Proiectarea Algoritmilor

Curs 7 – Puncte de articulație, Punți, Drumuri minime

Bibliografie

[1] Giumale – Introducere în Analiza Algoritmilor cap. 5.3, 5.4, 5.4.1

[2] Cormen – Introducere în Algoritmi cap. Heap-uri binomiale (20), Heap-uri Fibonacci (21), Drumuri minime de sursă unica - primele 2 subcapitole (25.1 și 25.2)

[3] R. Sedgewick, K. Wayne - Algorithms and Data Structures Fall 2007 – Curs Princeton - http://www.cs.princeton.edu/~rs/AlgsDS07/06PriorityQueues.pdf

[4] Heap Fibonacci: http://www.cse.yorku.ca/~aaw/Jason/FibonacciHeapAnimation.html

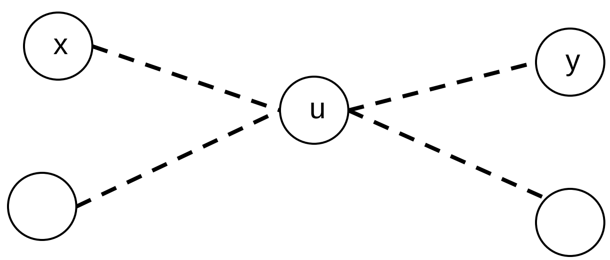
Obiective

- “Descoperirea” algoritmilor de:
  - Identificare a punctelor de articulație;
  - Identificare a punților;
  - Identificare a drumurilor de cost minim.

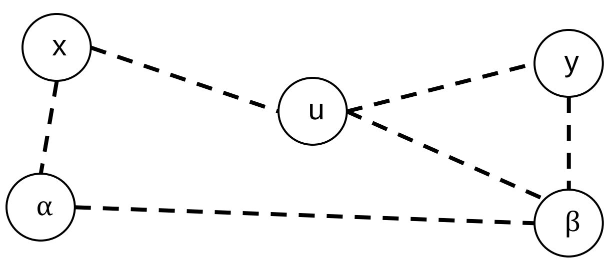
- Identificarea structurilor de date necesare pentru reducerea complexității acestor algoritmi.

Puncte de articulație. Def. Exemple

- Definiție: G = (V,E) graf neorientat, u∈V. U este punct de articulație dacă ∃ x,y∈V, x ≠ y, x ≠ u, y ≠ u, a.î. ∀ x..y în G trece prin u.



Orice drum x..y trece prin u → u este punct de articulație.



Exista x..α..y care nu trece prin u → u nu mai este punct de articulație!

Algoritm naiv de detectare a punctelor de articulație

- Elimină fiecare nod și verifică conectivitatea grafului rezultat:
  - Graf conex → nodul nu e punct de articulație.
  - Altfel → punct de articulație.

Complexitate?

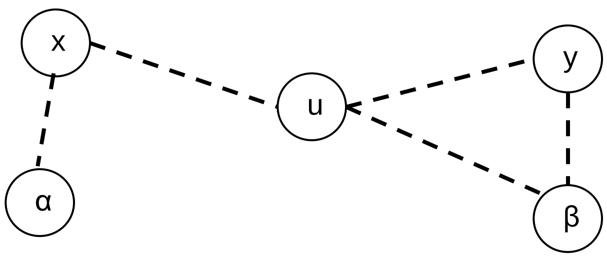
O(V(V+E))

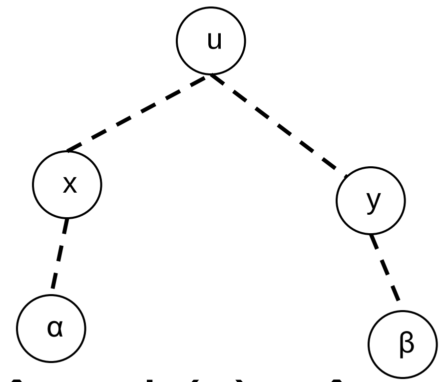
Puncte de articulație. Teoremă

- Teorema 5.15: $G = (V,E)$, graf neorientat și $u \in V$. U este punct de articulație în G $\Leftrightarrow$ în urma DFS în G una din proprietățile de mai jos este satisfăcută:
  - $p(u)$ = null și u domină cel puțin 2 subarbori;
  - $p(u) \neq$ null și $\exists v$ descendent al lui u în Arb(u) a.î. $\forall x \in$ Arb(v) și $\forall (x,z)$ parcursă de DFS(G) avem $d(z) \geq d(u)$.

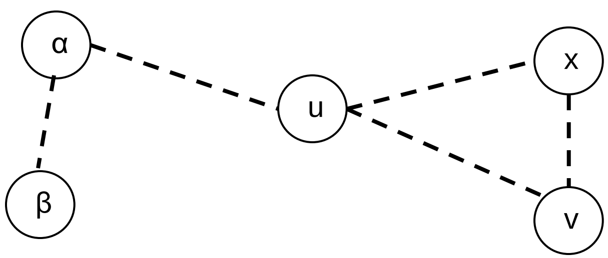
Situații posibile

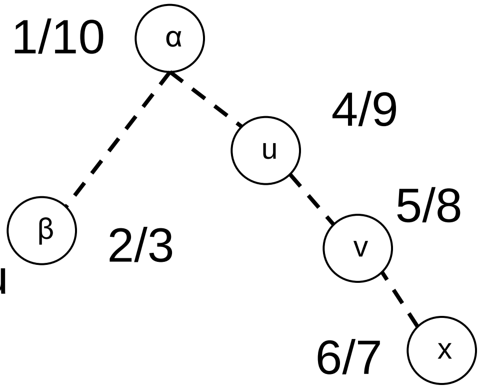
- 1) p(u) = null și u domina cel puțin 2 subarbori:





- 2) p(u) ≠ null și ∃v descendent al lui u în Arb(u) a.î. ∀x∈Arb(v) și ∀(x,z) parcursă de DFS(G) d(z) ≥ d(u):

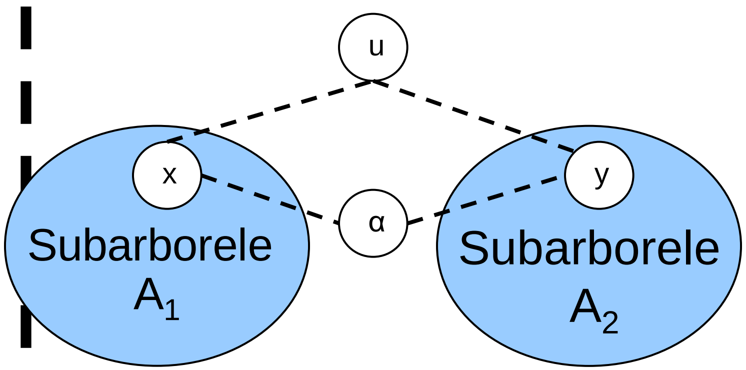




Pentru orice muchie din subarborele lui v nu există nici o muchie înapoi spre un nod descoperit înaintea lui u.

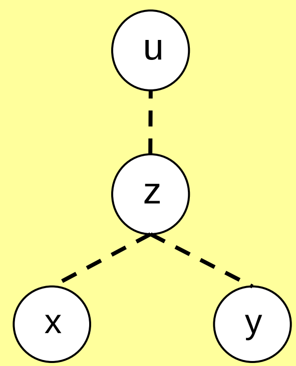
Puncte de articulație. Demonstrație teoremă (Ia)

- p(u) = null și u domină cel puțin 2 subarbori ⟹ u este punct de articulație.
- Dem (Reducere la absurd): Fie A_1 și A_2 cei 2 subarbori, $x \in A_1$, $y \in A_2$. Pp $\exists$ x..α..y și u $\notin$ x..α..y.
- z = primul nod descoperit de DFS din care se poate ajunge la x și la y. Cf. T drumurilor albe x,y ∊ Arb(z).
- Dar x,y∊Arb(u) ➔ 2 cazuri:



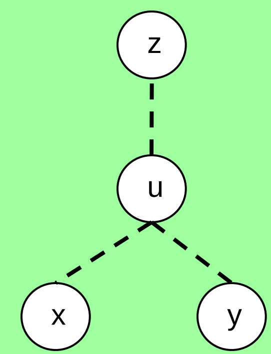
Pp $\exists$ x..α..y și u$\notin$ x..α..y.

Caz 1: d(u)<d(z):



Contradictie (1) x,y nu sunt în subarbori diferiți ai lui Arb(u).

Caz 2: d(z)<d(u):



Contradictie (1), p(u) ≠ null.

Puncte de articulație. Demonstrație teoremă (Ib)

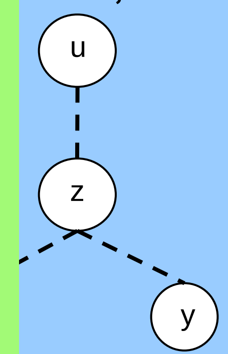
- u este punct de articulație și este descoperit în ciclul principal al DFS $\Longrightarrow$ p(u) = null și u domină cel puțin 2 subarbori.

- Dem (Reducere la absurd): Fie nodurile x și y a.î. u ∊ ∀ x..y. u = primul nod descoperit din cale (altfel u nu mai e descoperit în ciclul principal al DFS) => p(u) = null și x, y ∊ Arb(u).

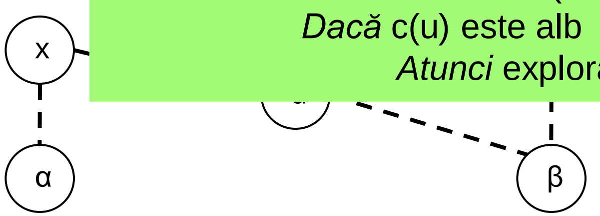
- pp că x, y
fie z rădăc
x..z..y → u
se contraz

DFS(G)
V = noduri(G)
Pentru fiecare nod u (u ∊ V)
c(u) = alb; p(u) = null; // inițializare structură date
timp = 0; // reține distanța de la rădăcina arborelui DFS pană la nodul curent
Pentru fiecare nod u (u ∊ V)
Dacă c(u) este alb
Atunci explorare(u); // explorez nodul

celași subarbore

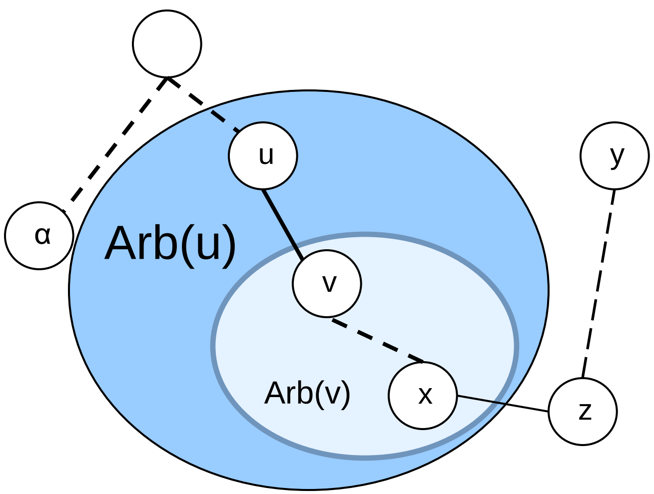


Contradicție ∃ x..z..y =>
u nu este punct de articulație



Puncte de articulație. Demonstrație teoremă (IIa)

- p(u) ≠ null și ∃ v descendent al lui u în Arb(u) a.î. ∀ x ∈ Arb(v) și ∀ (x,z) parcursă de DFS(G) are d(z) ≥ d(u) ⟹ u este punct de articulație.



Dem (Reducere la absurd): Pp. u nu e punct de articulație → ∃ w ∈ Arb(v), y ∉ Arb(u) a.î. y..w. Fie z primul nod din y..w a.î. z ∉ Arb(u) și x ultimul nod din w..y a.î. x ∈ Arb(u) → (x,z) taie frontiera Arb(u).

Dacă d(z) > d(u) → u..x, z alb la d(u) → z ∈ Arb(u) → contradicție (z ∉ Arb(u))

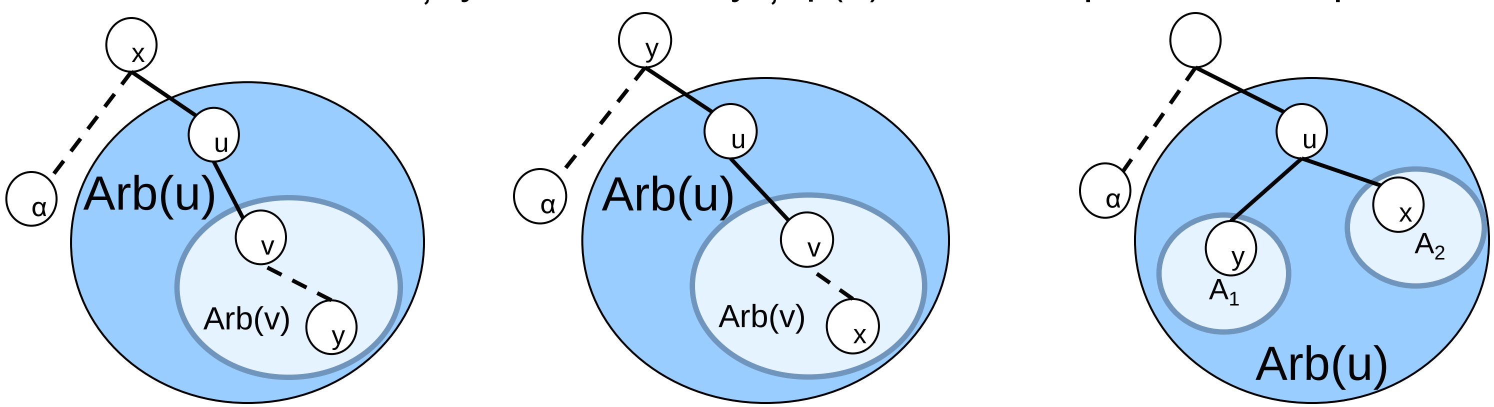
Dacă d(z) < d(u) → contradicție (ipoteza)

→ ∄ y..w → u punct de articulație

Puncte de articulație. Demonstrație teoremă (IIb)

- u este punct de articulație și nu este descoperit în ciclul principal al DFS $\Rightarrow$ p(u) $\neq$ null și $\exists$ v descendent al lui u în Arb(u) a.î. $\forall$ x $\in$ Arb(v) și $\forall$ (x,z) parcursă de DFS(G) având d(z) $\geq$ d(u).

- Dem: Fie nodurile x și y a.î. u $\in$ $\forall$ x..y și p(u) $\neq$ null. Se pot forma 3 tipuri de structuri:



- Pentru primele 2 structuri, nu trebuie sa existe muchie care sa formeze ciclu de la nici un nod din Arb(v) către vreun predecesor al lui u. Altfel $\exists$ x..y a.î. u $\notin$ x..y.

- Pentru a 3-a structura, trebuie să $\nexists$ muchie care să formeze ciclu către un predecesor al lui u de la niciun nod din cel puțin un subarbore A_1 sau A_2.

Puncte de articulație. Structuri de date.

- Structura de date de la DFS + pentru fiecare nod u ∊ V se rețin:
  - Low(u) = min{d(v) | v descoperit pornind din u în cursul DFS și c(v) ≠ alb}
  - Subarb(u) = numărul subarborilor dominați de u (dacă e ≥ 2, atunci avem un punct de articulație).

Idee algoritm

- Se aplică DFS și se salvează pentru fiecare nod până unde merge înapoi (low): low[u] = min {d(u), d(v) pentru toate muchiile înapoi (u,v), low(w) pentru toți fiii w ai lui u}.

- Pentru eficiență, trebuie ca fiii să se parcurgă înaintea părinților → ordinea inversă a d(u).

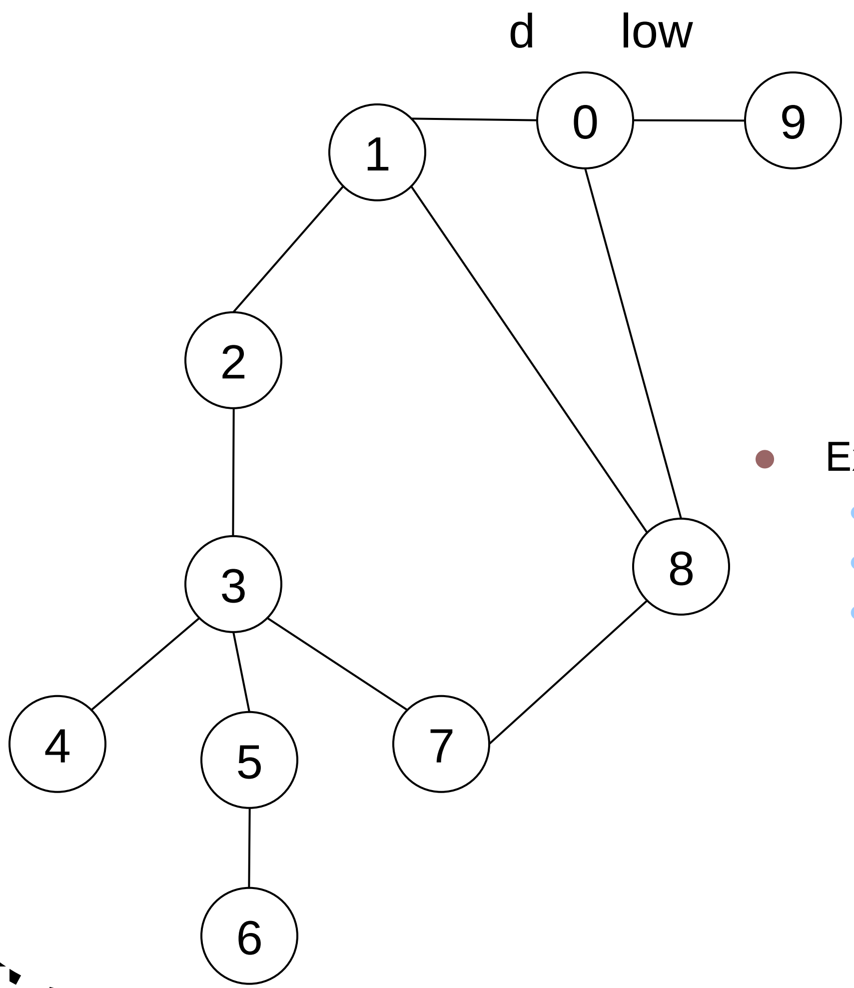
Algoritm Tarjan (I)

- Articulații (G)
  - V = noduri(G) // inițializări
  - Timp = 0;
  - **Pentru fiecare** (u ∈ V)
    - c(u) = alb;
    - d(u) = 0;
    - p(u) = null;
    - low(u) = 0;
    - subarb(u) = 0; // reține numărul de subarbori dominați de u
    - art(u) = 0; // reține punctele de articulație
  - **Pentru fiecare** (u∈V)
    - **Dacă** c(u) e alb
      - Exploreaza(u);
      - **Dacă** (subarb(u) > 1) // cazul în care u este rădăcina în arborele
        - art(u) = 1 // DFS și are mai mulți subarbori → cazul // 1 al teoremei

Algoritm Tarjan (II)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr subarbori
        // dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1;
        // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

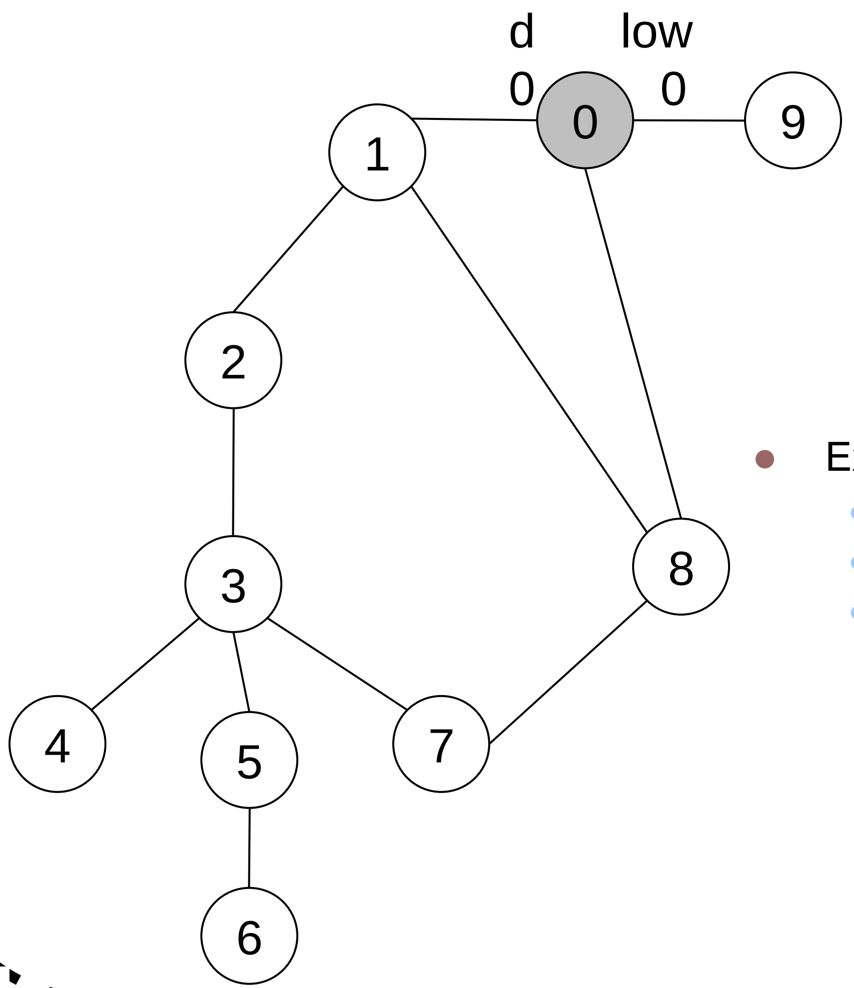
Exemplu rulare (1)



Timp = 0
C(i) = alb
D(i) = 0
Low(i) = 0
P(i) = null
Subarb(i) =0
Art(i) = 0
Exploreaza (0)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

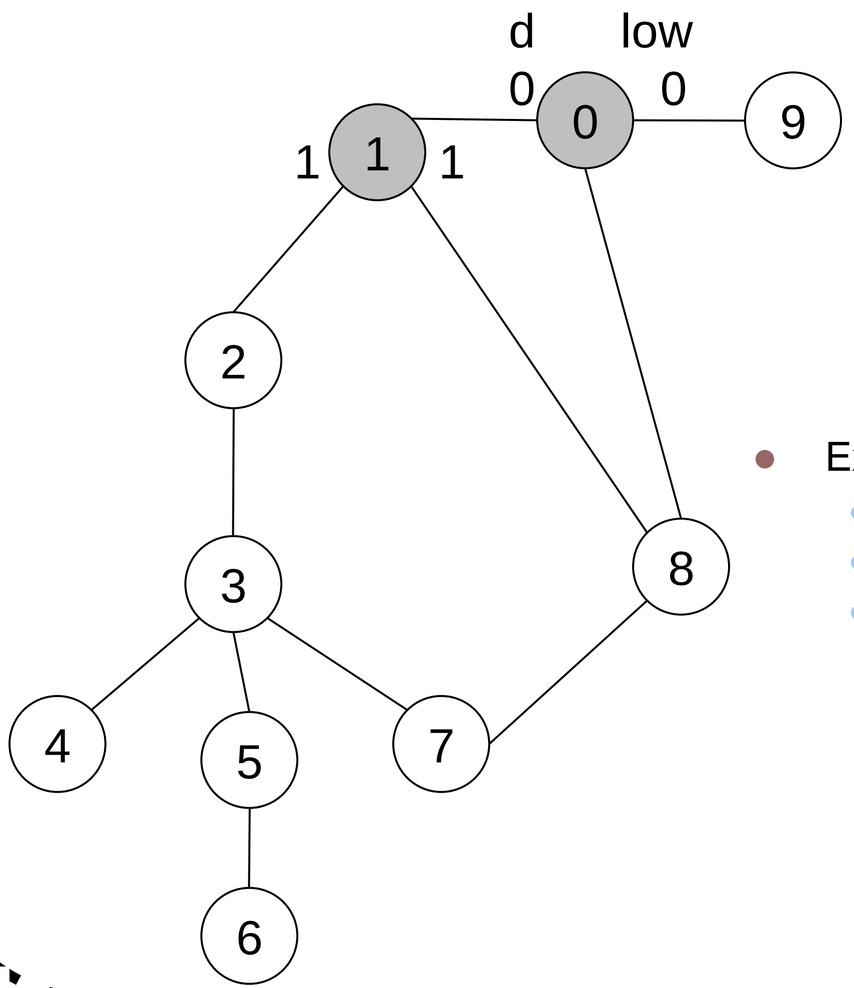
Exemplu rulare (2)



Low(0) = d(0) = 0
Timp = 1
C(0) = gri
P(1) = 0
Subarb(0) = 1
Exploreaza (1)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

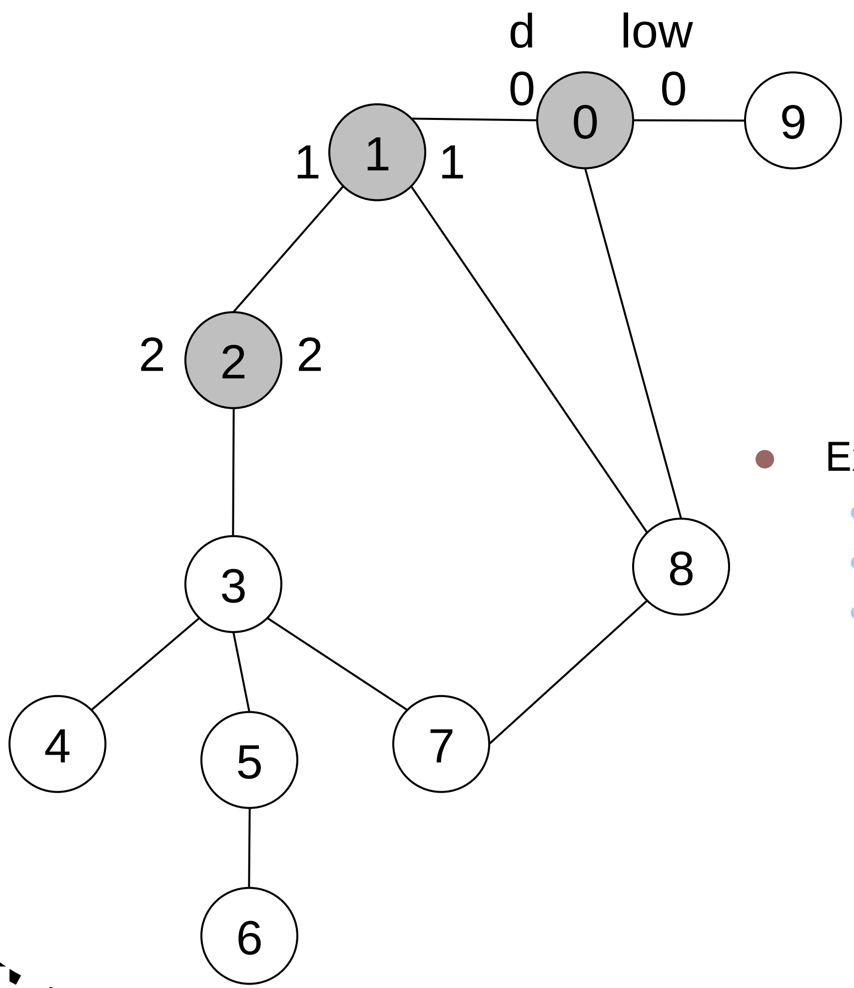
Exemplu rulare (3)



Low(1) = d(1) = 1
Timp = 2
C(1) = gri
P(2) = 1
Subarb(1) = 1
Exploreaza (2)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

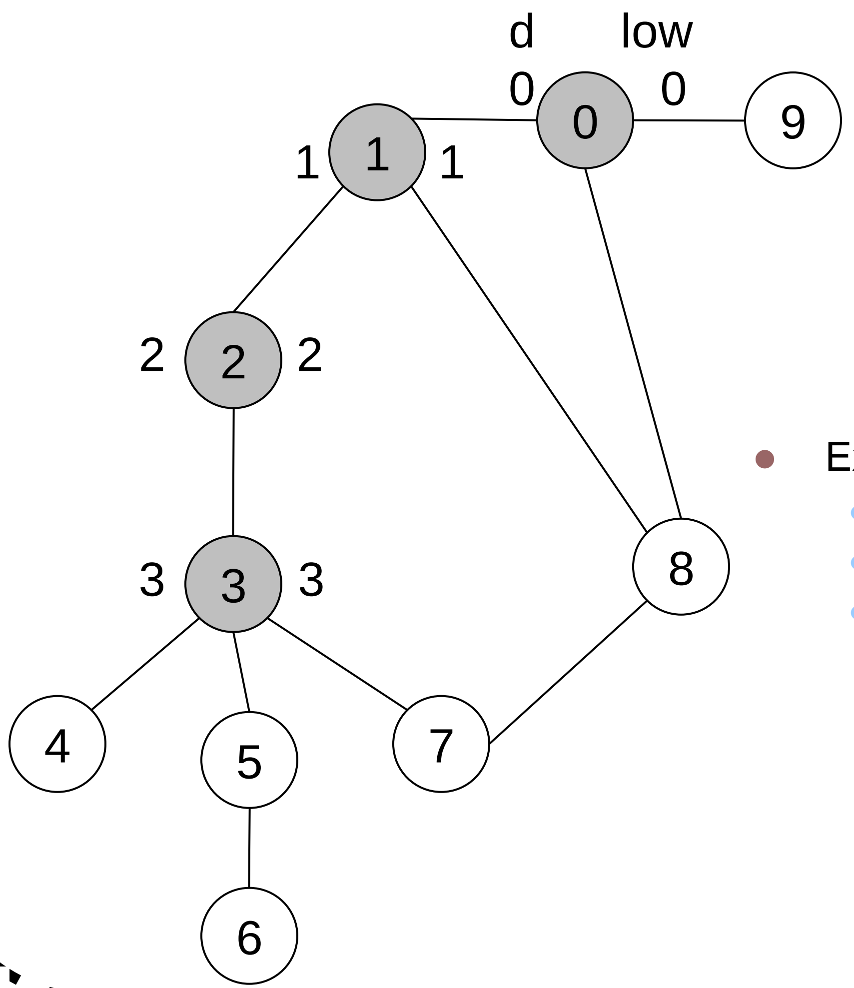
Exemplu rulare (4)



Low(2) = d(2) = 2
Timp = 3
C(2) = gri
P(3) = 2
Subarb(2) = 1
Exploreaza (3)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

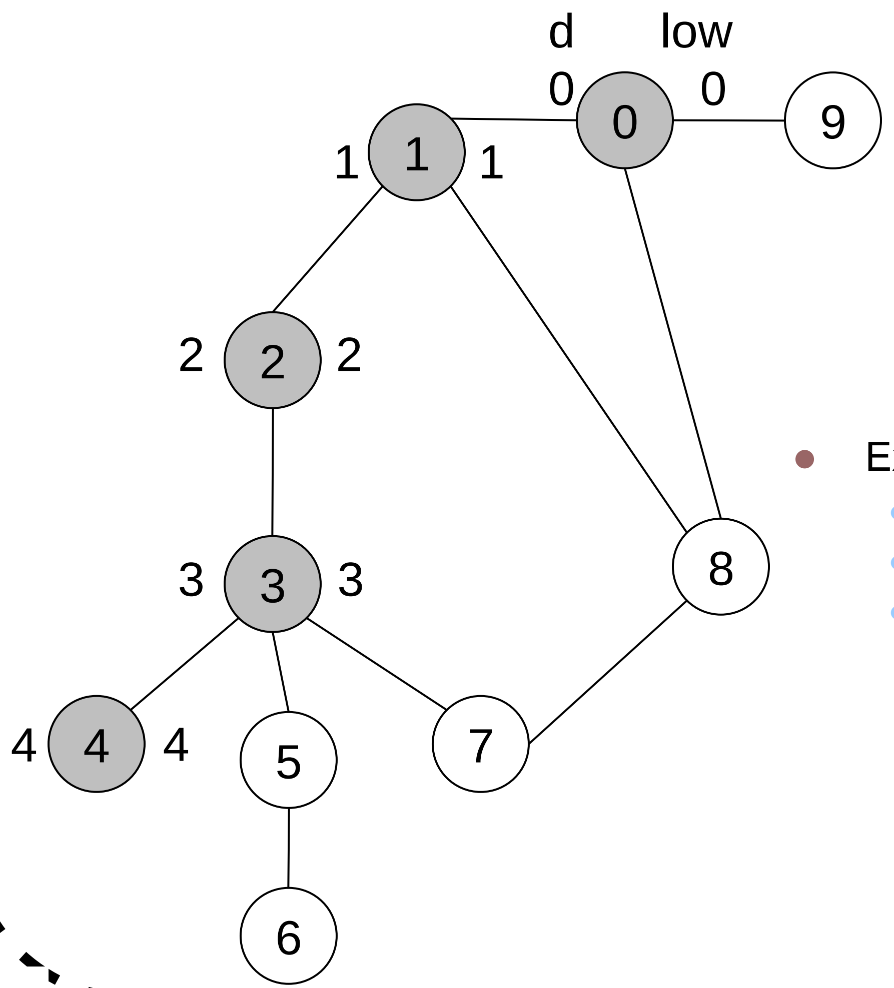
Exemplu rulare (5)



Low(3) = d(3) = 3
Timp = 4
C(3) = gri
P(4) = 3
Subarb(3) = 1
Exploreaza (4)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

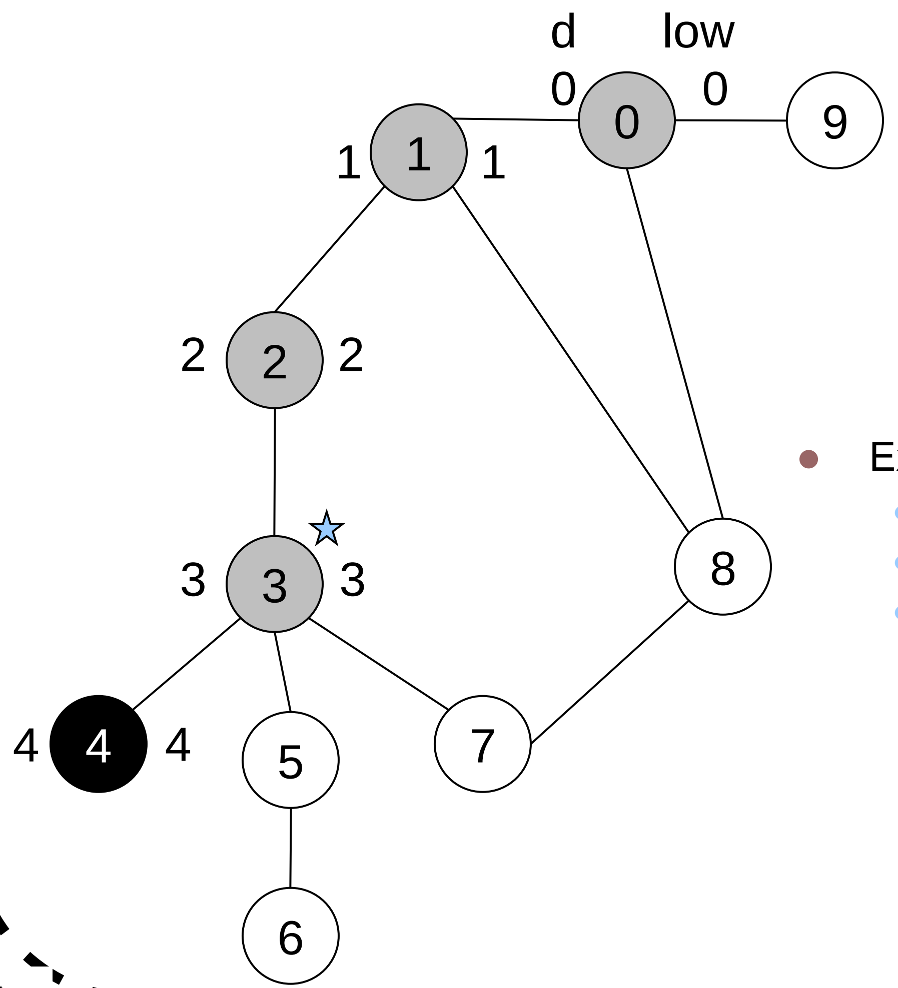
Exemplu rulare (6)



Low(4) = d(4)=4
Timp =5
C(4) =gri
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

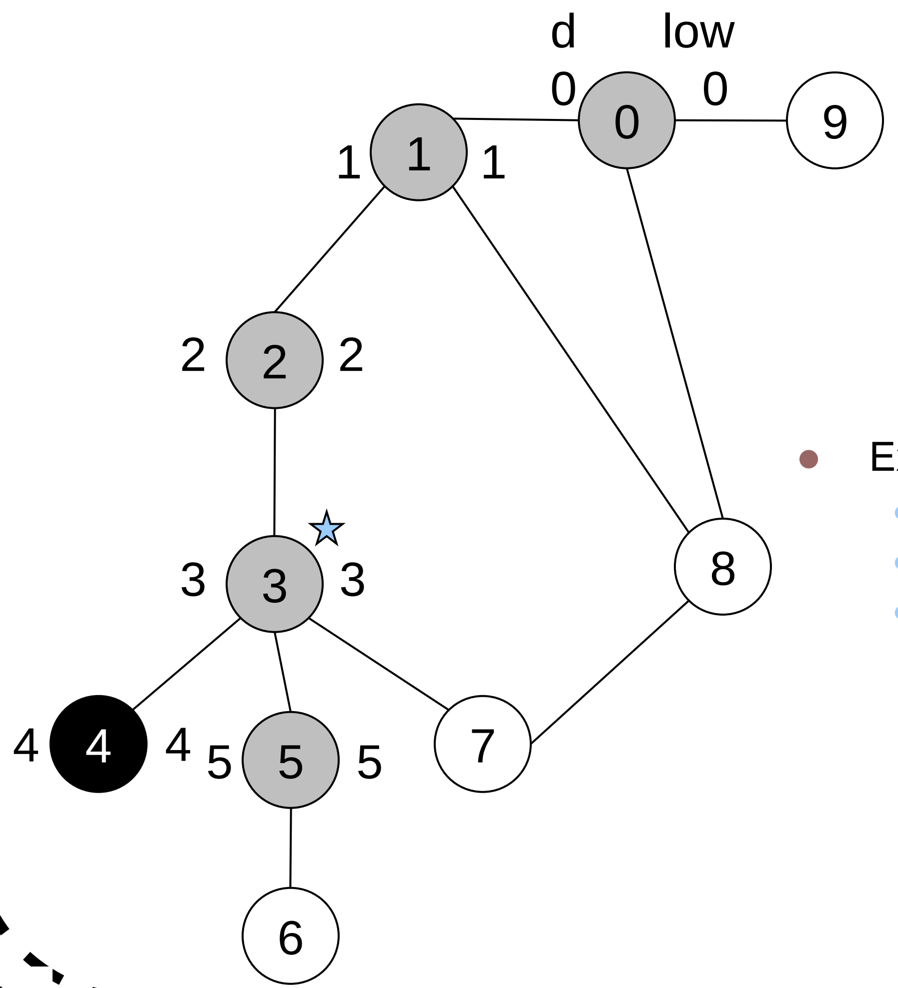
Exemplu rulare (7)



Low(4) = d(4) = 4
Timp = 5
C(4) = gri
revenire
Low (3) = min {low(3), low(4)} = 3
Low(4) > d(3) → art(3) = 1
P(5) = 3
Subarb(3) = 2
Exploreaza (5)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

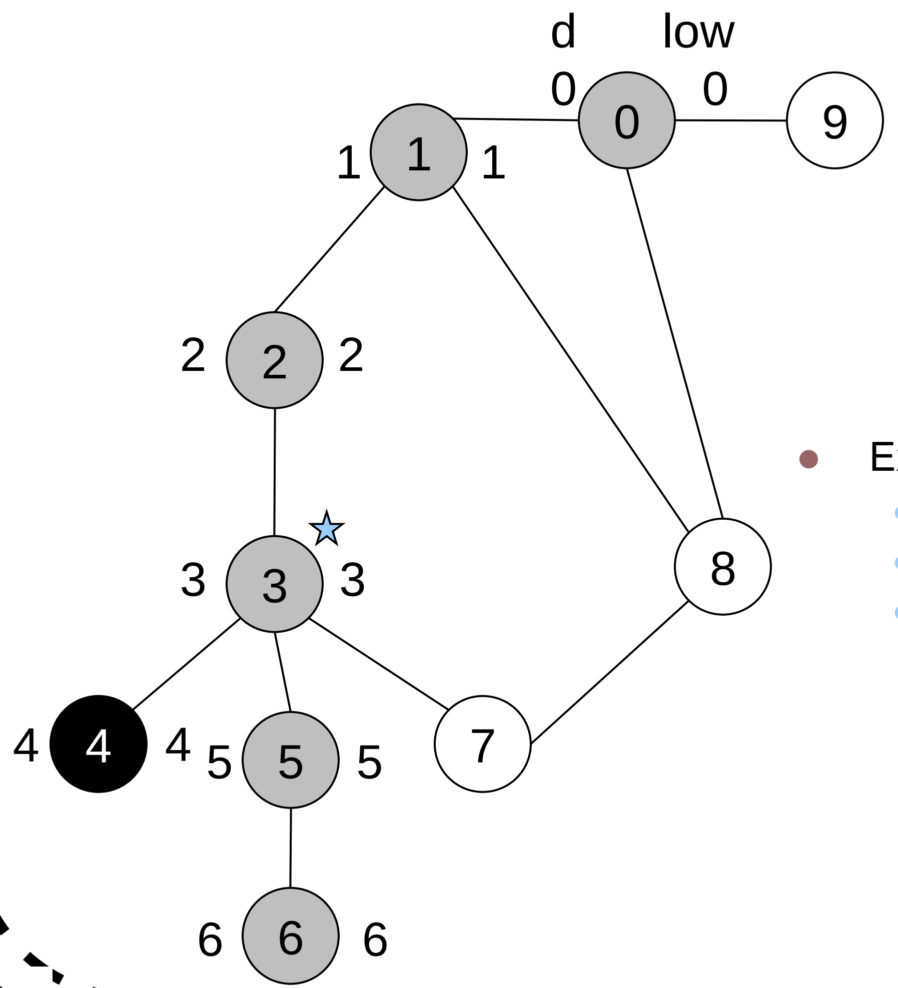
Exemplu rulare (8)



Low(5) = d(5) = 5
Timp = 6
C(5) = gri
P(6) = 5
Subarb(5) = 1
Exploreaza (6)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

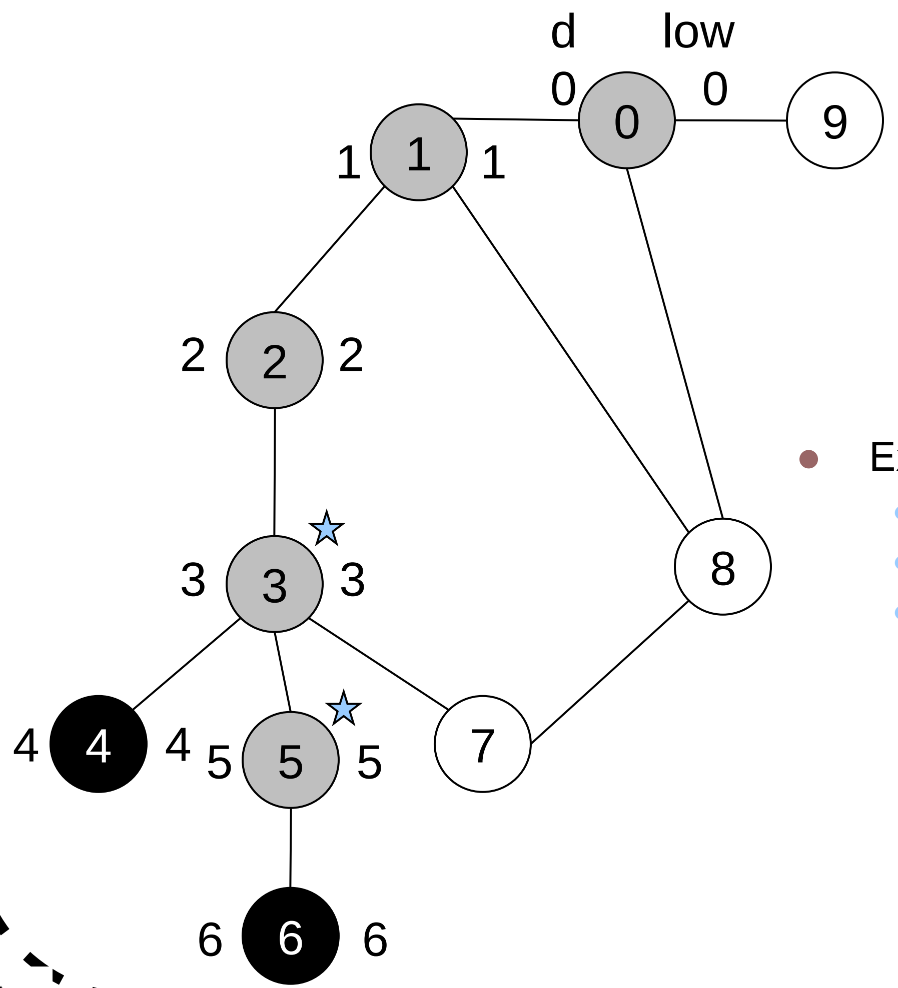
Exemplu rulare (9)



Low(6) = d(6) = 6
Timp = 7
C(6) = gri
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

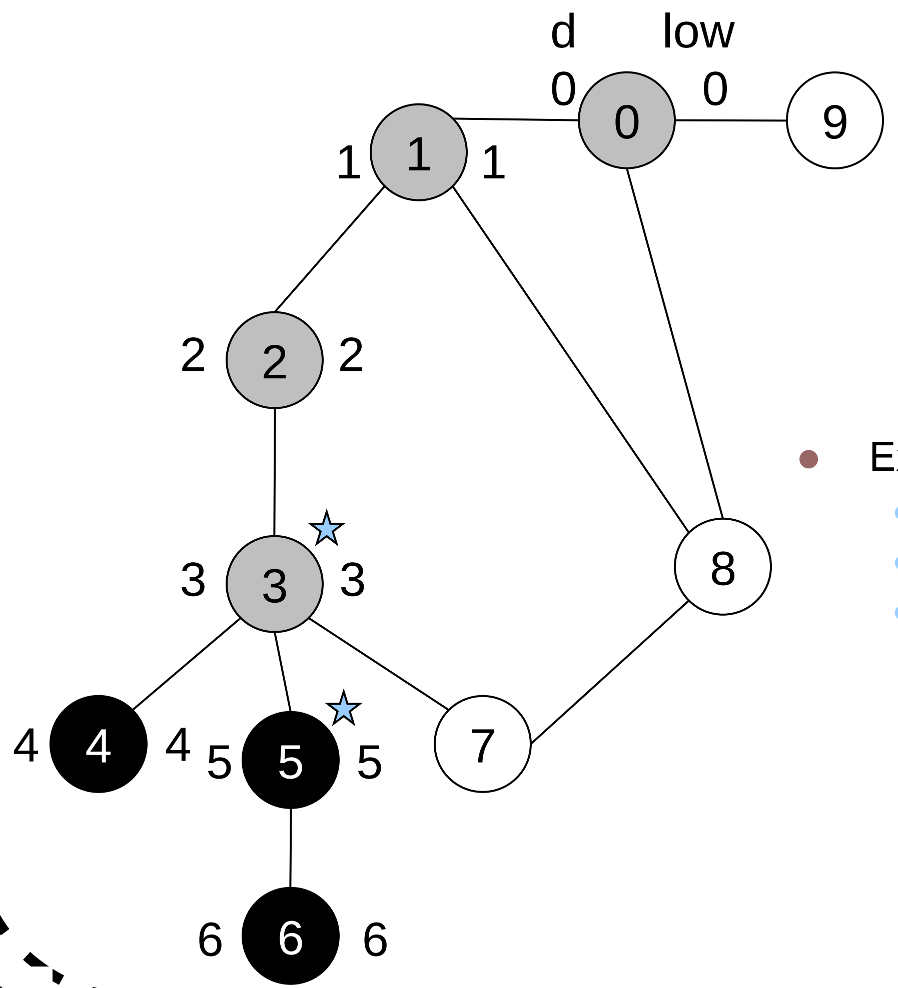
Exemplu rulare (10)



Low(6) = d(6) = 6
Timp = 7
C(6) = gri
revenire
Low (5) = min {low(5), low(6)} = 5
Low(6) > d(5) → art(5) = 1
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

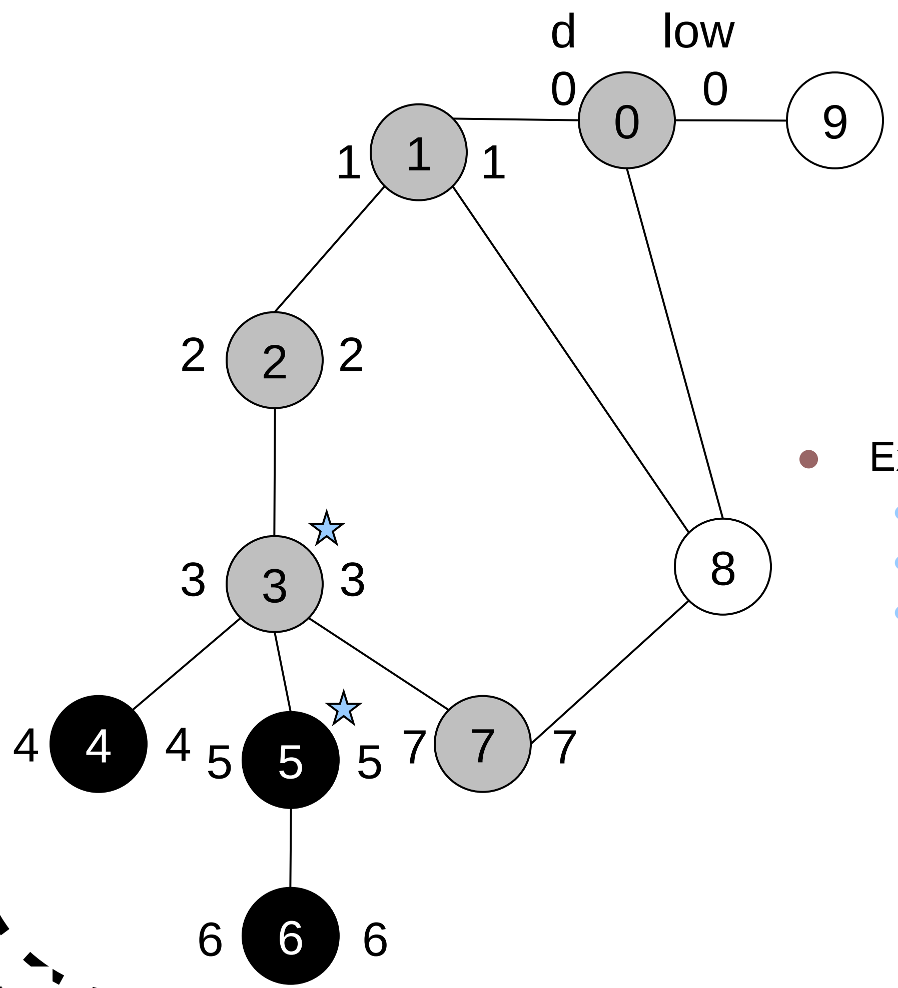
Exemplu rulare (11)



Low(5) = d(5) = 5
Timp = 7
C(5) = gri
revenire
Low (3) = min {low(3), low(5)} = 3
Low(5) > d(3) → art(3) = 1
P(7) = 3
Subarb(3) = 3
Exploreaza (7)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

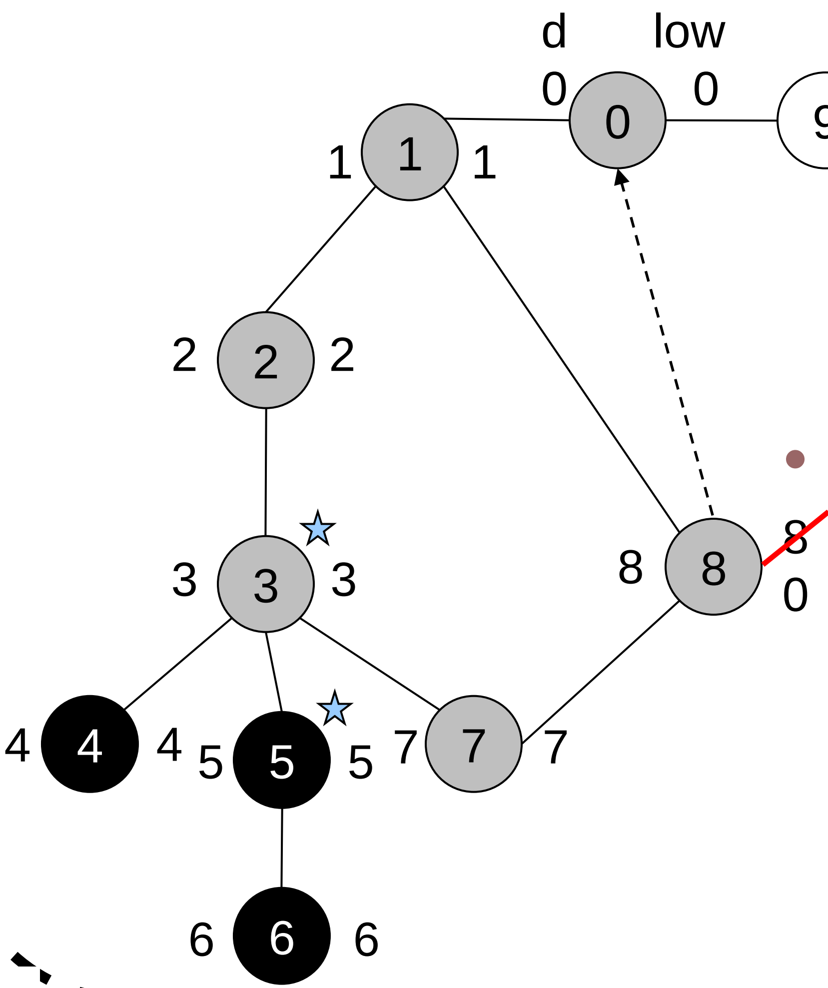
Exemplu rulare (12)



Low(7) = d(7) = 7
Timp = 8
C(7) = gri
P(8) = 7
Subarb(7) = 1
Exploreaza (8)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

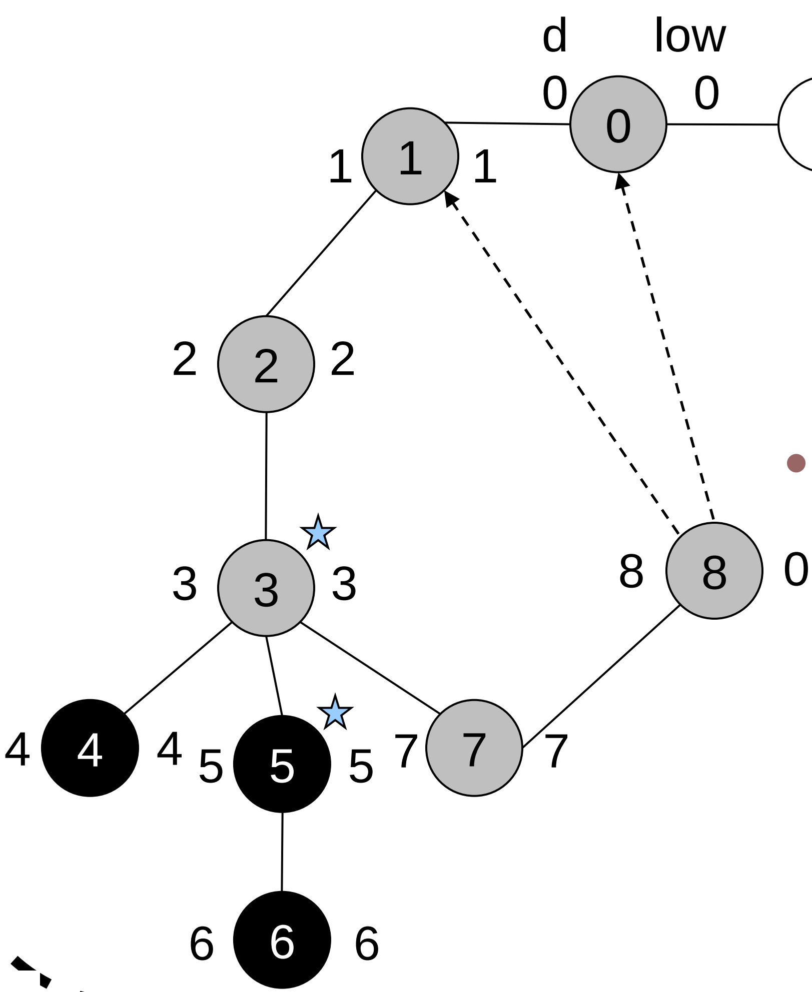
Exemplu rulare (13)



Low(8) = d(8) = 8
Timp = 9
C(8) = gri
Low(8) = min{d(0), low(8)} = 0

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

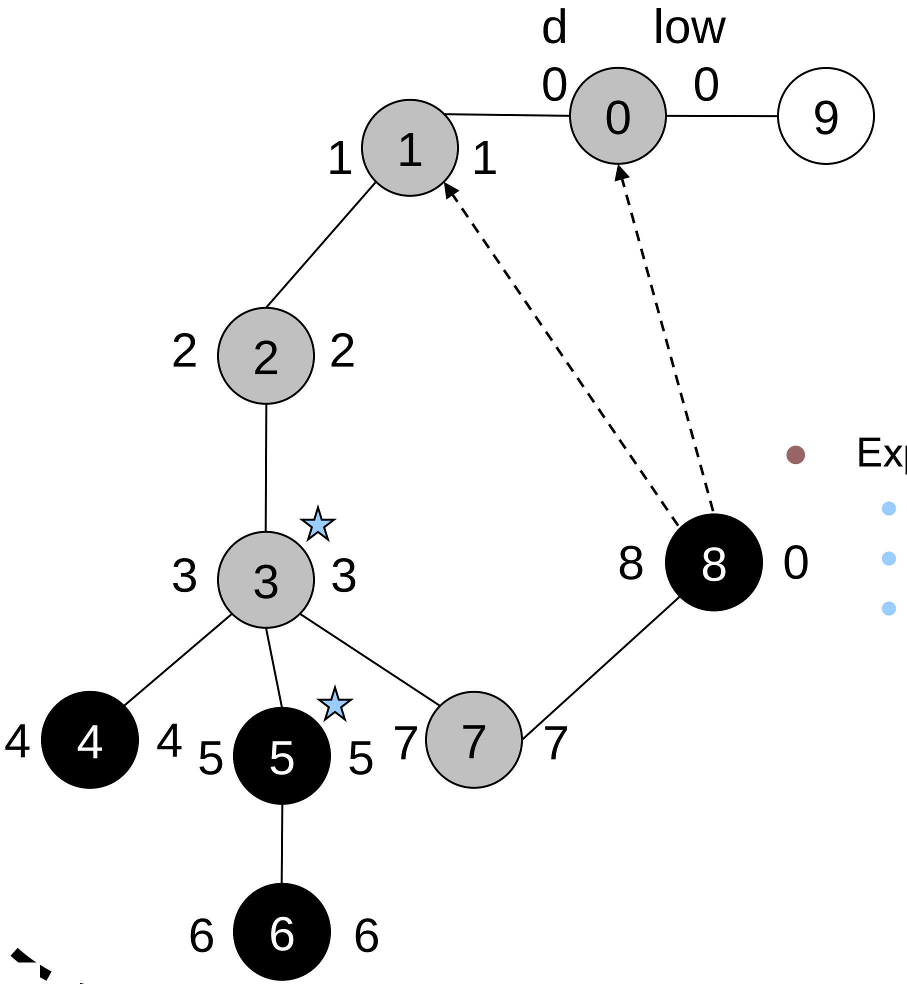
Exemplu rulare (14)



d(8) = 8
Low(8) = 0
Timp = 9
C(8) = gri
Low(8) = min{d(1), low(8)} = 0

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

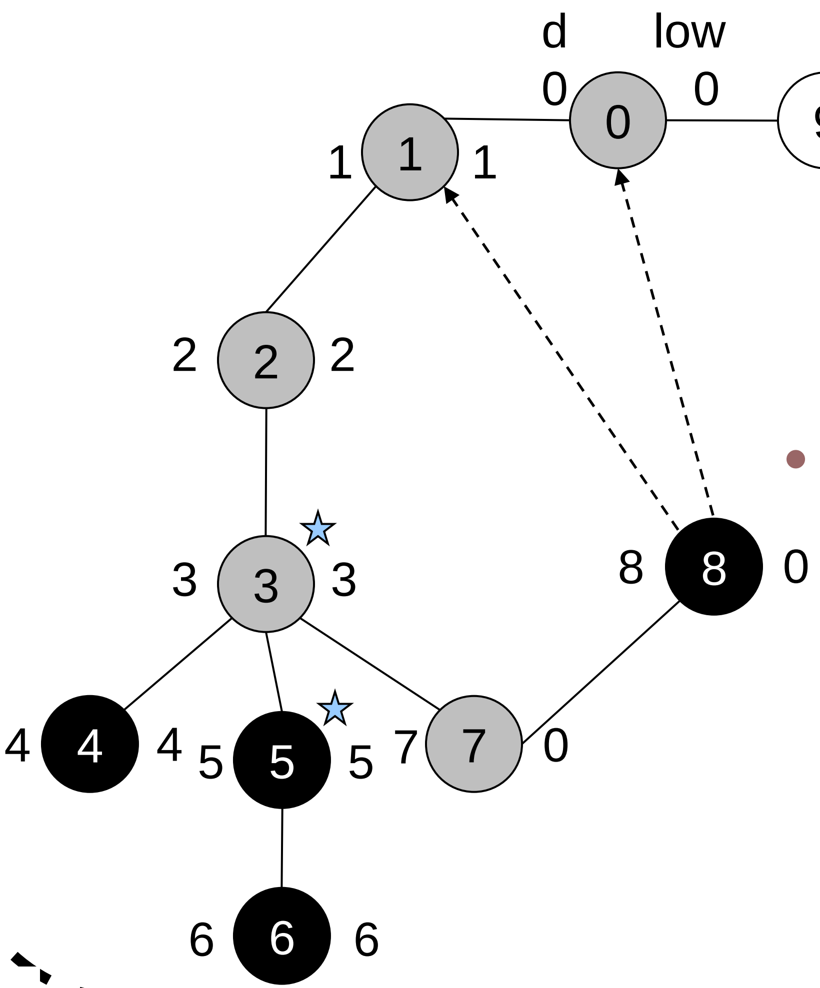
Exemplu rulare (15)



d(8) = 8
Low(8) = 0
Timp = 9
C(8) = gri
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∊ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

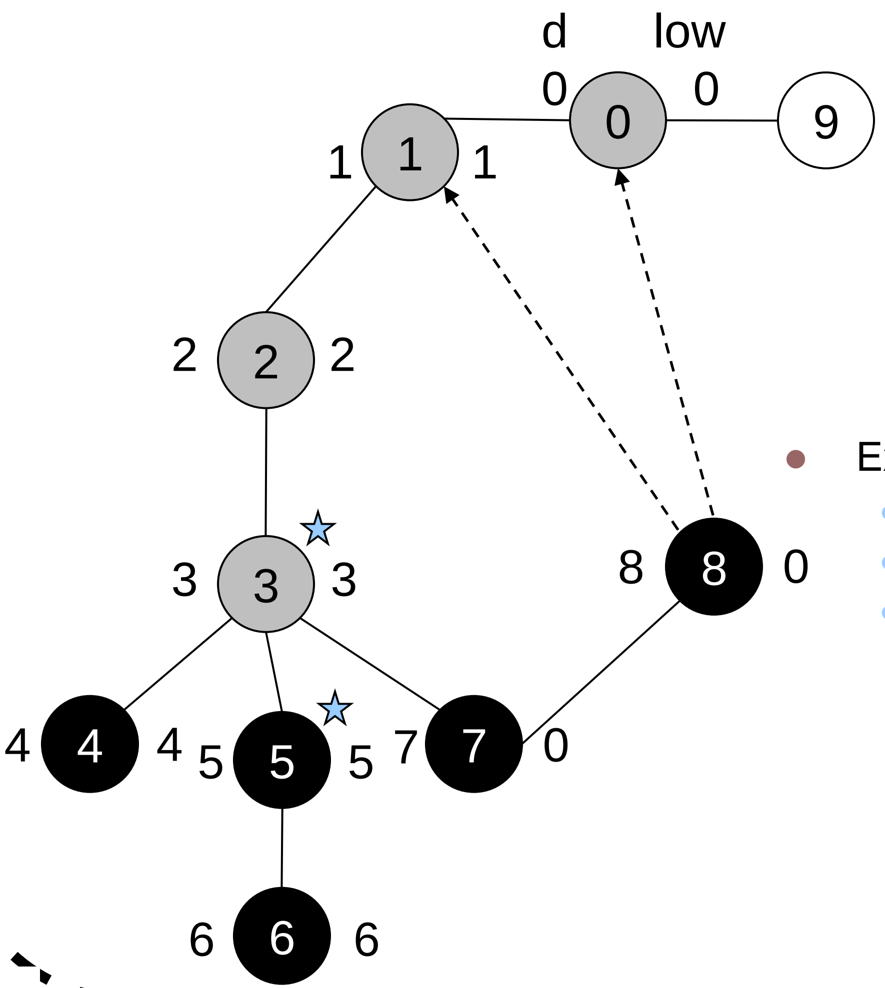
Exemplu rulare (16)



low(7) = min(low(7), low(8)) = 0
low(8) < d(7) → nu se modifică art(7)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

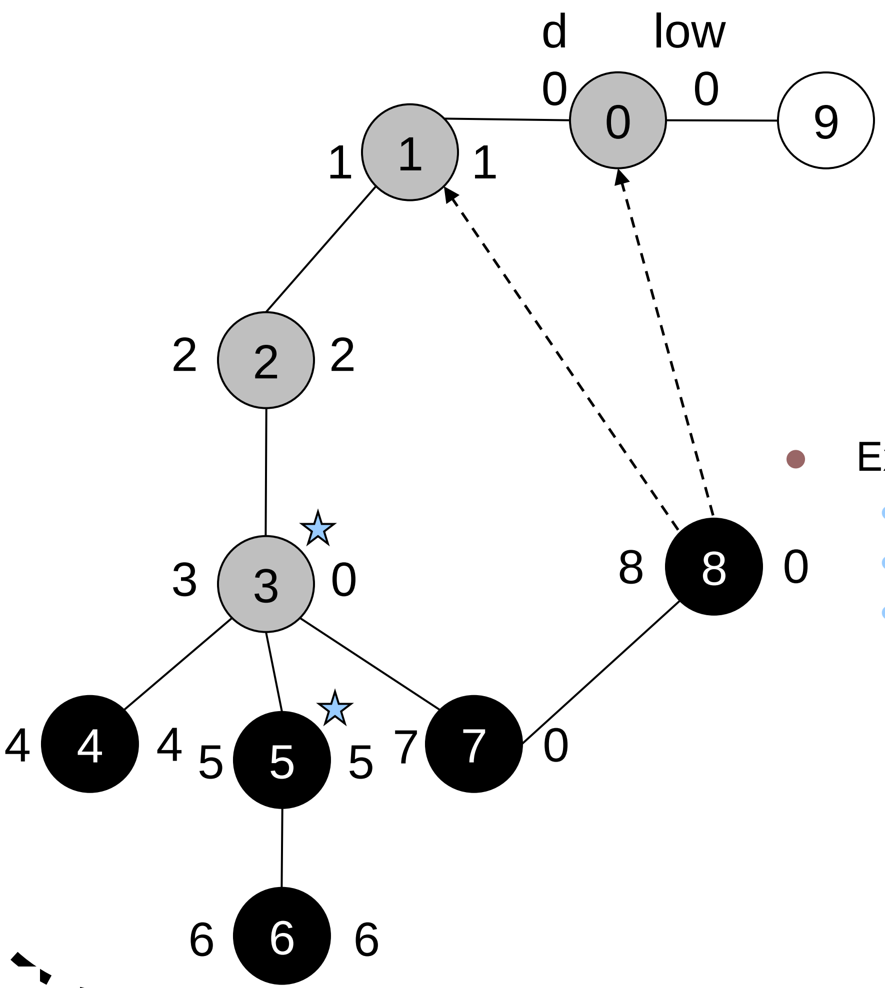
Exemplu rulare (17)



low(7) = min(low(7), low(8)) = 0
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

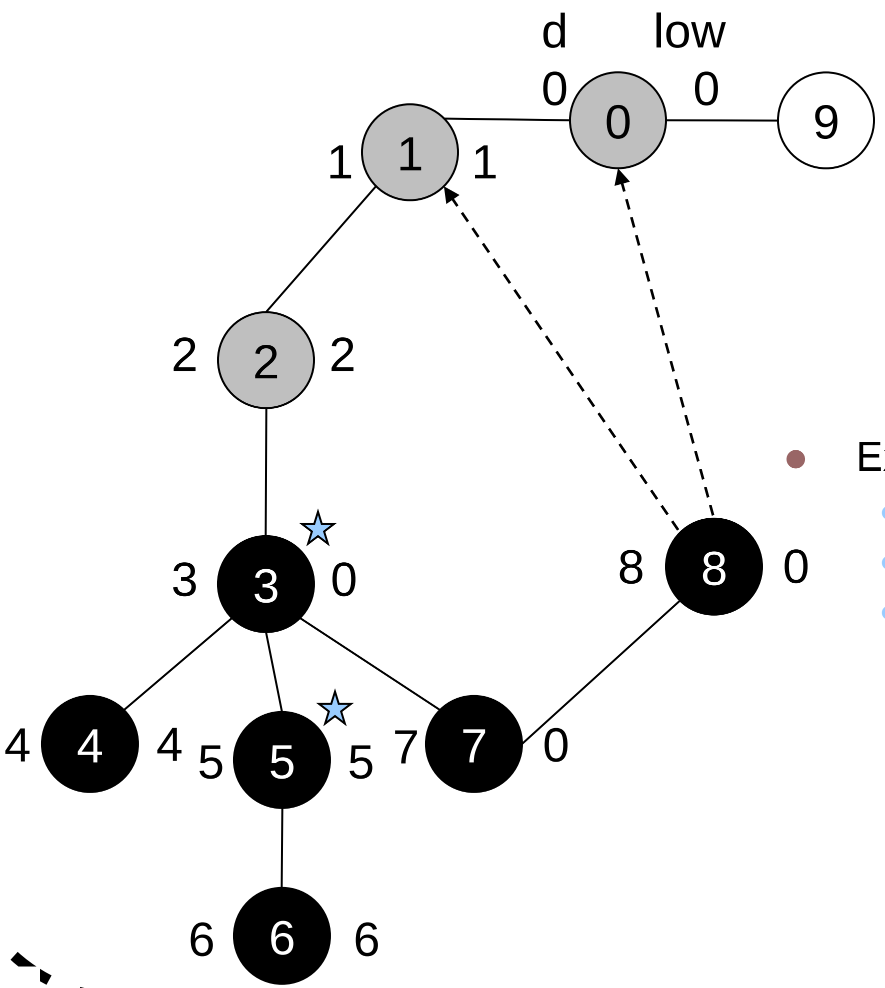
Exemplu rulare (18)



low(3) = min(low(3), low(7)) = 0
low(7) < d(3) → nu se modifică art(3)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

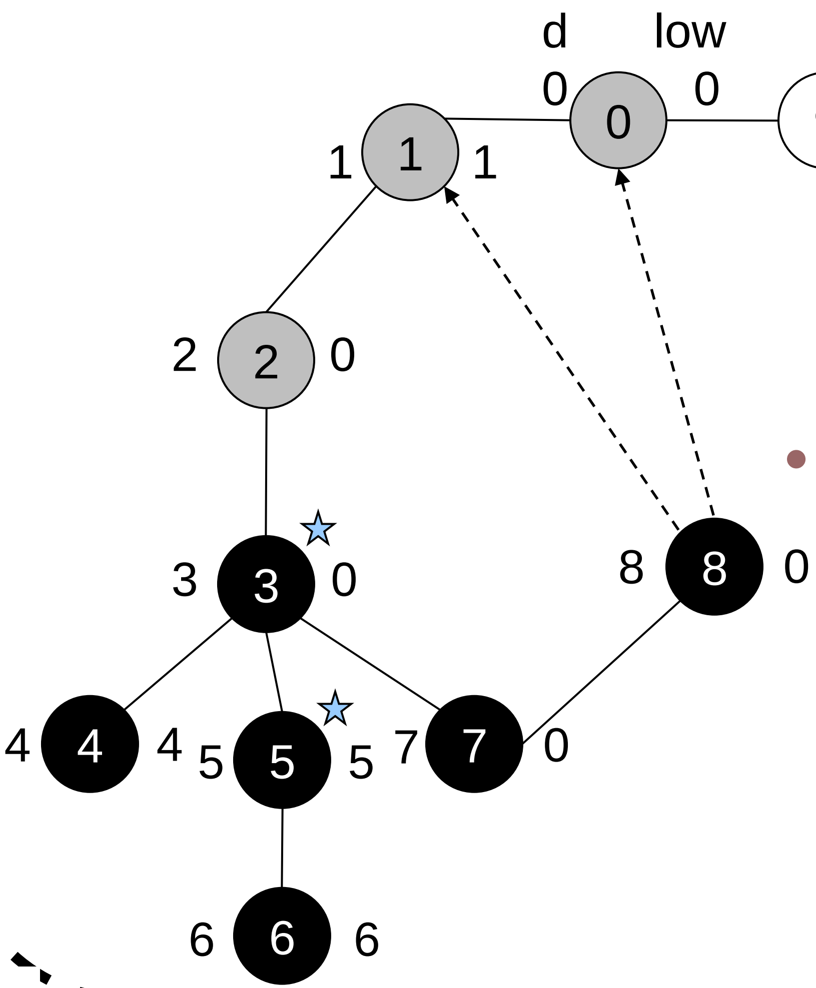
Exemplu rulare (19)



low(3) = min(low(3), low(7)) = 0
low(7) < d(3) → nu se modifică
art(3)
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

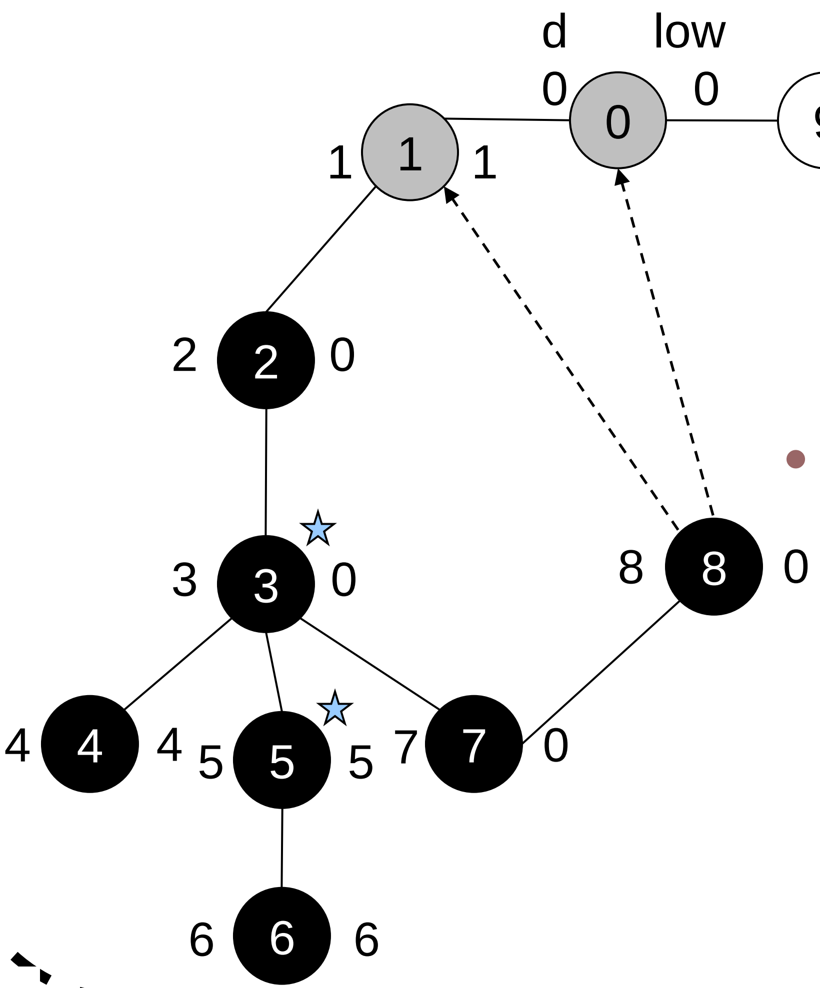
Exemplu rulare (20)



low(2) = min(low(2), low(3)) = 0
low(3) < d(2) → nu se modifică art(2)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

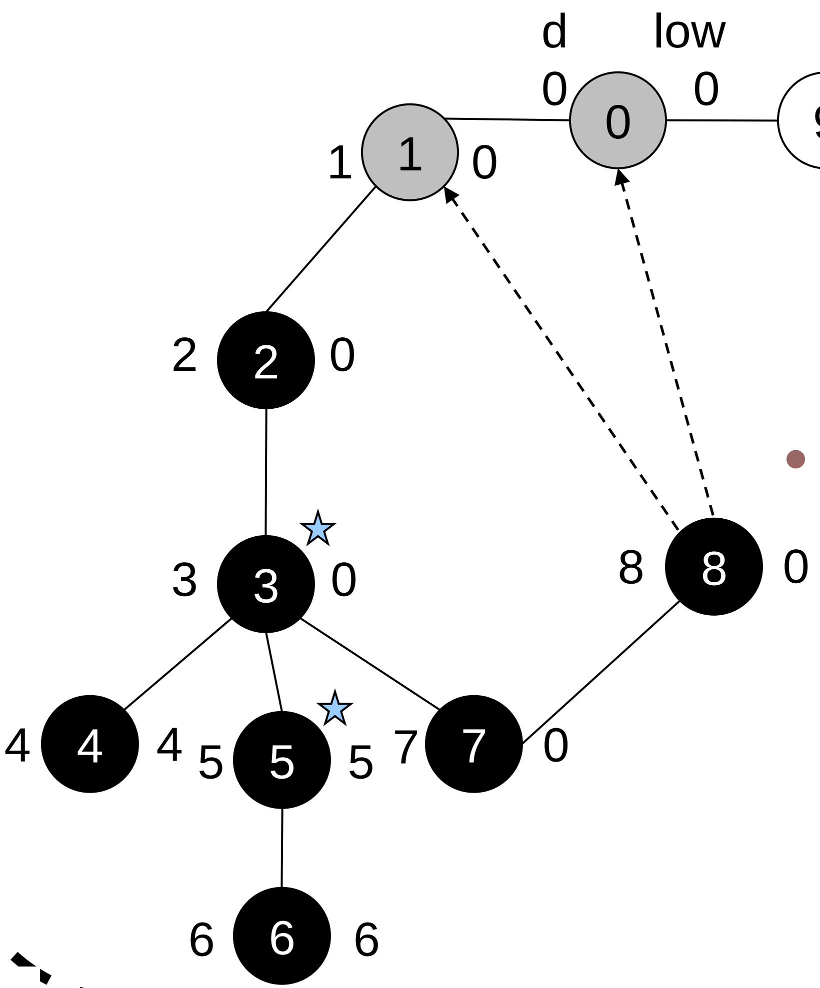
Exemplu rulare (21)



low(2) = min(low(2), low(3)) = 0
low(3) < d(2) → nu se modifică
art(2)
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

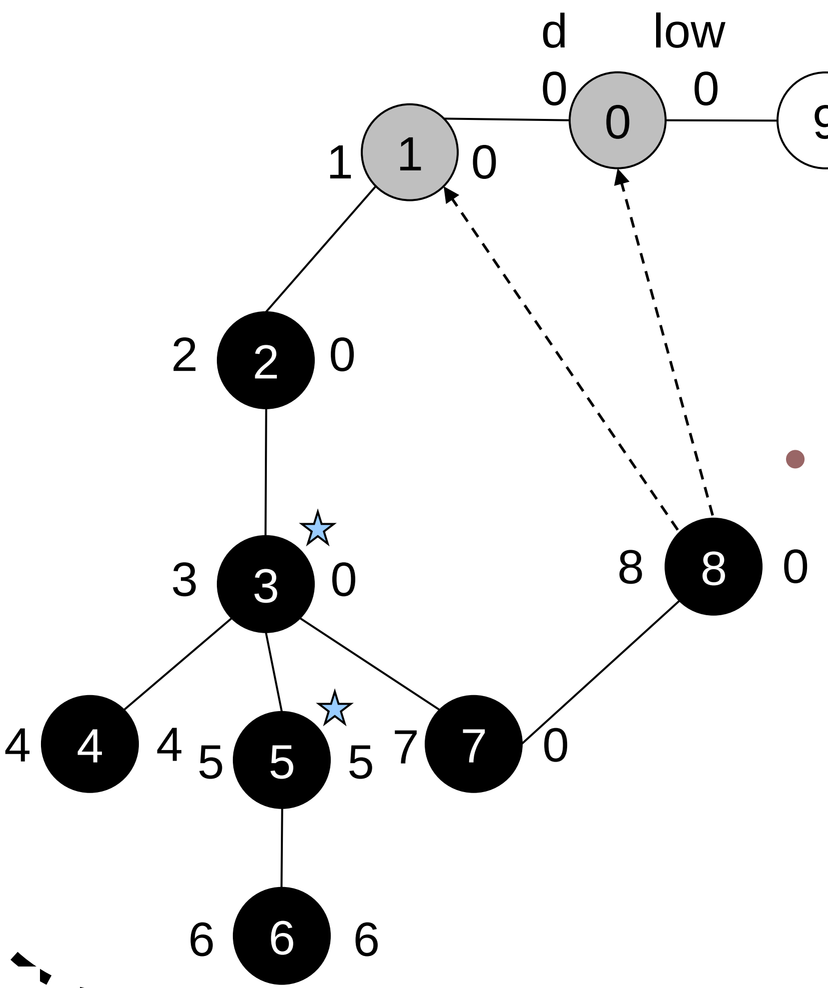
Exemplu rulare (22)



low(1) = min(low(1), low(2)) = 0
low(2) < d(1) → nu se modifică
art(1)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∊ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

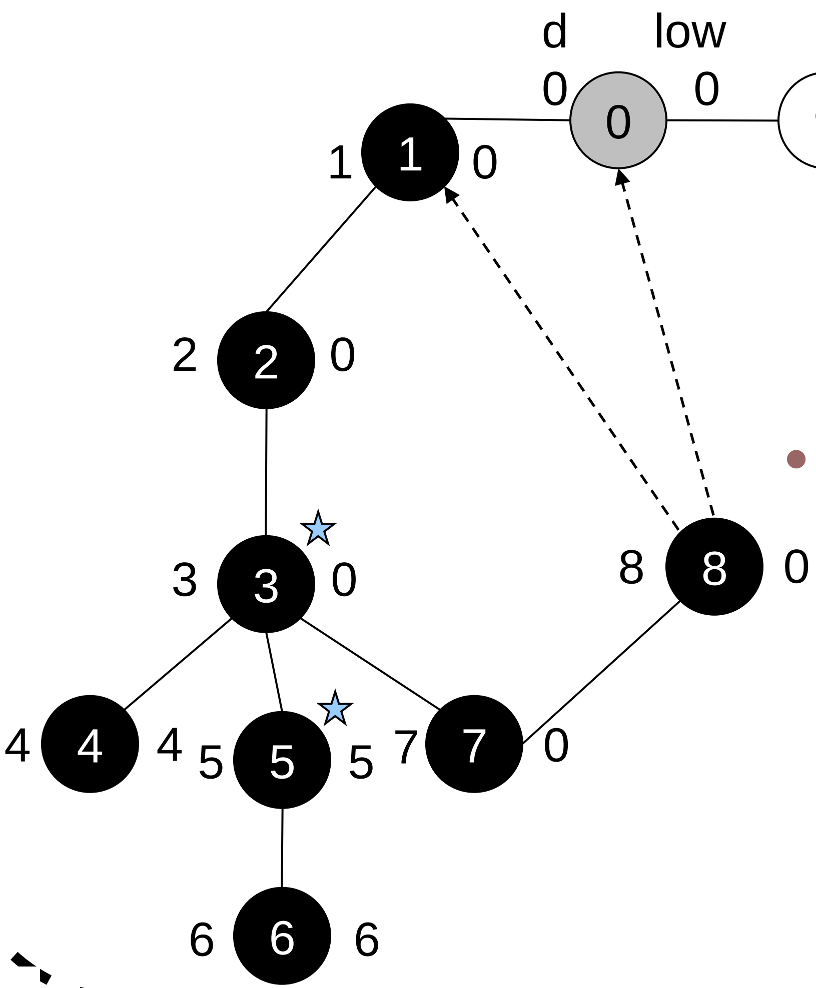
Exemplu rulare (23)



low(1) = 0
low(1) = min(low(1), d(8)) = 0

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

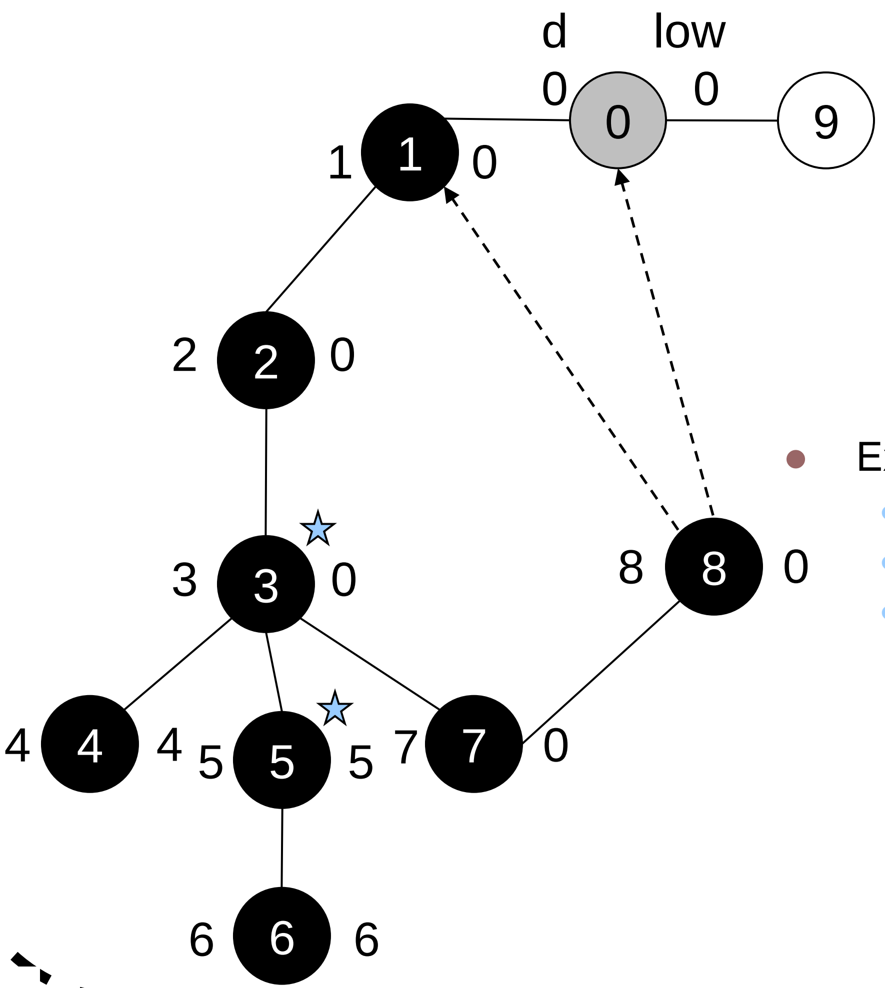
Exemplu rulare (24)



low(1) = 0
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

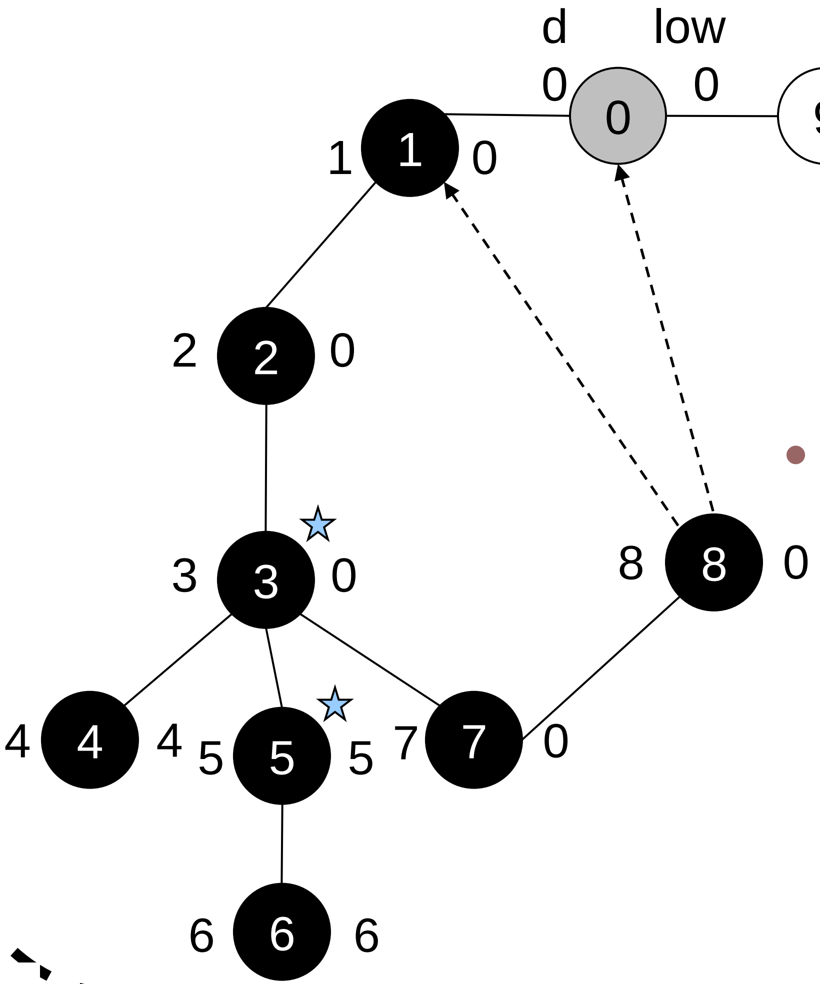
Exemplu rulare (25)



low(0) = min{low(1), low(0)} = 0
P(0) = null → continuă cu următorul copil

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

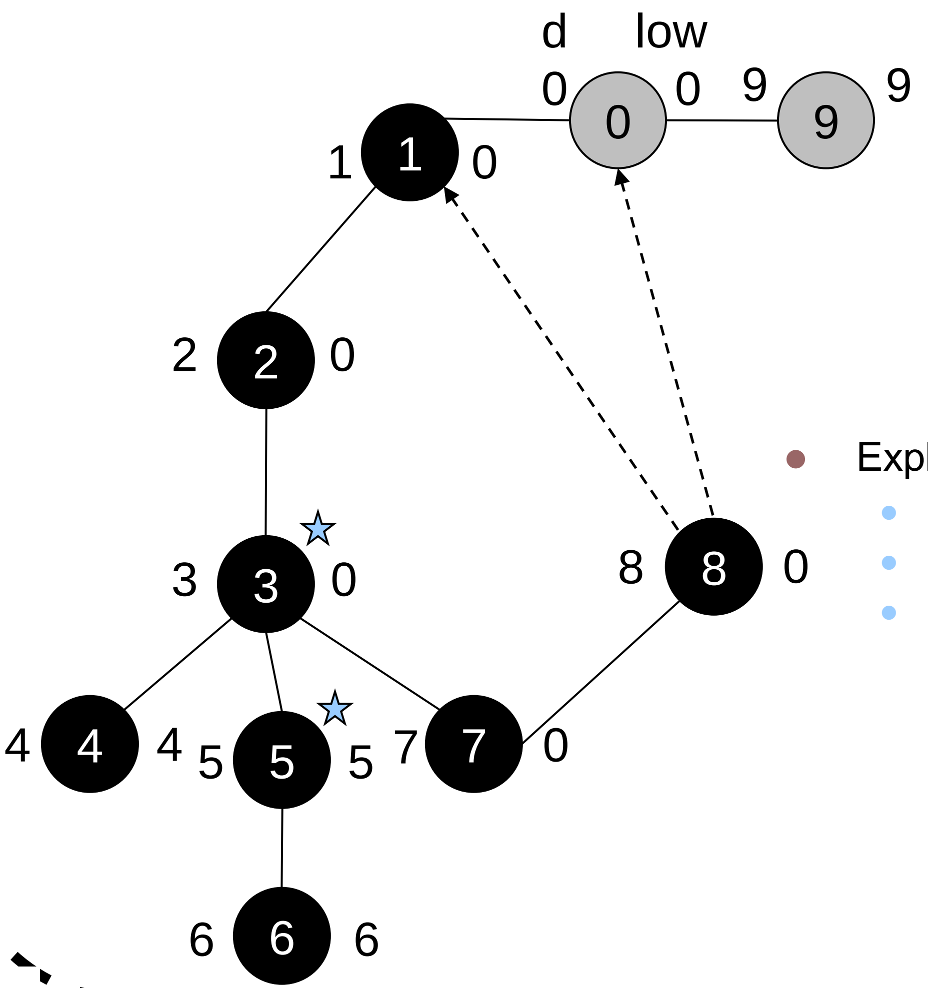
Exemplu rulare (26)



low(0) = min(low(0),d (8)) = 0
P(9) = 0
Subarb(0) = 2
Exploreaza (9)

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

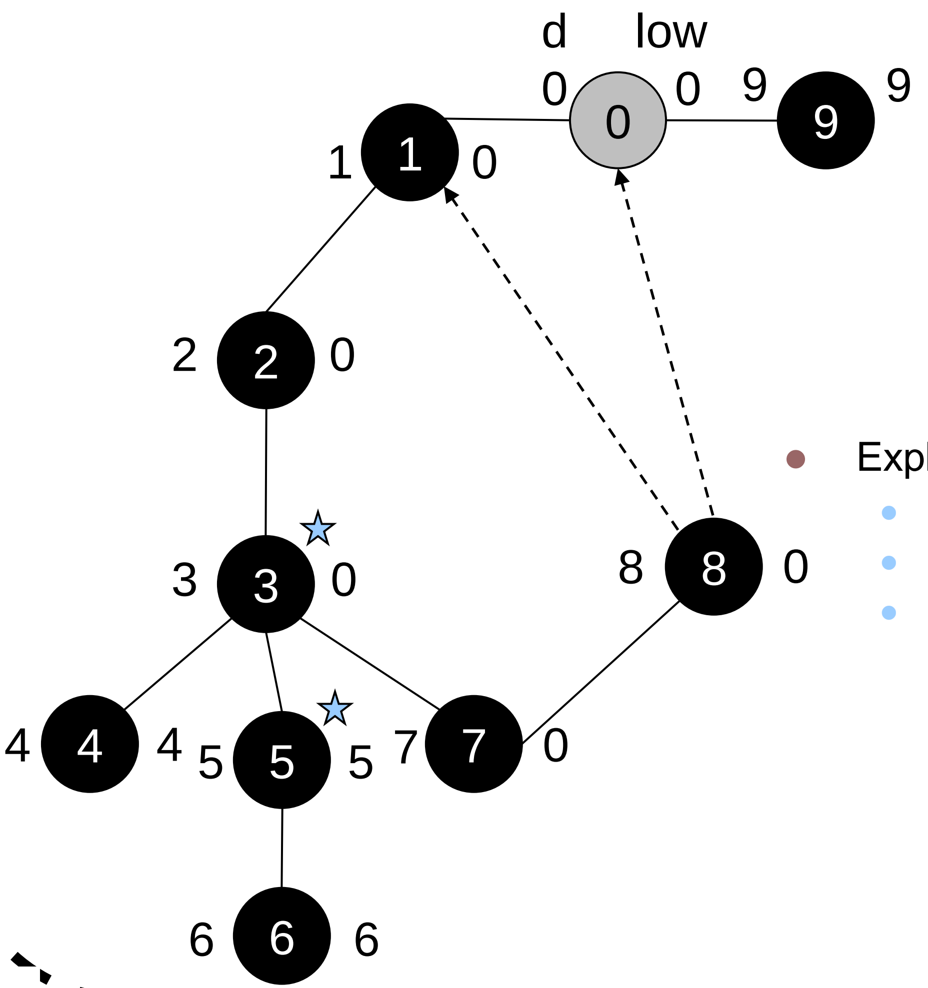
Exemplu rulare (27)



Low(9) = d(9) = 9
Timp = 10
C(9) = gri
revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

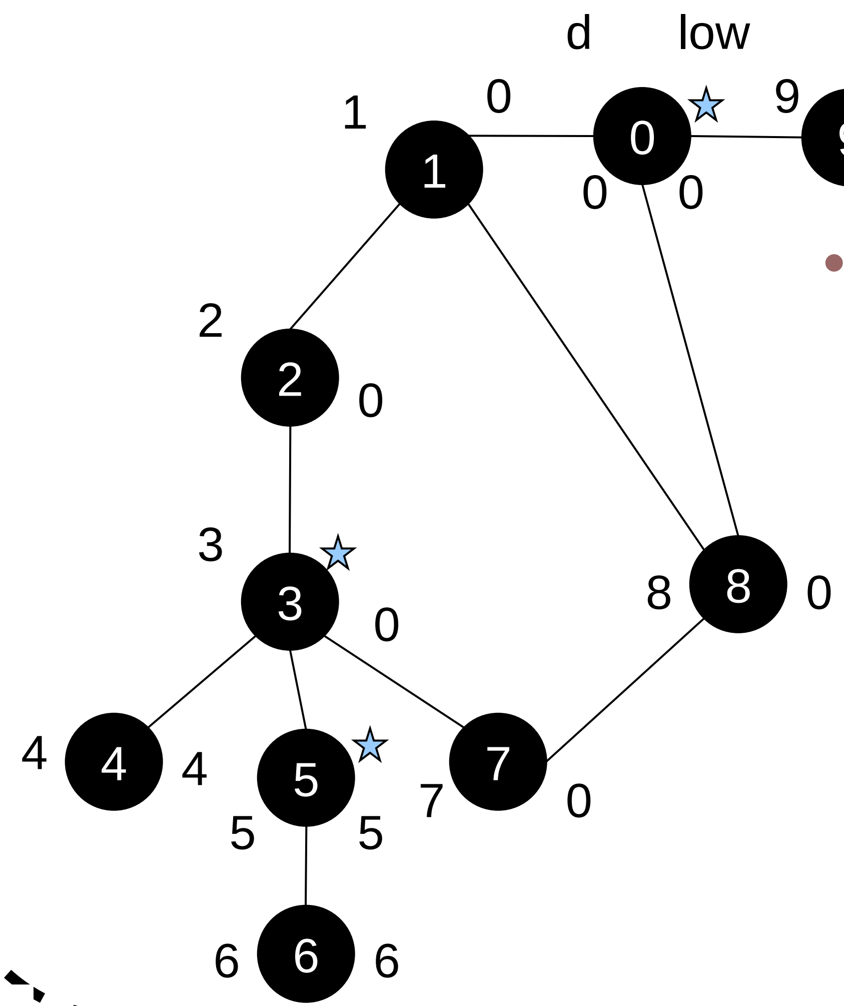
Exemplu rulare (28)



low(0) = min(low(0), low(9)) = 0
P(0) = null → revenire

- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v ∈ succs(u)
    - **Dacă** (c(v) e alb)
      - p(v) = u; subarb(u)++; // actualizare nr // subarbori dominați de u
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (p(u) != null && low(v) ≥ d(u)) art(u) = 1; // cazul 2 al teoremei
    - **Altfel** low(u) = min{low(u), d(v)} // actualizare low

Exemplu rulare (29)



Subarb(0) = 2 > 1 → art(0) = 1

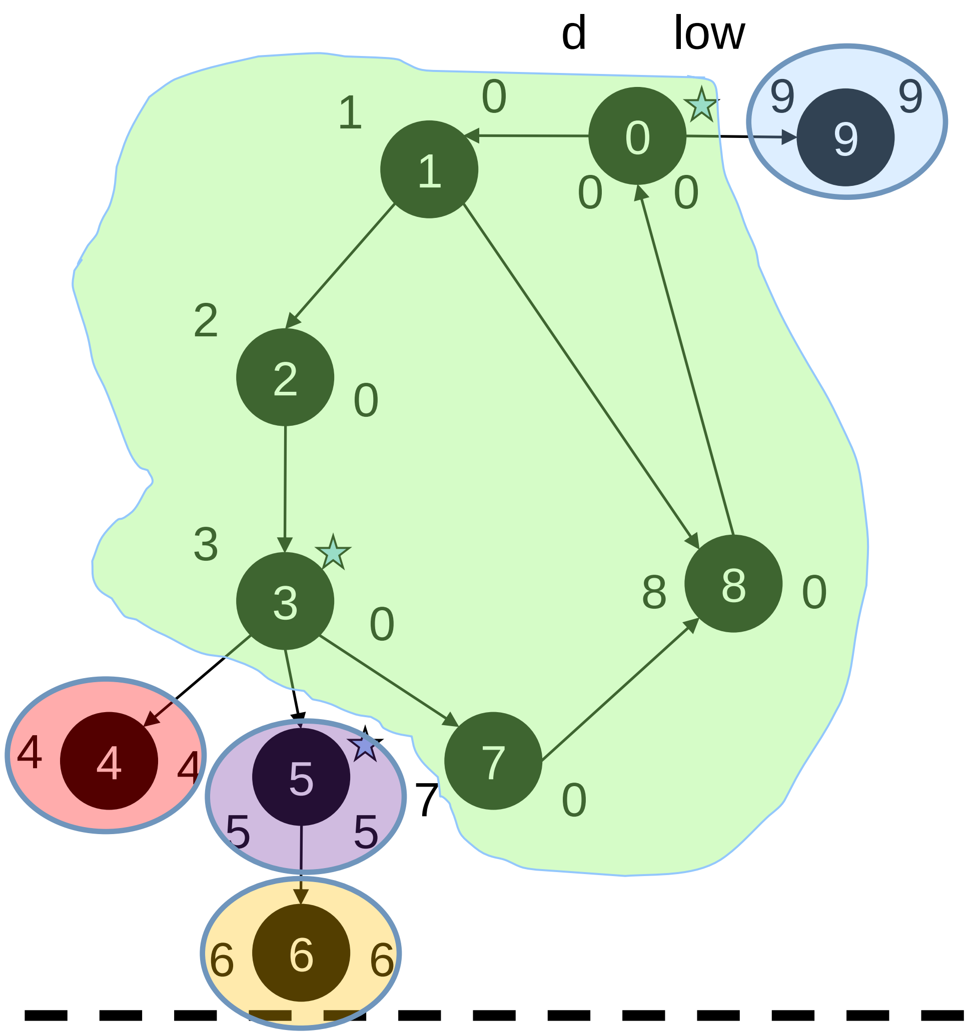
- Articulații (G)
  - V = noduri(G) // inițializări
  - Timp = 0;
  - **Pentru fiecare** (u ∊ V)
    - c(u) = alb;
    - d(u) = 0;
    - p(u) = null;
    - low(u) = 0;
    - subarb(u) = 0; // reține numărul de subarbori
      // dominați de u
    - art(u) = 0; // reține punctele de articulație
  - **Pentru fiecare** (u∊V)
    - **Dacă** c(u) e alb
      - Exploreaza(u);
      - **Dacă** (subarb(u) > 1) // cazul în care u este
        // rădăcina în arborele
        art(u) = 1 // DFS și are mai mulți subarbori
        // → cazul 1 al teoremei

Algoritmul lui Tarjan adaptat pentru determinarea CTC

- index = 0 // nivelul pe care este nodul în arborele DFS
- S = empty // se folosește o stivă care se inițializează cu Ø
- **Pentru fiecare** v din V
  - **Dacă** (v.index e nedefinit) **atunci** // se pornește DFS din fiecare nod pe care
    - Tarjan(v) // nu l-am vizitat încă

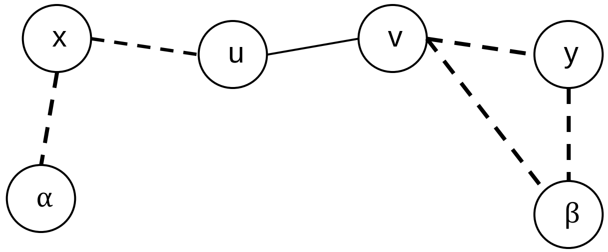
- Tarjan(v)
  - v.index = index // se setează nivelul nodului v
  - v.lowlink = index // reține strămoșul nodului v
  - index = index + 1 // incrementez nivelul
  - S.push(v) // introduc v în stivă
  - **Pentru fiecare** (v, v') din E // se prelucrează succesorii lui v
    - **Dacă** (v'.index e nedefinit **sau** v' e în S) **atunci** // CTC deja identificate sunt ignorate
      - **Dacă** (v'.index e nedefinit) **atunci** tarjan(v') // dacă nu a fost vizitat v' intru în recursivitate
      - v.lowlink = min(v.lowlink, v'.lowlink) //actualizez strămoșul
  - **Dacă** (v.lowlink == v.index) **atunci** // printez CTC începând de la coadă spre rădăcină
    - print “CTC:"
    - **Repetă**
      - v' = S.pop // extrag nodul din stiva și îl printez
      - print v'
    - **Până când** (v' == v) // până când extrag rădăcina

Exemplu rulare (CTC)

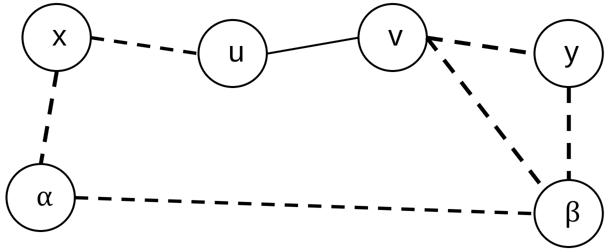


Punți

- Definiție: G = (V,E), graf neorientat și (u,v) ∊ E. (u,v) este punte în G ⇔ ∃ x,y ∊ V, x ≠ y, a.î. ∀ x..y conține muchia (u,v).



Orice drum x..y trece prin (u,v) =>(u,v) este punte



(u,v) nu este punte

Algoritm punți (I)

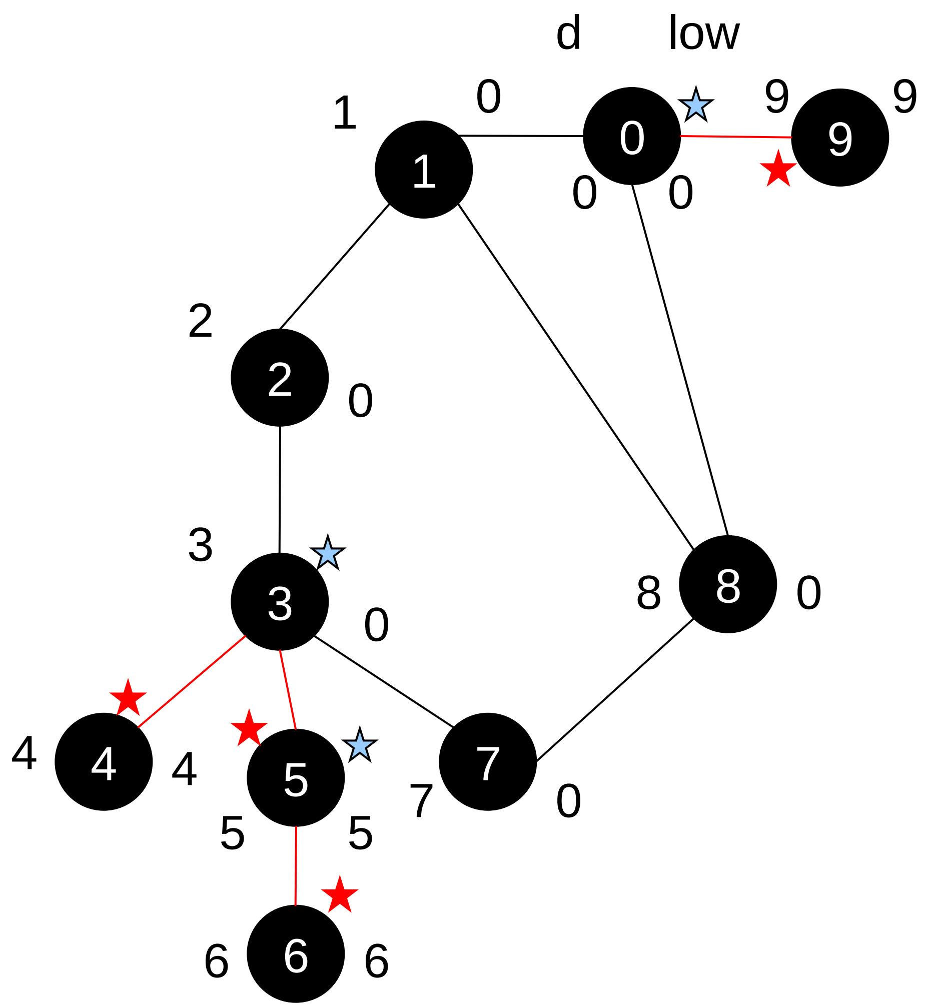
- Punți(G)
  - V = noduri(G) // inițializări
  - Timp = 0;
  - **Pentru fiecare** nod u (u ∊ V)
    - c(u) = alb;
    - d(u) = 0;
    - p(u) = null;
    - low(u) = 0;
    - punte(u) =0; // înlocuiește: subarb(u) = 0; art(u) = 0;
  - **Pentru fiecare** nod u (u ∊ V)
    - **Dacă** c(u) e alb
      - Explorează(u)

Algoritm punți (II)

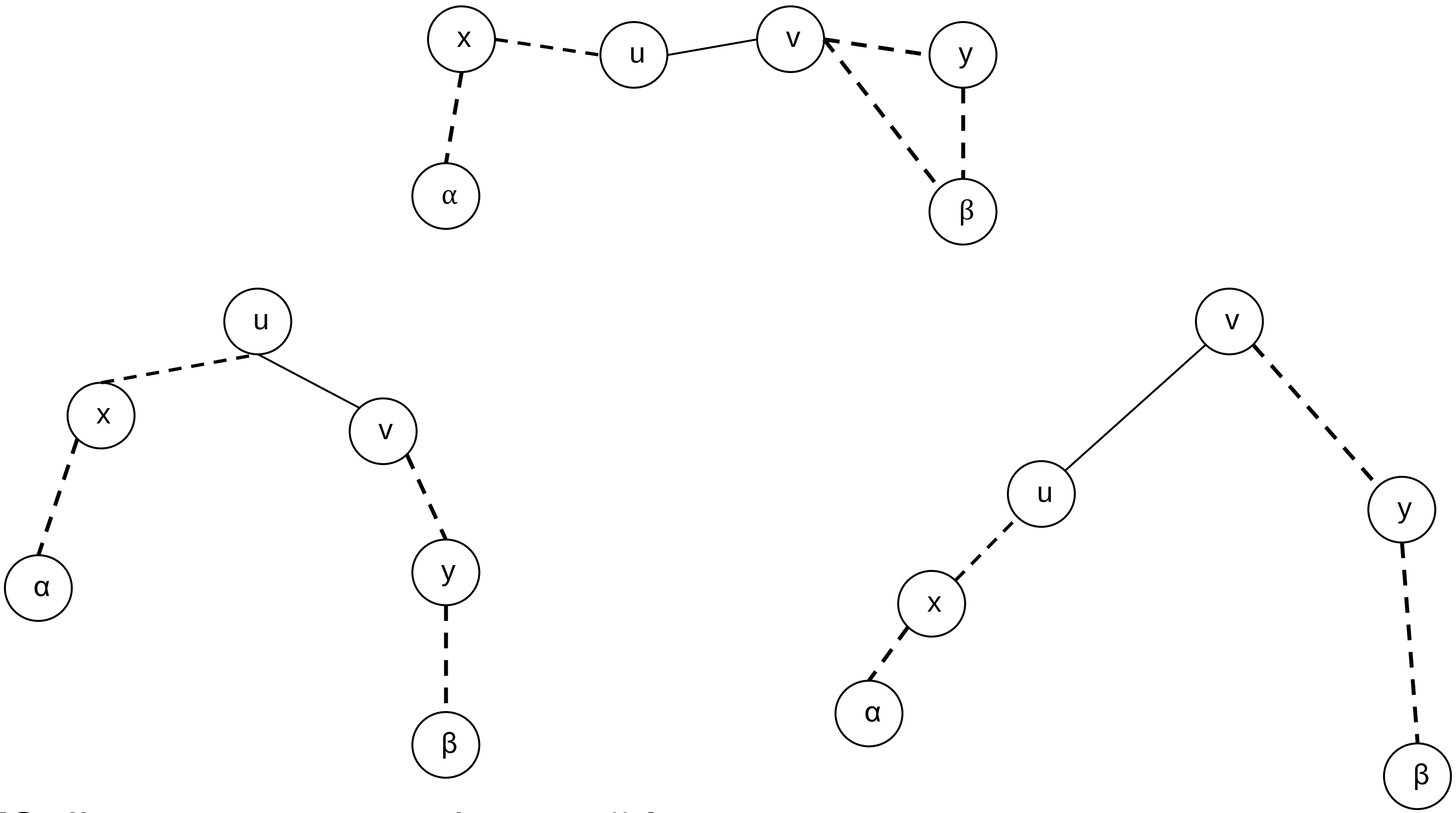
- Explorează(u)
  - d(u) = low(u) = timp++; // inițializări
  - c(u) = gri;
  - **Pentru fiecare** nod v (v ∈ succs(u))
    - **Dacă** c(v) e alb
      - p(v) = u; // se elimină: subarb(u)++;
      - Explorează(v);
      - low(u) = min{low(u), low(v)} // actualizare low
      - **Dacă** (low(v) > d(u)) punte(v) = 1;

      // în loc de: **Dacă**(p(u) != null && low(v) >= d(u))
    - **Altfel**
      - **Dacă** (p(u) ≠ v) low(u) = min{low(u), d(v)} // actualizare low

Exemplu rulare (Punți)



Exemplu



DFS din u; puntea este detectată în v

DFS din v; puntea este detectată în u

Drumuri de cost minim

- G = (V,E) un graf, iar w:E → $\Re$ o funcție de cost asociată arcelor grafului (w(u,v) = costul arcului (u,v)).
- Cost(u..v) = costul drumului u..v (este aditiv – costul drumului = suma costurilor arcelor).
- Variante:
  1. Drumuri punct – multipunct: pentru un nod dat s ∈ V, să se găsească un drum de cost minim de la s la ∀u ∈ V; Dijkstra, Bellman-Ford
  2. Drumuri multipunct – punct: pentru un nod dat e ∈ V, să se găsească un drum de cost minim de la ∀u ∈ V la e; G^T și apoi 1
  3. Drumuri punct – punct: pentru două noduri date u și v ∈ V, să se găsească un drum u..v de cost minim; Folosind 1
  4. Drumuri multipunct – multipunct: ∀u, v ∈ V, să se găsească un drum u..v de cost minim. Floyd-Warshall
  5. Drumuri de cost maxim!

Temă de gândire pentru acasă – posibil subiect de examen!

Optimalitatea drumurilor minime (I)

- Lemă 25.1 (Subdrumurile unui drum minim sunt drumuri optimale): $G = (V,E)$, $w : E \to \Re$ funcție de cost asociată. Fie $p = v_1v_2...v_k$ un drum optim de la v_1 la v_k. Atunci pentru orice i și j cu $1 \leq i \leq j \leq k$, subdrumul lui p de la v_i la v_j este un drum minim.

- Dem: Fie $p_{ij} = v_i..v_j$ subdrumul din p dintre v_i și v_j. $\to$ $p = v_1..v_i..v_j..v_k$ => cost (p) = cost ($v_1..v_i$) + cost ($v_i..v_j$) + cost ($v_j..v_k$).
- Pp. prin absurd că $v_i..v_j$ nu e optim => $\exists$p' a.î. cost (p') < cost ($v_i..v_j$) => p nu e drum minim $\to$ Contrazice ipoteza $\to$ p_{ij} este drum minim.

Optimalitatea drumurilor minime (II)

- Corolar 25.2: G = (V,E), w : E → $\mathfrak{R}$ funcție de cost asociată. Fie p = s..uv un drum optim de la s la v. Atunci costul optim al acestui drum poate fi scris ca $\delta(s,v) = \delta(s,u) + w(u,v)$.

- Dem: Conform teoremei anterioare, s..u e un drum optim => cost (s..u) = $\delta(s,u)$.

- Lemă 25.3: G = (V,E), w : E → $\mathfrak{R}$ funcție de cost asociată. $\forall$ (u,v) $\in$ E avem $\delta(s,v) \leq \delta(s,u) + w(u,v)$.

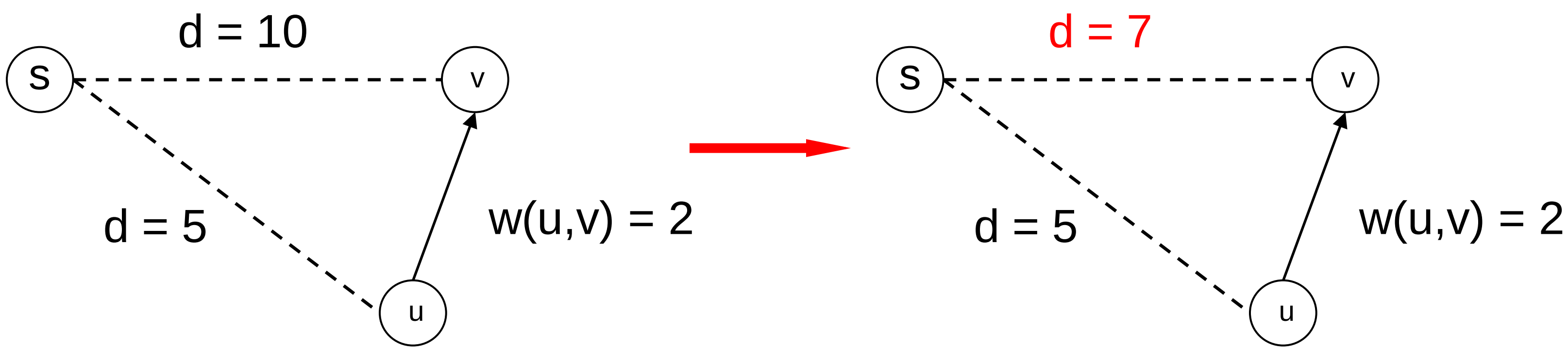
- Dem: Orice drum optim are costul mai mic ca al oricărui alt drum.

Drumuri minime de sursă unică

- Sunt concepuți pentru grafuri orientate.
- Bazați pe algoritmi Greedy.
- Se pornește de la nodul de start și pe baza unui optim local, drumurile sunt extinse și optimizate până la soluția finală.
- Notații:
  - d[v] = costul drumului descoperit s..v;
  - $\delta(u,v)$ = costul drumului optim u..v; $\delta(u,v)=\infty$ daca $v \notin R(u)$;
  - p(v) = predecesorul lui v pe drumul s..v.

Drumuri minime de sursă unică

- Relaxarea arcelor → dacă d[v] > d[u] + w(u,v), atunci actualizează d[v].



- Exemple: Dijkstra și Bellman–Ford.

Relaxarea arcelor (I)

- Lemă 25.5: G = (V,E), w : E → $\Re$ funcție de cost asociată. ∀ v ∈ V, d[v] obținut de algoritmul lui Dijkstra respectă d[v] ≥ δ(s,v). În plus, odată atinsă valoarea δ(s,v), ea nu se mai modifică.

- Dem:
  - ∀ v ∈ V, v ∉ R(s) → d[v] = δ(s,v) = ∞; d[s] = δ(s,s) = 0 (inițializare)
  - Pt v ∈ R(s), inițializare → d[v] = ∞ ≥ δ(s,v). Dem. prin reducere la absurd că după oricâte relaxări, relația se menține. Fie v primul vârf pentru care relaxarea (u,v) determină d[v] < δ(s,v) → după relaxarea (u,v): d[u] + w(u,v) = d[v] < δ(s,v) ≤ δ(s,u) + w(u,v) → d[u] < δ(s,u). Dar relaxarea nu modifică d[u], iar v e primul pentru care d[v] < δ(s,v). Contrazice presupunerea! => d[v] ≥ δ(s,v), ∀ v ∈ V
  - Cum d[v] ≥ δ(s,v) => odată ajuns la d[v] = δ(s,v), ea nu mai scade. Cum relaxarea nu creste valorile => d[v] nu se mai modifică.

Relaxarea arcelor (II)

- Lemă 25.7: $G = (V,E)$, $w : E \rightarrow \Re$ funcție de cost asociată. Fie $p = s..uv$ un drum optim de la s la v. Dacă $d[u] = \delta(s,u)$ la un moment dat, atunci începând cu momentul imediat următor relaxării arcului (u,v) avem $d[v] = \delta(s,v)$

- Dem:
  - Dacă înainte de relaxare $d[v] > d[u] + w(u,v)$, prin relaxare $\rightarrow$ $d[v] = d[u] + w(u,v)$. Altfel, $d[v] \leq d[u] + w(u,v)$ => după relaxare avem $d[v] \leq d[u] + w(u,v)$.
  - Cum $d[u] = \delta(s,u)$ și relaxarea (u,v) nu modifică $d[u]$ => $d[v] \leq d[u] + w(u,v) = \delta(s,u) + w(u,v) = \delta(s,v)$ (conf. Corolar 25.2) $\rightarrow$ $d[v] = \delta(s,v)$

Algoritmul lui Dijkstra (I)

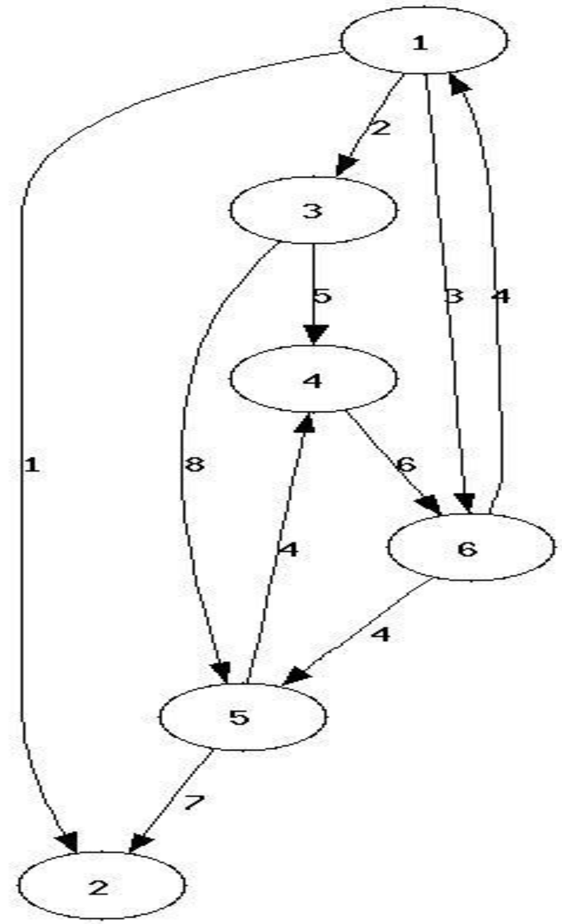
- Folosește o coadă de priorități în care se adaugă nodurile în funcție de distanța cunoscută în momentul respectiv de la s până la nod.
- Se folosește **NUMAI** pentru costuri pozitive (w(u,v) > 0, ∀u,v∈V).
- Dijkstra_generic (G,s)
  - V = nodurile lui G
  - **Cât timp** (V != ∅)
    - u = nod din V cu d[u] min
    - V = V - {u}
    - **Pentru fiecare** (v ∈ succesorii lui u) relaxare_arc(u,v)

    // optimizare drum s..v pentru v ∈ succesorilor lui u

Algoritmul lui Dijkstra (II)

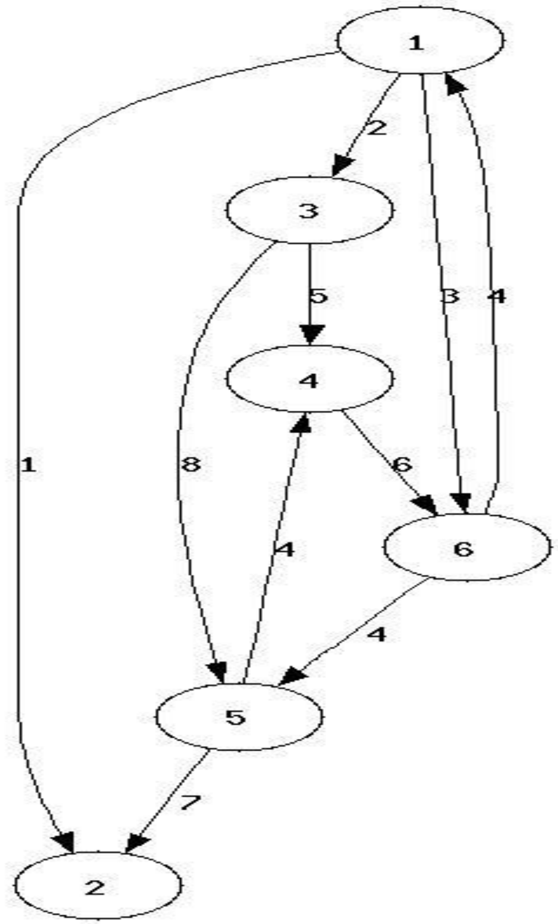
- Dijkstra(G,s)
  - **Pentru fiecare** (u ∊ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] minim
    - // Q = Q - {u} – se execută în cadrul lui ExtrageMin
    - **Pentru fiecare** (v ∊ Q și v din succesorii lui u)
      - **Dacă** (d[v] > d[u] + w(u,v))
        - d[v] = d[u] + w(u,v) // actualizez distanța
        - p[v] = u // și părintele

Exemplu (I)



Exemplu (II)

- d[1] = 0;
- (1): d[2] = 1; d[3] = 2; d[6] = 3;
- (2): d[4] = 7; d[5] = 10;
- (3): d[5] = 7;



- Dijkstra(G,s)
  - **Pentru fiecare** (u ∊ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] // minim
    - // Q = Q - {u} – se execută în cadrul lui ExtrageMin
    - **Pentru fiecare** (v ∊ Q și v din succesorii lui u)
      - **Dacă** (d[v] > d[u] + w(u,v))
        - d[v] = d[u] + w(u,v) // actualizez distanța
        - p[v] = u // și părintele

Complexitate Dijkstra

- Depinde de ExtrageMin – coadă cu priorități.
- Operații ce trebuie realizate pe coadă + frecvența lor:
  - insert – V;
  - delete – V;
  - conține? – V;
  - micșorează_val – E;
  - este_vidă? – V.

- Dijkstra(G,s)
  - **Pentru fiecare** (u ∈ V)
    - d[u] = ∞; p[u] = null;
  - d[s] = 0;
  - Q = construiește_coada(V) // coadă cu priorități
  - **Cât timp** (Q != ∅)
    - u = ExtrageMin(Q); // extrage din V elementul cu d[u] minim
    - // Q = Q - {u} – se execută in cadrul lui ExtrageMin
    - **Pentru fiecare** (v ∈ Q si v din succesorii lui u)
      - **Dacă** (d[v] > d[u] + w(u,v))
        - d[v] = d[u] + w(u,v) // actualizez distanța
        - p[v] = u // si părintele

Implementare cu vectori

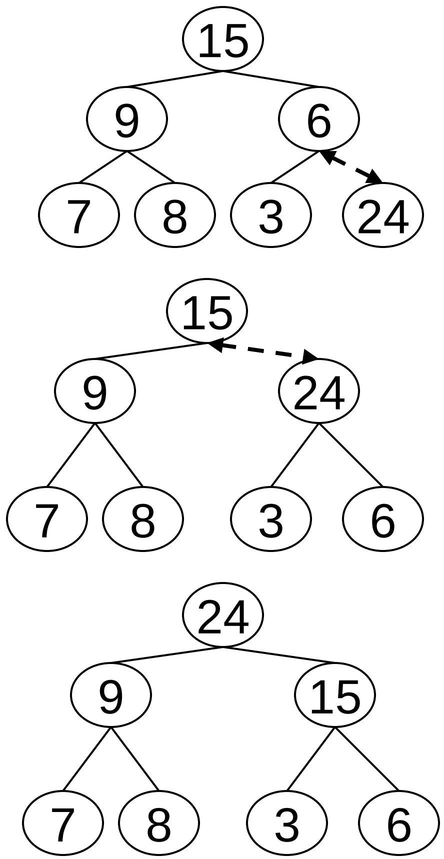
- Costuri:
  - insert – 1 * V = V;
  - delete – V * V = V^2 (necesită căutarea minimului);
  - conține? – 1 * V = V;
  - micșorează_val – 1 * E = E;
  - este_vidă? – 1 * V = V;
- Cea mai bună metodă pentru grafuri “dese” ($E \approx V^2$)!

Implementare cu heap binar

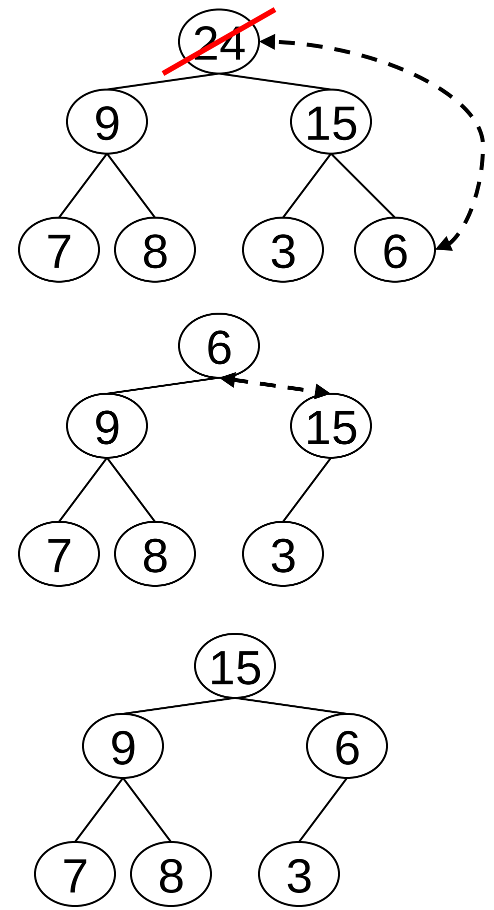
- Heap binar – structură de date de tip arbore binar + 2 constrângeri:
  - Fiecare nivel este complet; ultimul se umple de la stânga la dreapta;
  - $\forall u \in$ Heap; $u \geq$ răd(st(u)) și $u \geq$ răd(dr(u)) unde $\geq$ este o relație de ordine pe mulțimea pe care sunt definite elementele heapului.

Operatii pe Heap Binar

insert



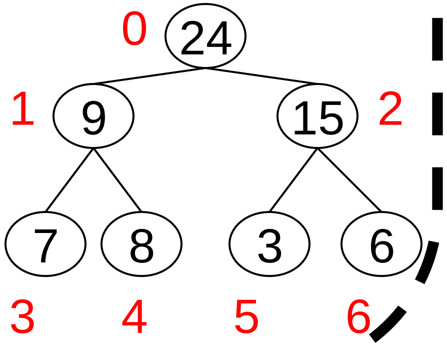
delete



Implementare Heap Binar

- Implementare folosind vectori.
- Poziție[i] = unde se găsește în indexul de valori elementul de pe poziția i din heap.
- Reverse[i] = unde se găsește în heap elementul de pe poziția i din valoare.
- Implementare disponibila la [3].

Index	0	1	2	3	4	5	6
Valoare	7	6	15	8	24	9	3
Poziție	4	5	2	0	3	6	1
Reverse	3	6	2	4	0	1	5



Heap Binar

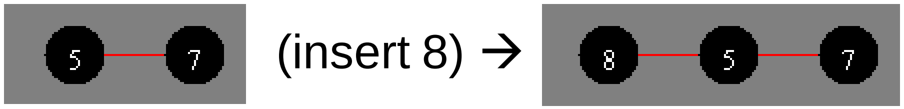
- Costuri:
  - insert – logV * V = VlogV;
  - delete – logV * V = VlogV;
  - conține? – 1 * V = V;
  - micșorează_val – logV * E = ElogV;
  - este_vidă? – 1 * V = V.
- Eficient dacă graful are arce puține comparativ cu numărul de noduri.

Heap Fibonacci

- Poate fi format din mai mulți arbori.
- Cheia unui părinte ≤ cheia oricărui copil.
- Fiind dat un nod u și un heap H:
  - p(u) – părintele lui u;
  - copil(u) – legătura către unul din copiii lui u;
  - st(u), dr(u) – legătura la frații din stânga și din dreapta (cei de pe primul nivel sunt legați între ei astfel);
  - grad(u) – numărul de copii ai lui u;
  - min(H) – cel mai mic nod din H;
  - n(H) – numărul de noduri din H.

Operatii Heap Fibonacci

- Inserare nod – O(1)
  - construiește un nou arbore cu un singur nod



- Min – accesibil direct - min(H) – O(1)

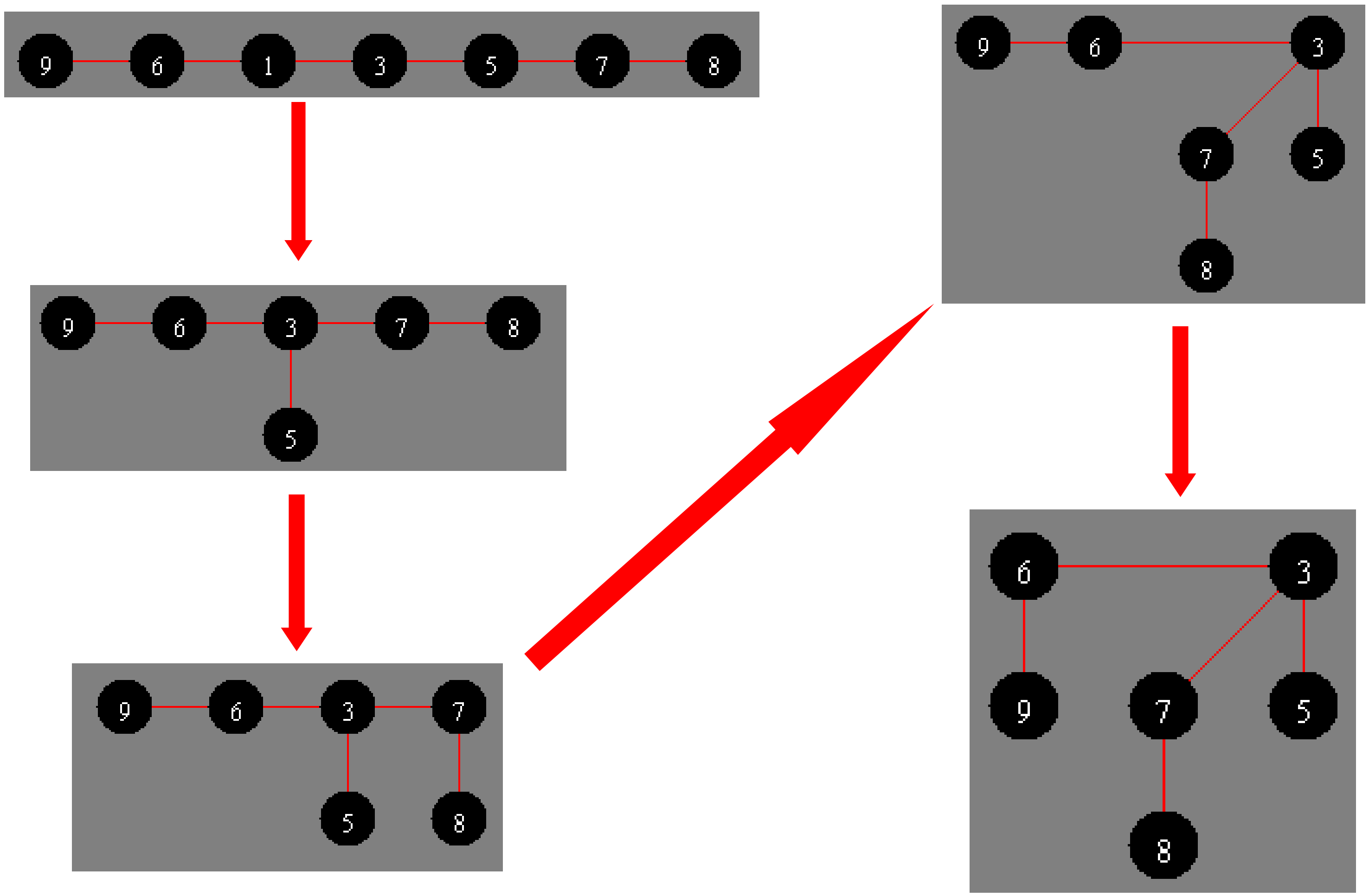
- ExtrageMin O(logn) – cost amortizat!
  - Mută copiii minimului pe prima coloană;
  - **Consolidează** heap-ul.

Operatii Heap Fibonacci

- Consolidare Heap
  - Cât timp există 2 arbori cu grade egale Arb(x) și Arb(y), x < y:
    - Arb(y) adăugat ca și copil al lui x;
    - grad[x] ++;

- Applet și implementare disponibile la [4].

Consolidare Heap



Costuri Heap Fibonacci

- Costuri:
  - insert – 1 * V = V;
  - delete – logV * V = VlogV(amortizat!);
  - micșorează_val – 1 * E = E;
  - este_vidă? – 1 * V = V.
- Cea mai rapidă structură dpdv teoretic.

Concluzii Dijkstra

- Implementarea trebuie realizată în funcție de tipul grafului pe care lucrăm:
  - vectori pentru grafuri “dese”;
  - heap pentru grafuri “rare”.

- Heapul Fibonacci este mai eficient decât heapul binar dar mai dificil de implementat.

ÎNTREBĂRI?