Suport curs algoritmica grafurilor
XII. Algoritmi pentru puncte de articulație, punți și componente biconectate.

Definiţie 12.0.1. *Fie $G = (V, E)$un graf neorientat şi $u \in V$ un vârf oarecare din graf. Vârful u este* ***punct de articulaţie*** *al grafului G dacă există cel puţin două vârfuri $x, y \in V, x \neq y, x \neq u$, şi $y \neq u$, astfel încât orice lanţ $x \rightsquigarrow y$ trece prin u.*

Definiţie 12.0.2. *Fie $G = (V, E)$un graf neorientat şi $(u, v) \in E$ o muchie oarecare din graf. Muchia (u, v) este* ***punte*** *în graful G dacă există cel puţin două vârfuri $x, y \in V, x \neq y$, astfel încât orice lanţ $x \rightsquigarrow y$ în G conţine muchia (u, v).*

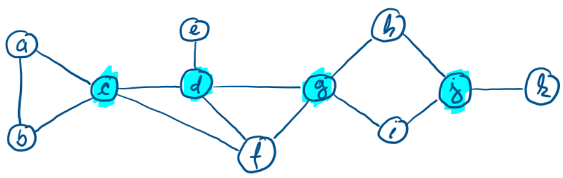


Figura 1: Puncte de articulaţie şi punţi.

Pentru graful din figura 1, vârfurile c, d, g, j sunt puncte e articulație iar muchiile $(d, e), (j, k)$ sunt punți.

12.1 Puncte de articulație

Teorema 12.1. *Fie $G = (V, E)$un graf neorientat şi $u \in V$ un vârf oarecare în graf. Vârful u este un punct de articulaţie în G dacă şi numai dacă în urma $DFS(G)$ una din proprietăţile de mai jos este satisfăcută:*

a. $u.\pi = NIL$ şi u domină cel puţin doi subarbori (sau: u este rădăcină în arborele indus de DFS şi u are cel puţin doi copii).

b. $u.\pi \neq NIL$ şi există un vârf $v \in V$ descendent al lui u n arborele dat de $DFS(G)$ (arbore notat $Arb(u)$) astfel încât pentru orice vârf x din $Arb(v)$ şi orice muchie (x, z) parcursă de DFS avem $z.d \geq u.d$ (sau: u nu este rădăcină în arborele indus de DFS şi are un copil v în arbore astfel încât nici un vârf din subarborele dominat de v nu este conectat cu un strămoş al lui u printr-o muchie înapoi - copii lui nu pot ajunge pe altă cale pe un nivel superior în arborele de adâncime).

Algoritmul pentru determinarea unui punct de articulație urmărește teorema (12.1). 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.

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 12.1;
- 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 (12.1).

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.

Mai jos este prezentat algoritmul pentru găsirea punctelor de articulație din G.

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
```

EXPLORARE(u)

```
u.d = u.b = timp + +
u.color = gri
for v ∈ succs(u) do
    if u.c = alb then
        v.π = u
        u.subarb + +
        EXPLORARE(v)
        u.b = min{u.b, v.b}
        if u.π ≠ NIL ∧ v.b ≥ u.d then
            u.articulatie = 1
    else
        u.b = min{u.b, v.d}
```

Figura 2 prezintă un exemplu. În urma DFS o muchie (u, v) este de tip înapoi (back) dacă se respectă inegalitățile între timpii de descoperire și finalizare $u.d > v.d$, $u.f < v.f$ și culoarea vârfurilor este gri. O muchie din G apartine arborelui descoperit de DFS dacă se respectă inegalitățile între timpii de descoperire și finalizare $u.d < v.d$, $u.f > v.f$ și culoarea vârfului v este albă și a vârfului u este gri.

Figura 2c prezintă lista de adiacență pentru graful din figura 2a. Dacă se parcurge lista de adiacență în ordine alfabetică, timpii de descoperire și finalizare pentru $DFS(G)$ sunt prezentați în figura 2b (în interiorul vârfurilor sunt prezentați d/f). Arborele îndus de $DFS(G)$ este prezentat în figura 2d. Conform teoremei (12.1) vârfurile d și f sunt puncte de articulație (atenție și la atributul culoare al vârfului nu doar la timpii de descoperire).

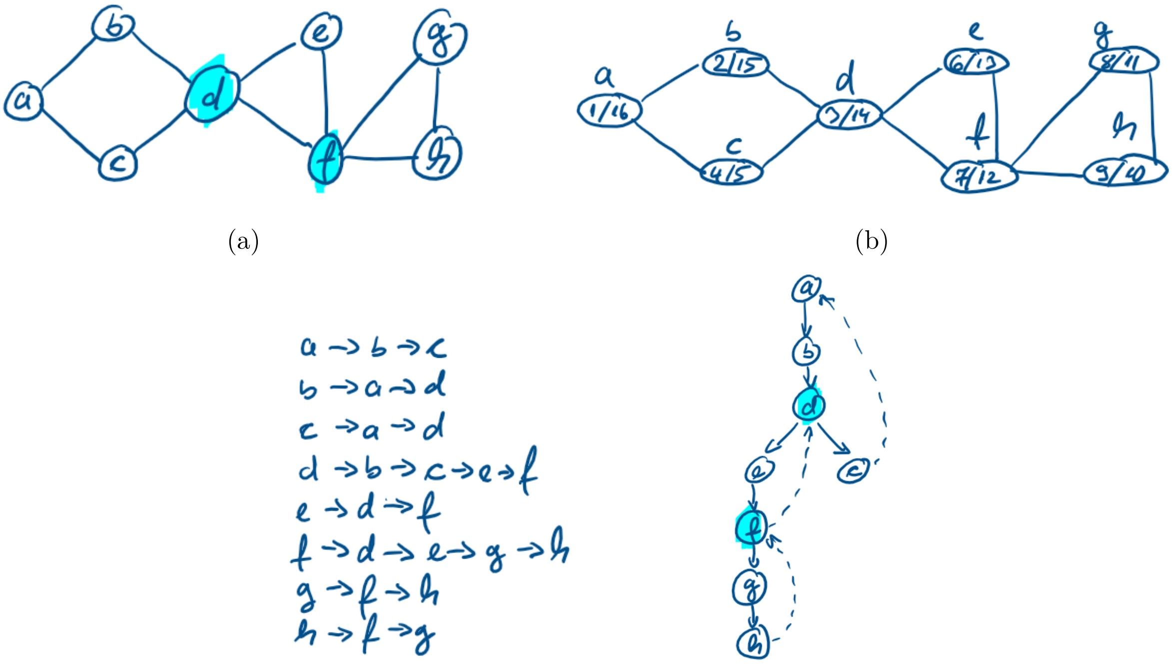


Figura 2: Exemplu de căutare a punctelor de articulaţii într-un graf.

12.2 Punți

Teorema 12.2. *Fie $G = (V, E)$un graf neorientat şi $(u, v) \in E$ o muchie oarecare din graf. Muchia (u, v) este punte în G dacă şi numai dacă în urma $DFS(G)$ una din proprietăţile de mai jos este satisfăcută:*

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

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 $v, v.d < u.b$.

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 $u \to v$ atunci parcurgerea în sens invers trebuie blocată. Altfel dacă arcul este străbătut ulterior în sensul $v \to u$ vârful u este descoperit ca vârf de culoare gri (muchia (v, u) este arc invers) iar valoarea $v.b$ este actualizată la $\min\{u.b, v.b\}$ $\to$ 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.

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 $ARTICULATII$: $\Theta(V + E)$.

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

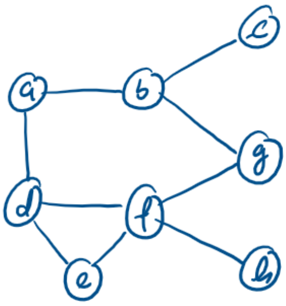
PUNTI(G)

```
for u ∈ V do
    u.color = alb
    u.π = NIL
    u.punte = 0
timp = 0
for u ∈ V do
    if u.color = alb then
        EXPLORARE_PUNTI(u)
```

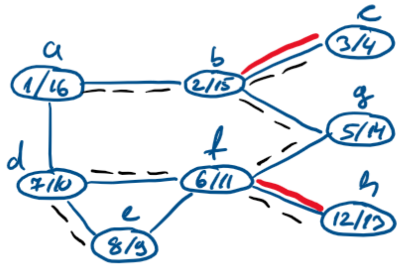
EXPLORARE_PUNTI(u)

```
u.d = u.b = timp++
u.color = gri
for v ∈ succs(u) do
    if u.color = alb then
        v.π = alb
        EXPLORARE_PUNTI(v)
        u.b = min{u.b, v.b}
        if v.b > u.d then
            v.punte = 1
    else
        if u.π ≠ v then
            u.b = min{u.b, v.d}
```

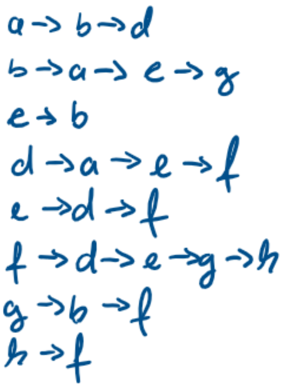
Figura 3 prezintă rezultatul pentru căutarea muchiilor punți într-un graf (graful inițial, timpii descoperiți de DFS, lista de adiacență și arborele găsit).



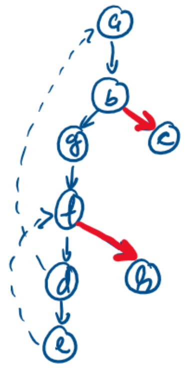
(a)



(b)



(c)



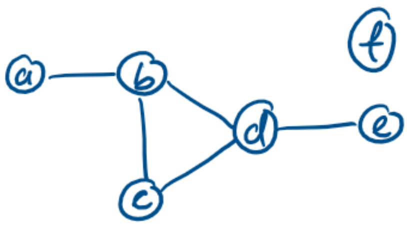
(d)

Figura 3: Exemplu de căutare a muchiilor punţi într-un graf.

12.3 Componente biconectate

Teorema 12.3. *Fie $G = (V, E)$un graf neorientat şi u un vârf nesingular din G ($succs(u) \neq \emptyset$). Vârful u este vârf de start al unei bicomponente a lui G dacă şi numai dacă în urma $DFS(G)$ există cel puţin 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 $u.d \leq z.d$.*

Figura 4 prezintă un graf și componentele biconectate.



(a)

(b)

Figura 4: Componente biconectate.

Mai jos este prezentat algoritmul pentru căutarea componentelor biconectate.

BICOMPONENTE(G)

for $u \in V$ **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
```

EXPLORARE_BICOMP(u)

```
u.d = u.b = timp++
u.color = gri
componente_u = ∅
for v ∈ succs(u) do
    if v.color = alb then
        componente_u = componente_u ∪ EXPLORARE_BICOMP(v)
        u.b = min{u.b, v.b}
        if u.d ≤ v.b then
            componente_u = componente_u ∪ {COLECTARE(u, v)}
    else
        u.b = min{u.b, v.d}
return componente_u
```

COLECTARE(start, vecin)

```
start.color = negru
componenta = PARCURGERE ∪ {start}
start.color = gri
return componenta
```

PARCURGERE(varf)

```
varf.color = negru
componenta = {varf}
for v ∈ succs(varf) do
    if v.color = gri then
        componenta = componenta ∪ PARCURGERE(v)
return componenta
```