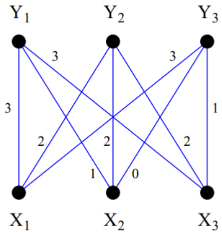
Grafuri bipartite ponderate

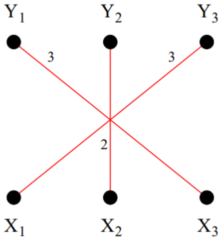
Fie un graf bipartit ponderat unde:

- muchia $(x, y) \in E$ are asociată ponderea $w(x, y)$
- ponderea cuplajului M este suma ponderilor muchiilor din cuplajul M

$$w(M) = \sum_{(x,y) \in M} w(x, y)$$

Problema: pentru graful bipartit G găsiți un cuplaj de pondere maximă.





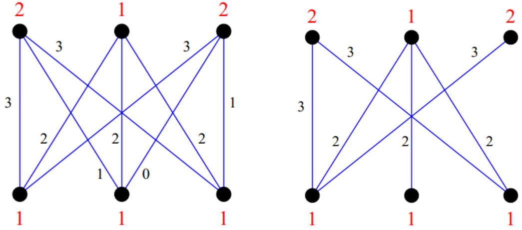
Etichetarea grafului

- o etichetare a vârfurilor este o funcție $l : V \to \mathbb{R}$,
- o etichetare fezabilă respectă:

$$l(x) + l(y) \geq w(x, y), \forall x \in X, y \in Y,$$

- un graf egal (ținând cont de l) este un graf $G = (V, E_l)$ unde:

$$E_l = \{(x, y) | l(x) + l(y) = w(x, y)\}.$$



(a) etichetare fezabila l (b) Graf egal G_l

Etichetarea grafurilor (II)

Teorema Kuhn-Munkres

Dacă l este fezabilă și M este un cuplaj perfect în E_l atunci M este un cuplaj de pondere maximă.

- Teorema KM transformă problema găsirii unui cuplaj de pondere maximă (problemă de optimizare) într-o problemă combinatorială ce presupune găsirea unui cuplaj perfect.
- pentru un cuplaj M și o etichetare fezabilă l avem:

$$w(m) \leq \sum_{v \in V} l(v)$$

(seamănă cu teorema fluxului maxim și a tăieturii minime)

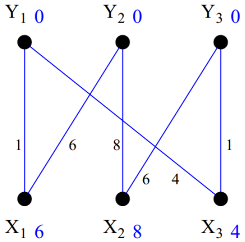
Un posibil algoritm

cuplaj(G**)**

1: start cu o etichetare fezabilă l și un cuplaj M în E_l
2: **while** cuplajul M nu e perfect **do**
3: caută un M-lanț de creștere pentru M în E_l (crește dimensiunea lui M)
4: **if** nu există un M-lanț de creștere **then**
5: îmbunătățește l la l' astfel încât $E_l \subset E_{l'}$

- în fiecare pas se crește dimensiunea lui M sau E_l
- conform teoremei Kuhn-Munkres, M va fi un cuplaj de pondere maximă

Găsirea unei etichetări fezabile inițiale



- pentru a găsi inițial o etichetare fezabilă se poate folosi:

$$\forall y \in Y, l(y) = 0,\ \ \forall x \in X, l(x) = \max_{y \in Y}\{w(x,y)\}$$

- astfel este evident

$$\forall x \in X, y \in Y, w(x,y) \leq l(x) + l(y)$$

Îmbunătățirea etichetării

- fie l o etichetare fezabilă
- se definește un vecin al lui $u \in V$ și un set $S \subseteq V$ astfel:

$$N_l(u) = \{v|(u,v) \in E_l\}, \;\; N_l(S) = \cup_{u \in S} N_l(u)$$

Lema

Fie $S \subseteq X$ și $T = N_l(S) \neq Y$. Fie

$$\alpha_l = \min_{x \in S, y \notin T} \{l(x) + l(y) - w(x,y)\} \tag{1}$$

$$l'(v) = \begin{cases} l(v) - \alpha_l & \text{dacă } v \in S, \\ l(v) + \alpha_l & \text{dacă } v \in T, \\ l(v) & \text{altfel.} \end{cases} \tag{2}$$

Atunci l' este o etichetare fezabilă și

1. dacă $(x,y) \in E_l$ pentru $x \in S, y \in T$ atunci $(x,y) \in E_{l'}$
2. dacă $(x,y) \in E_l$ pentru $x \notin S, y \notin T$ atunci $(x,y) \in E_{l'}$
3. există o muchie $(x,y) \in E_{l'}$ pentru $x \in S, y \notin T$

Metoda maghiară - exemplu matriceal

Vreau să organizez o petrecere, vreau să angajez un muzician, bucătar și serviciu de curățenie. Am la dispoziție 3 companii, fiecare poate furniza un singur serviciu. Ce companie trebuie să furnizeze fiecare serviciu astfel încât costul total să fie minim?

Companie	Cost muzician	Cost bucătar	Cost curățenie
A	108	125	150
B	150	135	175
C	122	148	250

Metoda maghiară - exemplu matriceal (II)

Companie	Cost muzician	Cost bucătar	Cost curățenie
A	108	125	150
B	150	135	175
C	122	148	250

Fie matricea asociată tabelului:

108	125	150
150	135	175
122	148	250

Metoda maghiară - exemplu matriceal (III)

Fie matricea asociată tabelului:

108	125	150
150	135	175
122	148	250

Pas 1. Se scade valoarea minimă de pe fiecare rând din fiecare element de pe rând:

0	17	42
15	0	40
0	26	128

Metoda maghiară - exemplu matriceal (IV)

Pas 1. Se scade valoarea minimă de pe fiecare rând din fiecare element de pe rând:

0	17	42
15	0	40
0	26	128

Pas 2. Se scade valoarea minimă de pe fiecare coloană din fiecare element de pe coloană:

0	17	2
15	0	0
0	26	88

Metoda maghiară - exemplu matriceal (V)

Pas 3. Se desenează linii pe rândurile și coloanele din matrice ce conțin valoarea 0 astfel încât să se traseze cât mai puține linii:

0	17	2
15	0	0
0	26	88

Au fost trasate doar două linii ($2 < n = 3$), algoritmul continuă.

Metoda maghiară - exemplu matriceal (VI)

Se caută cea mai mică valoare care nu este acoperită de nicio linie. Se scade această valoare de pe fiecare rând pe care nu s-au trasat linii și apoi se adaugă la fiecare coloană pe care am trasat linii. Apoi, se revine la Pasul 3.

-2	15	0
15	0	0
-2	24	86

(a) Scade val. minimă

0	15	0
17	0	0
0	24	86

(b) Adaugă pe col.

Metoda maghiară - exemplu matriceal (VII)

Se revine la Pasul 3:

0	15	0
17	0	0
0	24	86

Am trasat 3 linii, $n = 3$, algoritmul a terminat. Se alege o alocare prin alegerea unui set de valori 0 astfel încât fiecere rând sau coloană sa aibă o singură valoare selectată.

0	15	0
17	0	0
0	24	86

Ex. compania C trebuie sa furnizeze muzicianul, compania A trebuie să furnizeze serviciul de curățenie → compania B ne dă bucătarul.

Metoda maghiară - exemplu matriceal (VIII)

Soluția finală:

108	125	150
150	135	175
122	148	250

Putem verifica soluția și exhaustiv:

- $108 + 135 + 250 = 493$
- $108 + 148 + 175 = 431$
- $150 + 125 + 250 = 525$
- $150 + 148 + 150 = 448$
- $122 + 125 + 175 = 422$
- $122 + 135 + 150 = 407$

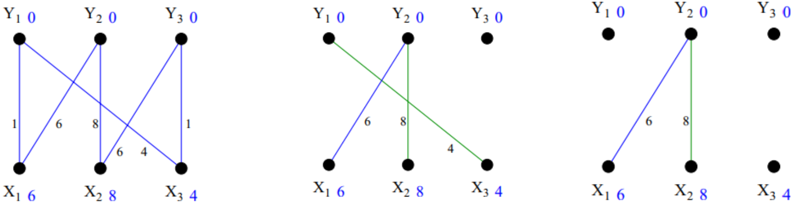
Metoda maghiară

metoda_maghiară(G)

```
1:  generează o etichetare inițială l și un cuplaj M în E_l
2:  if M nu este un cuplaj perfect then
3:      alege un vârf liber x ∈ X
4:      S = {u}, T = ∅
5:      if N_l(S) = T then
6:          actualizează etichetele conform (1) și (2) (forțând N_L(S) ≠ T)
7:      if N_l(S) ≠ T then
8:          alege y ∈ N_l(S) − T
9:          if y e liber then
10:             u − y este un lanț-M de creștere.
11:             îmbunătățește M și sari la linia 2
12:         else
13:             T = T ∪ {y} și sari la linia 5
```

Complexitatea algoritmului $O(V^4)$, ulterior redusă la $O(V^3)$.

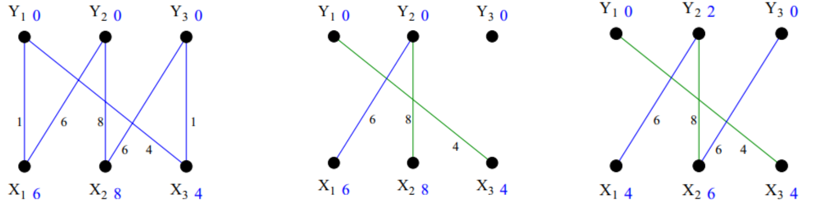
Metoda maghiară - exemplu graf



(a) graful inițial (b) E_l și cuplaj (c) lanț alternant

- graful inițial, etichetarea vârfurilor și graful egal asociat
- cuplajul inițial $M = \{(x_3, y_1), (x_2, y_2)\}$, $S = \{x_1\}$, $T = \varnothing$
- deoarece $N_l(S) \neq T$ mergi la linia 7: alege $y_2 \in N_l(S) - T$
- y_2 este în cuplaj, crește lanțul alternant prin adăugarea lui (y_2, x_2), $S = \{x_1, x_2\}$, $T = \{y_2\}$
- $N_l(S) = T$, sari la linia 5

Metoda maghiară - exemplu (II)



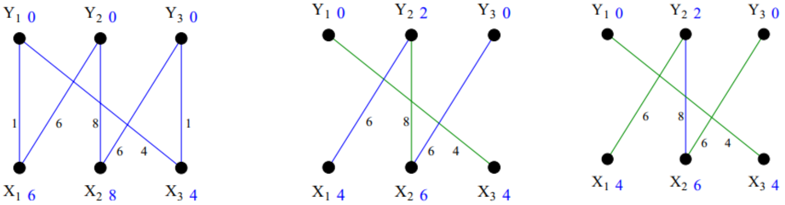
(a) graful inițial (b) vechiul graf E_l și M (c) noul graf E_l și M

- $S = \{x_1, x_2\}$, $T = \{y_2\}$ și $N_l(S) = T$
- se determină α_l

$$\alpha_l = \min_{x \in S, y \notin T} \begin{cases} 6+0-1 & (x_1, y_1) \\ 6+0-0 & (x_1, y_3) \\ 8+0-0 & (x_2, y_1) \\ 8+0-6 & (x_2, y_3) \end{cases} = 2$$

Metoda maghiară - exemplu (III)

- redu etichetele lui S cu 2, mărește etichetele lui T cu 2
- acum $N_l(S) = \{y_2, y_3\} \neq \{y_2\} = T$, $S = \{x_1, x_2\}$



(a) graful inițial (b) noul graf E_l și M (c) cuplaj nou

- alege $y_3 \in N_l(S) - T$ și adaugă în T
- y_3 nu e în cuplaj, am găsit un lanț de creștere (x_1, y_2, x_2, y_3)
- cuplajul $\{(x_1, y_2), (x_2, y_3), (x_3, y_1)\}$ are costul $6 + 6 + 4 = 16$ care este egal cu suma etichetelor grafului final

Puncte de articulație și punți

Definiție

fie $G = (V, E)$ un graf neorientat și $u \in V$ un vârf oarecare din graf. Vârful u este ***punct de articulație*** *al grafului G dacă există cel puțin două vârfuri $x, y \in V, x \neq y, x \neq u$, și $y \neq u$, astfel încât orice lanț $x \rightsquigarrow y$ trece prin u.*

Definiție

fie $G = (V, E)$ un graf neorientat și $(u, v) \in E$ o muchie oarecare din graf. Muchia (u, v) este ***punte*** *în graful G dacă există cel puțin două vârfuri $x, y \in V, x \neq y$, astfel încât orice lanț $x \rightsquigarrow y$ în G conține muchia (u, v).*

Puncte de articulație

Teorema

Fie $G = (V, E)$ un graf neorientat si $u \in V$ un vârf oarecare din G. Vârful u este ***punct de articulație*** *în G dacă și numai dacă în urma $DFS(G)$ una din proprietățile de mai jos este satisfăcută:*

a. *$u.\pi = null$ și u domină cel puțin doi subarbori*
b. *$u.\pi \neq null$ și există un vârf v descendent al lui u în $ARB(u)$ astfel încât pentru orice vârf x din $ARB(v)$ și orice arc (x, z) parcurs de DFS avem $z.d \geq u.d$.*

- $ARB(u)$ este arborele ce are rădăcina u
- $u.d$ timpul în care a fost descoperit vârful u

Puncte de articulație (II)

Algoritmul pentru determinarea unui punct de articulație urmărește teorema 7.1. În afară de proprietățile necesare DFS, unui vârf $u \in V$ i se atașează proprietățile:

1. $u.b = \min\{v.d | v$ descoperit pornind din u în cursul DFS si $v.color \neq$ alb$\}$;
2. $subarb(u)$ este numărul subarborilor dominați de u.

Puncte de articulație (III)

Există mai multe momente importante în cursul *DFS* în care $u.b$ este modificat sau vârful u este testat pentru a fi marcat ca vârf de articulație:

- în momentul descoperirii lui u, $u.b = u.d$;
- în momentul în care din u se ajunge la un succesor v al lui u și $v.color = alb$, $u.b = \min\{u.b, v.d\}$;
- în momentul în care dintr-un succesor v al lui u se revine în u, $u.b = \min\{u.b, v.b\}$, dacă $u.b \geq u.d$ și $u.\pi \neq null$ atunci u este un vârf de articulație - cazul (b) din teorema 7.1;
- în momentul în care din u se revine în ciclul principal al *DFS*: dacă u domină doi subarbori, atunci u este punct de articulație - cazul (a) din teorema 7.1.

Puncte de articulație (IV)

- Se marchează cu $u.articlulatie = 1$ vârfurile de articulație din graf.
- Pentru a verifica ușor numărul de subarbori *DFS* al unui vârf se introduce proprietatea $u.subarb$, $\forall v \in V$, inițial $u.subarb = 0$. Acest atribut crește pentru fiecare succesor alb al lui u.

Puncte de articulație - algoritm

ARTICULATII(G)

```
timp = 0
for u ∈ V do
    u.color = alb
    u.π = NIL
    u.subarb = 0
    u.articulatie = 0
for u ∈ V do
    if u.color = alb then
        EXPLORARE(u)
        if u.subarb > 1 then
            u.articulatie = 1
```

Puncte de articulație - algoritm (II)

EXPLORARE(u)

$u.d = u.b = timp++$
$u.color = gri$
for $v \in succs(u)$ **do**
  if $u.c = alb$ **then**
    $v.\pi = u$
    $u.subarb++$
    $EXPLORARE(v)$
    $u.b = \min\{u.b, v.b\}$
    if $u.\pi \neq NIL \wedge v.b \geq u.d$ **then**
      $u.articulatie = 1$
  else
    $u.b = \min\{u.b, v.d\}$

Punti

Teorema

fie $G = (V, E)$ *un graf neorientat și* $(u, v) \in E$ *o muchie oarecare din graf. Muchia* (u, v) *este punte în* G *dacă și numai dacă în urma* $DFS(G)$ *una din proprietățile de mai jos este satisfăcută:*

1. v *este descendentul direct al lui* u *în* $ARB(u)$ *și nu există nici un descendent* $DFS(G)$ *al lui* v *care să formeze arce inverse cu vreun vârf* $z, z.d \leq u.d$.
2. u *este descendent direct al lui* v *în* $ARB(v)$ *și nu există nici un descendent* $DFS(G)$ *al lui* u *care să formeze arce inverse cu vreun vârf* $z, z.d \leq u.d$.

Punti (II)

Algoritmul pentru detectarea muchiilor punți străbate în adâncime graful și verifică următoarele proprietăți simetrice impuse unei muchii (u, v) pentru a fi punte:

1. $v.\pi = u$ și parcurgerea în adâncime a grafului pornind din v - străbătând muchii diferite de (u, v) - nu descoperă vârful u sau vârfuri explorate înaintea lui $u, u.d < v.b$. În acest caz, pentru a ajunge de la u la v sau la orice alt vârf descoperit din v, nu există alte lanțuri decât cele care trec prin (u, v).
2. $u.\pi = v$ și parcurgerea în adâncime a grafului pornind din u - străbătând muchii diferite de (u, v) - nu descoperă vârful v sau vârfuri explorate înaintea lui $v, v.d < u.b$.

Punți (III)

- În cursul $DFS(G)$ parcurgerea muchiilor grafului G trebuie efectuată într-un sens;
- dacă v este un vârf adiacent lui u și culoarea lui v este albă și muchia (u, v) este străbătută de algoritm în sensul $u \rightarrow v$ atunci parcurgerea în sens invers trebuie blocată;
- altfel dacă arcul este străbătut ulterior în sensul $v \rightarrow u$ vârful u este descoperit ca vârf de culoare gri (muchia (v, u) este arc invers) iar valoarea $v.b$ este actualizată la $\min\{u.b, v.b\} \rightarrow$ va satisface $v.b = u.d$ chiar dacă pentru orice alt vârf x la care se ajunge din v avem $x.b > u.d$;
- în acest caz muchia (u, v) nu este recunoscută ca punte deși este punte.

Punți (IV)

- Pentru a bloca parcurgerea în sens invers a muchiei (u, v) algoritmul se folosește de $v.\pi$;
- la prima descoperire a vârfului v din u pe muchia (u, v) se stabilește $v.\pi = u$;
- la avansul ulterior din v se evită orice muchie (v, x) pentru care $v.\pi = x$;
- complexitatea algoritmului este aceeași ca și pentru *DFS* sau $ARTICULATII : \Theta(V + E)$

Punti - algoritm

- Fiecărui vârf din graf i se atașează atributul *punte* astfel $u.punte = 1$ înseamnă că muchia $(u, u.\pi)$ este punte în G.

PUNTI(G)

for $u \in V$ **do**
    $u.color = alb$
    $u.\pi = NIL$
    $u.punte = 0$
$timp = 0$
for $u \in V$ **do**
    if $u.color = alb$ **then**
        EXPLORARE_PUNTI(u)

Punti - algoritm (II)

EXPLORARE_PUNTI(u)

$u.d = u.b = timp++$
$u.color = gri$
for $v \in succs(u)$ **do**
    if $u.color = alb$ **then**
        $v.\pi = alb$
        EXPLORARE_PUNTI(v)
        $u.b = \min\{u.b, v.b\}$
        if $v.b > u.d$ **then**
            $v.punte = 1$
    else
        if $u.\pi \neq v$ **then**
            $u.b = \min\{u.b, v.d\}$

Componente biconectate

Definiție

fie $G = (V, E)$ un graf neorientat. O ***componentă biconectată*** *(sau bicomponentă) a lui G este un subgraf maximal $G_b = (V_b, E_b)$ cu $V_b \subseteq V$ și $E_b \subseteq E$ care nu conține puncte de articulație.*

sau

Definiție

fie $G = (V, E)$ un graf neorientat. O ***componentă biconectată*** *(sau bicomponentă) a lui G este un subgraf maximal $G_b = (V_b, E_b)$ cu $V_b \subseteq V$ și $E_b \subseteq E$ astfel încât pentru orice muchii α și β din E_b există un ciclu simplu care contine muchiile α și β.*

Componente biconectate (II)

Teorema

fie $G = (V, E)$ un graf neorientat și u un vârf nesingular din G ($succs(u) \neq \emptyset$). Vârful u este vârf de start al unei bicomponente a lui G dacă și numai dacă în urma $DFS(G)$ există cel puțin un subarbore $ARB(v)$ dominat de u astfel încât pentru orice muchie (x, z) - cu x în $ARB(v)$ - descoperit în cursul $DFS(G)$ avem $u.d \leq z.d$.

Componente biconectate - algoritm

BICOMPONENTE(G)

```
for u ∈ V do
    u.color = alb
timp = 0
componente = ∅
for u ∈ V do
    if u.color = alb then
        if sucs(u) ≠ ∅ then
            componente = componente ∪ EXPLORARE_BICOMP(u)
        else
            u.color = negru
            componente = componente ∪ {u}
return componente
```

Componente biconectate - algoritm (II)

EXPLORARE_BICOMP(u)

$u.d = u.b = timp++$
$u.color = gri$
$componente_u = \emptyset$
for $v \in succs(u)$ **do**
    if $v.color = alb$ **then**
        $componente_u = componente_u \cup$ EXPLORARE_BICOMP(v)
        $u.b = \min\{u.b, v.b\}$
        if $u.d \leq v.b$ **then**
            $componente_u = componente_u \cup \{COLECTARE(u, v)\}$
    else
        $u.b = \min\{u.b, v.d\}$
return $componente_u$

Componente biconectate - algoritm (III)

COLECTARE(start, vecin)

1: $start.color = negru$
2: $componenta = PARCURGERE \cup \{start\}$
3: $start.color = gri$
4: **return** $componenta$

PARCURGERE(varf)

1: $varf.color = negru$
2: $componenta = \{varf\}$
3: **for** $v \in succs(varf)$ **do**
4:     **if** $v.color = gri$ **then**
5:         $componenta = componenta \cup PARCURGERE(v)$
6: **return** $componenta$