

# Suport curs algoritmica grafurilor

## XII. Algoritmi pentru puncte de articulație, punți și componente biconectate.

**Definiție 12.0.1.** 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 12.0.2.** 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  $(u, v)$ .

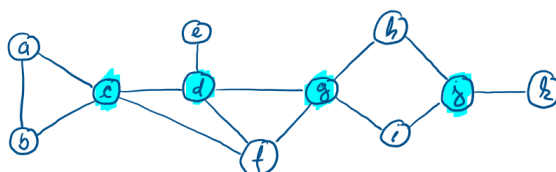


Figura 1: Puncte de articulație și punți.

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

### 12.1 Puncte de articulație

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

- $u.\pi = NIL$  și  $u$  domină cel puțin doi subarbori (**sau**:  $u$  este rădăcină în arborele indus de  $DFS$  și  $u$  are cel puțin doi copii).
- $u.\pi \neq NIL$  și există un vârf  $v \in V$  descendent al lui  $u$  în arborele dat de  $DFS(G)$  (arbore notat  $Arb(u)$ ) astfel încât pentru orice vârf  $x$  din  $Arb(v)$  și orice muchie  $(x, z)$  parcursă de  $DFS$  avem  $z.d \geq u.d$  (**sau**:  $u$  nu este rădăcină în arborele indus de  $DFS$  și are un copil  $v$  în arbore astfel încât nici un vârf din subarboarele dominate de  $v$  nu este conectat cu un strămoș 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.  $u.b = \min\{v.d \mid v \text{ descoperit pornind din } u \text{ în cursul } DFS \text{ și } v.color \neq alb\}$ ;
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  $u.articulație = 1$  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( $G$ )

```

1: timp = 0
2: for  $u \in V$  do
3:    $u.color = alb$ 
4:    $u.\pi = NIL$ 
5:    $u.subarb = 0$ 
6:    $u.articulație = 0$ 
7: for  $u \in V$  do
8:   if  $u.color = alb$  then
9:      $EXPLOARE(u)$ 
10:    if  $u.subarb > 1$  then
11:       $u.articulație = 1$ 

```

#### EXPLOARE( $u$ )

```

1:  $u.d = u.b = timp + +$ 
2:  $u.color = gri$ 
3: for  $v \in succs(u)$  do
4:   if  $u.c = alb$  then
5:      $v.\pi = u$ 
6:      $u.subarb + +$ 
7:      $EXPLOARE(v)$ 
8:      $u.b = \min\{u.b, v.b\}$ 
9:     if  $u.\pi \neq NIL \wedge v.b \geq u.d$  then
10:        $u.articulație = 1$ 
11:   else
12:      $u.b = \min\{u.b, v.d\}$ 

```

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 vârfului este gri. O muchie din  $G$  aparține 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 vârfului 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).

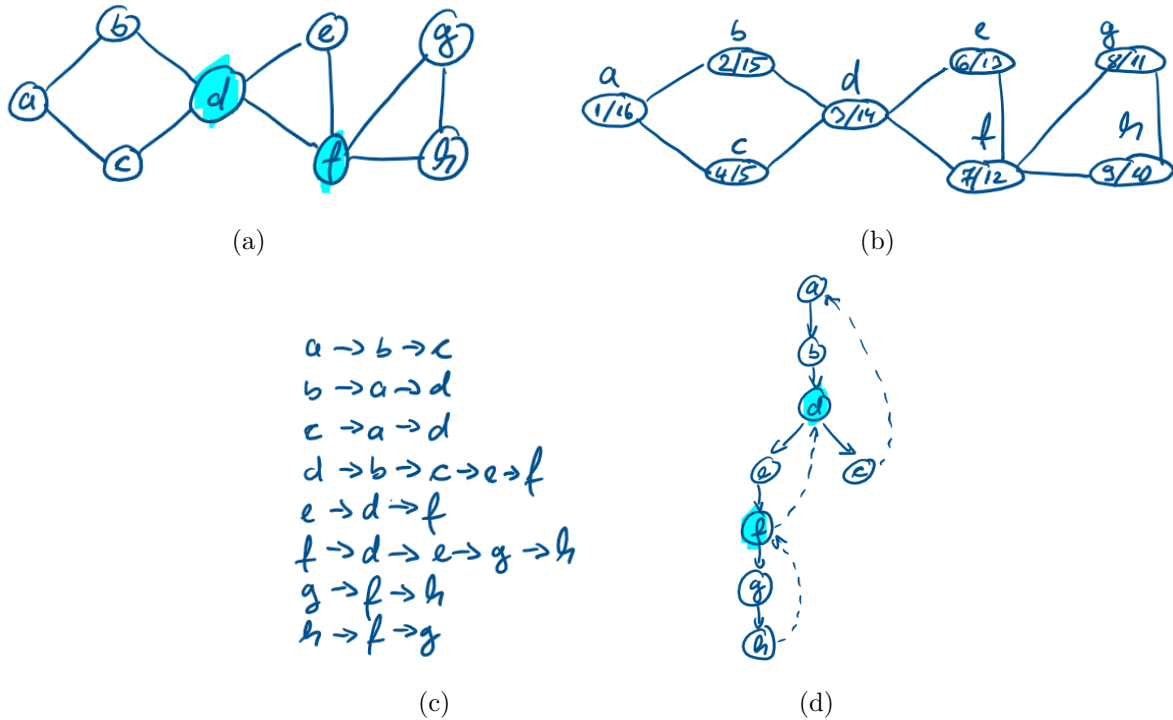


Figura 2: Exemplu de căutare a punctelor de articulații într-un graf.

## 12.2 Punți

**Teorema 12.2.** 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  $z, z.d \leq u.d$ .

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  $u \rightarrow v$  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.

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( $G$ )**

```

1: for  $u \in V$  do
2:    $u.color = alb$ 
3:    $u.\pi = NIL$ 
4:    $u.punte = 0$ 
5:  $timp = 0$ 
6: for  $u \in V$  do
7:   if  $u.color = alb$  then
8:      $EXPLOARE\_PUNTI(u)$ 

```

#### **EXPLOARE\_PUNTI( $u$ )**

```

1:  $u.d = u.b = timp + +$ 
2:  $u.color = gri$ 
3: for  $v \in succs(u)$  do
4:   if  $u.color = alb$  then
5:      $v.\pi = u$ 
6:      $EXPLOARE\_PUNTI(v)$ 
7:      $u.b = \min\{u.b, v.b\}$ 
8:     if  $v.b > u.d$  then
9:        $v.punte = 1$ 
10:  else
11:    if  $u.\pi \neq v$  then
12:       $u.b = \min\{u.b, v.d\}$ 

```

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

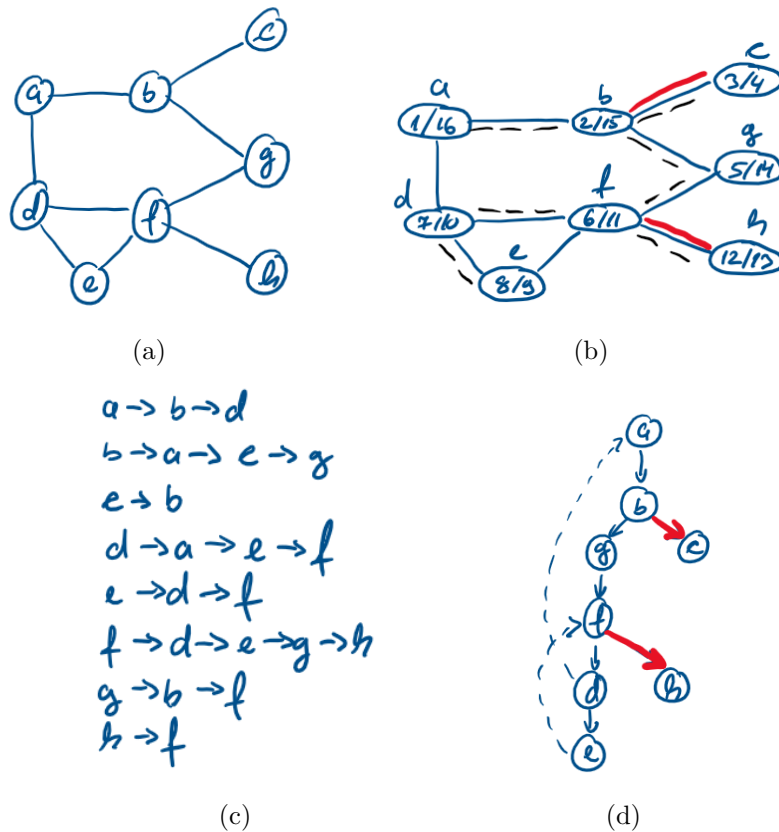


Figura 3: Exemplu de căutare a muchiilor punți într-un graf.

### 12.3 Componente biconectate

**Teorema 12.3.** Fie  $G = (V, E)$  un graf neorientat și  $u$  un vârf nesingular din  $G$  ( $\text{succs}(u) \neq \emptyset$ ). Vârful  $u$  este vârf de start al unei bicomponente a lui  $G$  dacă și numai dacă în urma  $\text{DFS}(G)$  există cel puțin un subarbore  $\text{ARB}(v)$  dominat de  $u$  astfel încât pentru orice muchie  $(x, z)$  - cu  $x$  în  $\text{ARB}(v)$  - descoperit în cursul  $\text{DFS}(G)$  avem  $u.d \leq z.d$ .

Figura 4 prezintă un graf și componentele biconectate.

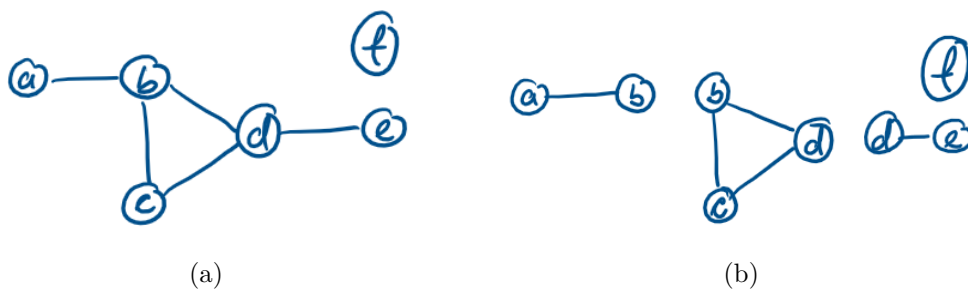


Figura 4: Componente biconectate.

Mai jos este prezentat algoritmul pentru căutarea componentelor biconectate.

#### BICOMPONENTE( $G$ )

1: **for**  $u \in V$  **do**

```

2:   u.color = alb
3: timp = 0
4: componente =  $\emptyset$ 
5: for u ∈ V do
6:   if u.color = alb then
7:     if sucs(u) ≠  $\emptyset$  then
8:       componente = componente ∪ EXPLORARE_BICOMP(u)
9:     else
10:      u.color = negru
11:      componente = componente ∪ {u}
12: return componente

```

#### EXPLORARE\_BICOMP(*u*)

```

1: u.d = u.b = timp + +
2: u.color = gri
3: componenteu =  $\emptyset$ 
4: for v ∈ succs(u) do
5:   if v.color = alb then
6:     componenteu = componenteu ∪ EXPLORARE_BICOMP(v)
7:     u.b = min{u.b, v.b}
8:     if u.d ≤ v.b then
9:       componenteu = componenteu ∪ {COLECTARE(u, v)}
10:  else
11:    u.b = min{u.b, v.d}
12: return componenteu

```

#### COLECTARE(*start*, *vecin*)

```

1: start.color = negru
2: componenta = PARCURGERE ∪ {start}
3: start.color = gri
4: return componenta

```

#### PARCURGERE(*varf*)

```

1: varf.color = negru
2: componenta = {varf}
3: for v ∈ succs(varf) do
4:   if v.color = gri then
5:     componenta = componenta ∪ PARCURGERE(v)
6: return componenta

```