# Grafuri

## Conectivitate. Arbori. Căutare. Grafuri bipartite. Arbori de acoperire

Mircea Marin

29 noiembrie 2019

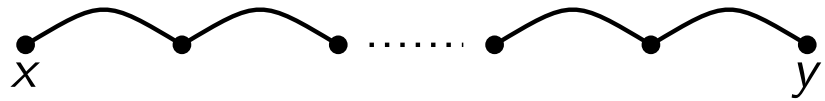
# Cuprins

- Conectivitate
- Arbori de acoperire
- Căutare
  - algoritmi de parcurgere în lățime și adâncime
    - aplicații: sortare topologică, determinarea componentelor conexe, etc.
  - căutarea în grafuri bipartite
    - algoritmul lui Dijkstra
- Grafuri bipartite și cuplaje
- Arbori minimi de acoperire
  - algoritmul lui Kruskal

# Conectivitate

Fie $G = (V, E)$ un graf și $x, y \in V$. $x$ este conectat cu $y$ dacă

cazul 1: $G$ este neorientat și conține un lanț (engl. *chain*) de la $x$ la $y$:

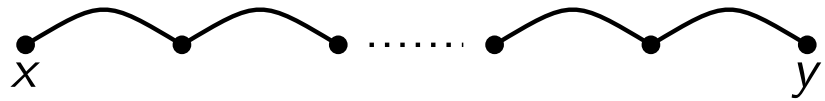



cazul 2: $G$ este orientat și conține un drum (engl. *walk*) de la $x$ la $y$:

# Conectivitate

Fie $G = (V, E)$ un graf și $x, y \in V$. $x$ este conectat cu $y$ dacă

cazul 1: $G$ este neorientat și conține un lanț (engl. *chain*) de la $x$ la $y$:



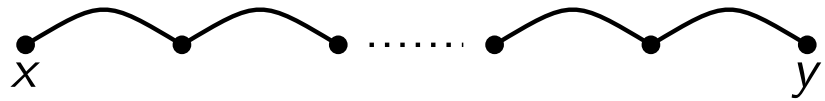

cazul 2: $G$ este orientat și conține un drum (engl. *walk*) de la $x$ la $y$:


$G$ este conex dacă orice două noduri sunt conectate.

# Conectivitate

Fie $G = (V, E)$ un graf și $x, y \in V$. $x$ este conectat cu $y$ dacă

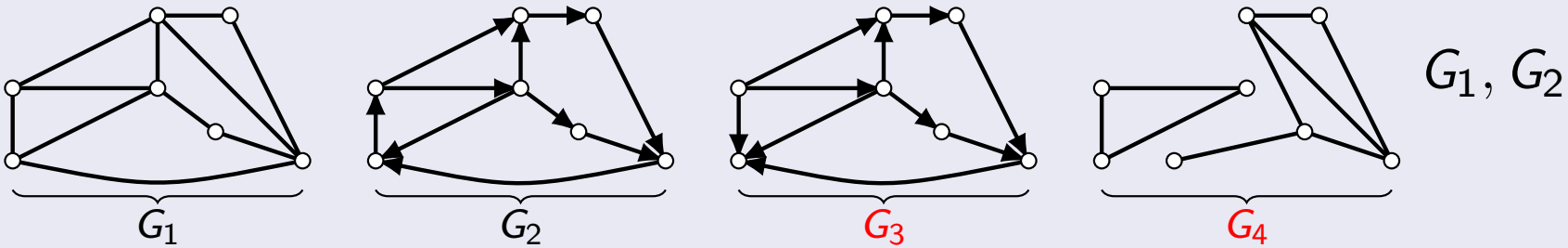
cazul 1: $G$ este neorientat și conține un lanț (engl. *chain*) de la $x$ la $y$:




cazul 2: $G$ este orientat și conține un drum (engl. *walk*) de la $x$ la $y$:


$G$ este conex dacă orice două noduri sunt conectate.

**Exemplu: grafuri conexe și grafuri neconexe**

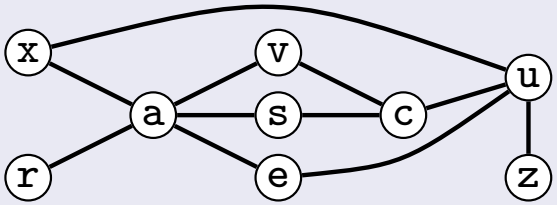



$G_1$, $G_2$ sunt conexe. $G_3$, $G_4$ nu sunt conexe.

Fie $G = (V, E)$ un graf și $x, y \in V$ a.î. $x$ este conectat cu $y$.

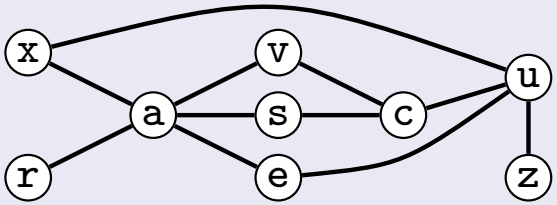
- Distanța $d(x, y)$ de la $x$ la $y$ este
  1. lungimea celui mai scurt lanț de la $x$ la $y$, dacă $G$ este neorientat, sau
  2. lungimea celui mai scurt drum de la $x$ la $y$ dacă $G$ este orientat.
- Dacă $G$ este conex, atunci:
  - Excentricitatea lui $x$ în $G$ este **distanța cea mai mare** de la $x$ la un nod în $G$: $e(x) := \max\{d(x, y) \mid y \in V\}$
  - Centrul lui $G$ este mulțimea de noduri cu excentricitate minimă: $u \in V$ este în centru dacă $e(u) = \min\{e(x) \mid x \in V\}$.
  - Raza lui $G$ este excentricitatea unui nod din centrul lui $G$.
  - Diametrul lui $G$ este excentricitatea maximă a unui nod din $G$: $diam(G) := \max\{e(x) \mid x \in V\}$

## Exemplu



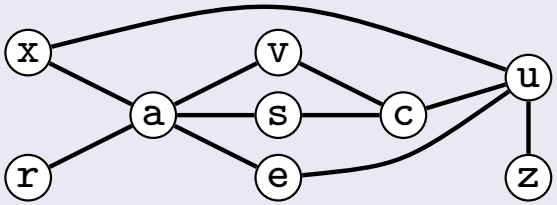

- Distanța de la x la c este
  Distanța de la r la z este
- Excentricitatea lui x este
  Excentricitatea lui z este
- Centrul grafului este
- Raza grafului este
- Diametrul grafului este

## Exemplu



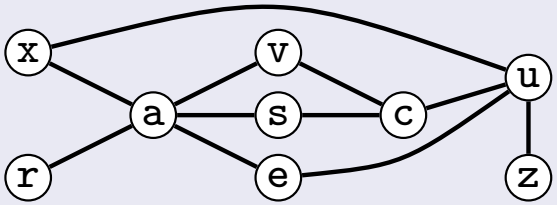

- Distanța de la x la c este 2.
  Distanța de la r la z este
- Excentricitatea lui x este
  Excentricitatea lui z este
- Centrul grafului este
- Raza grafului este
- Diametrul grafului este

## Exemplu



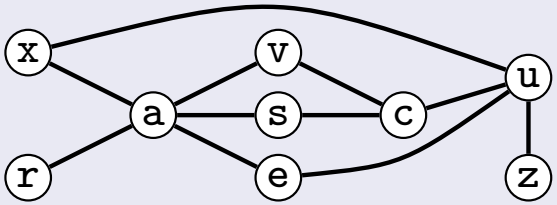

- Distanța de la x la c este 2.
  Distanța de la r la z este 4.
- Excentricitatea lui x este
  Excentricitatea lui z este
- Centrul grafului este
- Raza grafului este
- Diametrul grafului este

## Exemplu



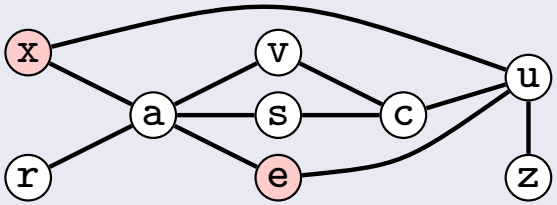

- Distanța de la x la c este 2.
  Distanța de la r la z este 4.
- Excentricitatea lui x este 2.
  Excentricitatea lui z este
- Centrul grafului este
- Raza grafului este
- Diametrul grafului este

## Exemplu



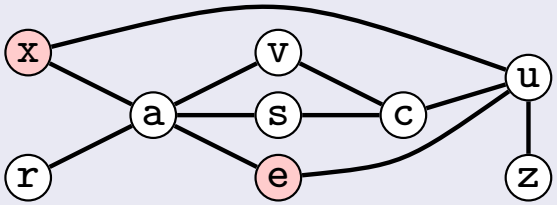

- Distanța de la x la c este 2.
  Distanța de la r la z este 4.
- Excentricitatea lui x este 2.
  Excentricitatea lui z este 4.
- Centrul grafului este
- Raza grafului este
- Diametrul grafului este

## Exemplu



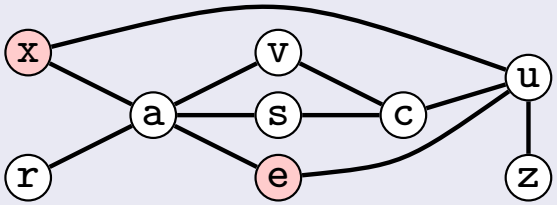

- Distanța de la x la c este 2.
  Distanța de la r la z este 4.
- Excentricitatea lui x este 2.
  Excentricitatea lui z este 4.
- Centrul grafului este $\{x, e\}$
- Raza grafului este
- Diametrul grafului este

## Exemplu




- Distanța de la x la c este 2.
  Distanța de la r la z este 4.
- Excentricitatea lui x este 2.
  Excentricitatea lui z este 4.
- Centrul grafului este $\{x, e\}$
- Raza grafului este 2.
- Diametrul grafului este

## Exemplu



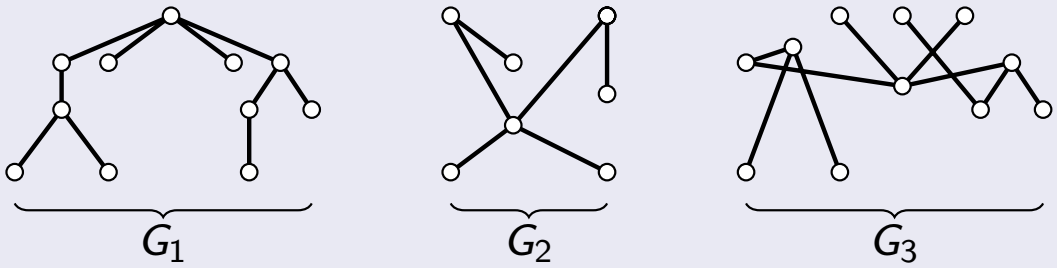

- Distanța de la x la c este 2.
  Distanța de la r la z este 4.
- Excentricitatea lui x este 2.
  Excentricitatea lui z este 4.
- Centrul grafului este $\{x, e\}$
- Raza grafului este 2.
- Diametrul grafului este 4.

# Arbori

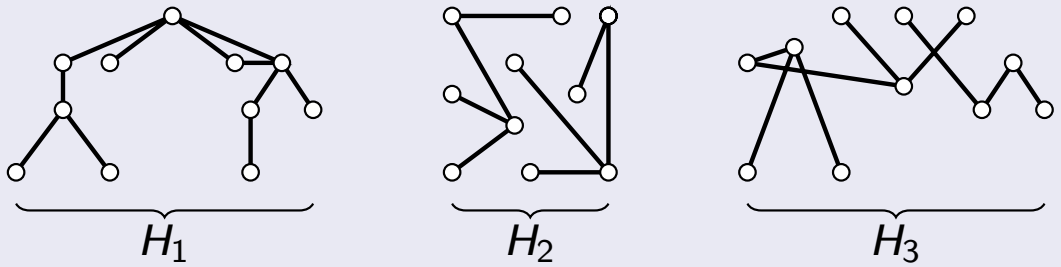
Arbore = graf simplu conectat și fără cicluri.

## Exemple

- Sunt arbori:




- Nu sunt arbori:

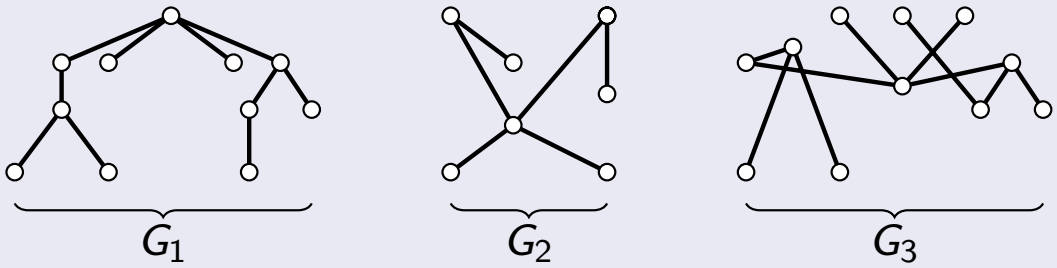


# Arbori

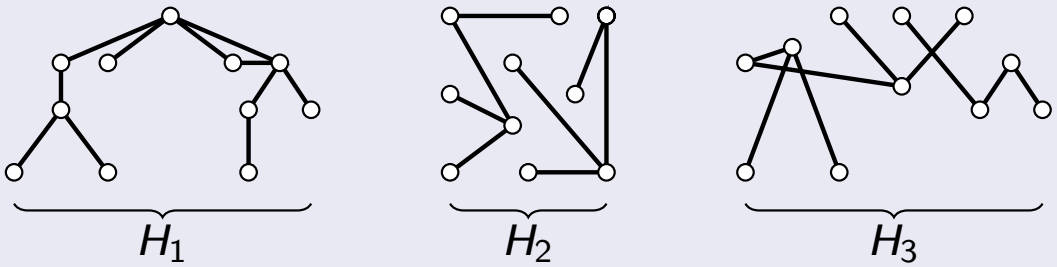
Arbore = graf simplu conectat și fără cicluri.

## Exemple

- Sunt arbori:

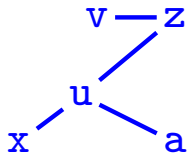



- Nu sunt arbori:

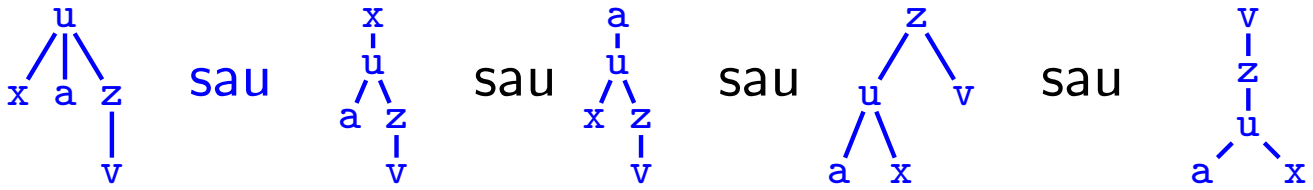



REMARCĂ: Orice arbore poate fi redesenat în mod uzual: cu un nod rădăcină $r$ conectat cu vecinii lui desenați sub $r$, și

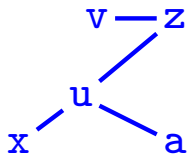
- orice nod $x \neq r$ are un vecin deasupra lui (părintele lui $x$)
- toți ceilalți vecini sunt sub $x$ (copiii lui $x$)

- Exemple de redesenări uzuale ale arborelui 

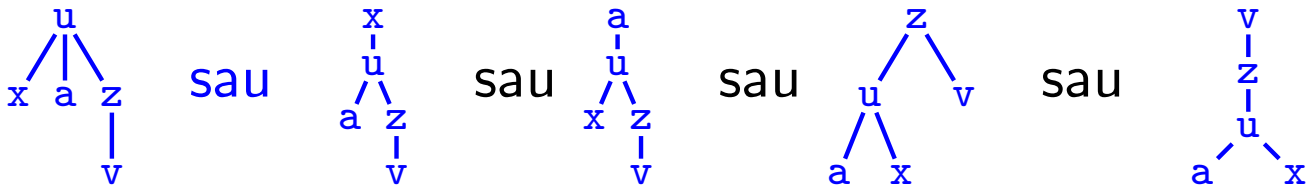
:



# Arbori și păduri

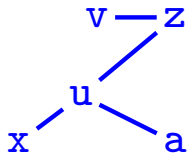
- Exemple de redesenări uzuale ale arborelui 

:

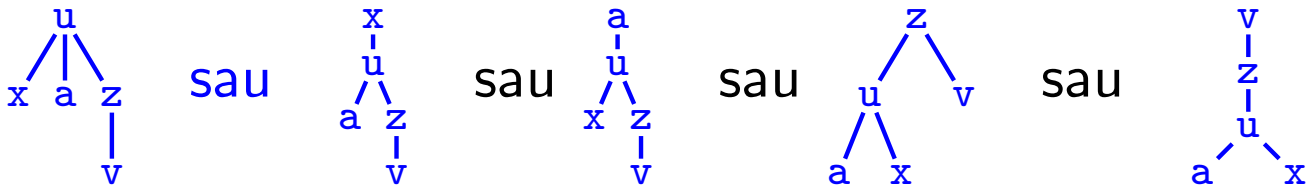



- O pădure este un arbore ale cărui componente sunt arbori.

# Arbori și păduri

- Exemple de redesenări uzuale ale arborelui 

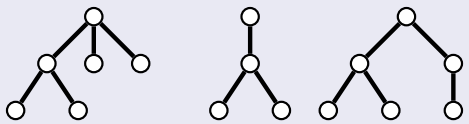
:




- O pădure este un arbore ale cărui componente sunt arbori.

## Observații

1. Un graf simplu este pădure dacă și numai dacă nu are cicluri.
2. Orice arbore cu $n$ noduri are $n-1$ muchii.
3. Orice pădure cu $n$ noduri și $k$ arbori are $n-k$ muchii. De exemplu, pădurea



are 16 noduri, 3 arbori, și 16-3=13 muchii.

# Arbori de acoperire

Fie $G = (V, E)$ un graf simplu conex. $T$ este un arbore de acoperire al lui $G$ dacă:

1. $T$ este arbore
2. $T$ se obține din $G$ ștergând doar muchii.

## Observații

1. Orice arbore de acoperire al unui graf simplu conex $G$ se poate obține repetând pasul următor până când $G$ nu mai are cicluri:
   - Se caută un ciclu $C$ în $G$ și se șterge o muchie a lui $C$ din $G$
2. Dacă $G$ este graf simplu (posibil neconex), putem obține o pădure de acoperire a lui $G$ repetând pasul următor până când $G$ nu mai are cicluri:
   - Se caută un ciclu $C$ în $G$ și se șterge o muchie a lui $C$ din $G$

Vom considera 2 tipuri de grafuri simple $G = (V, E)$:

Grafuri neorientate: nu conțin bucle, iar între două noduri există cel mult o muchie.

Grafuri orientate: nu conțin bucle, iar de la un nod la altul există cel mult o muchie.

- Spunem că se poate ajunge de la un nod $s$ la un nod $x$ în $G$ dacă există un lanț (dacă $G$ este neorientat) sau un drum (dacă $G$ este orientat) de la $s$ la $x$.
- Descoperirea tuturor nodurilor la care se poate ajunge de la un nod sursă $s$ se poate face cu un algoritm de căutare.
- Cei mai cunoscuți algoritmi de căutare sunt:
  1. căutarea în lățime (BFS, breadth-first search)
  2. căutarea în adâncime (DFS, depth-first search)

Explorează sistematic muchiile lui $G$ pt. a găsi toate nodurile la care se poate ajunge de la un nod sursă $s$.

## Caracteristici ale căutării în lățime

- Pentru fiecare nod $x$ la care se poate ajunge din $s$, calculează distanța $d[x]$ (cel mai mic nr. de muchii) de la $s$ la $x$.
- Construiește un arbore $G_\pi$ cu rădăcina $s$ cu următoarele proprietăți:
  1. conține toate nodurile la care se poate ajunge din $s$.
  2. pentru orice nod $x$ din $G_\pi$, ramura lui $G_\pi$ de la $s$ la $x$ este una din cele mai scurte căi de la $s$ la $x$ în $G$.
- Algoritmul este incremental: pentru fiecare distanță posibilă $k > 0$, alg. descoperă mai întâi toate nodurile la distanță $k$ de $s$ înainte de a descoperi nodurile la distanță $k + 1$ de $s$.

# Căutarea în lățime (BFS)

## Cum se calculează arborele $G_\pi$ și distanțele $d[x]$?

Presupunem că $G = (V, E)$ este reprezentat cu liste de adiacență.

- Fiecărui nod $x$ al lui $G$ i se atribuie (1) o culoare $c[x]$ care este alb (a), gri (g) sau negru (n) și (2) un predecesor $\pi[x]$ în arborele $G_\pi$. Inițial
  - $c[x] =$ a pentru toți $x \in V - \{s\}$, $c[s] =$ g.
  - $d[x] = \infty$ pentru toți $x \in V - \{s\}$, $d[s] = 0$.
  - $\pi[x] =$NIL pentru toți $x \in V$.
- Un nod devine descoperit când căutarea ajunge prima oară la el.
- Un nod $x$ se colorează gri atunci când devine descoperit, și negru imediat după ce toți vecinii lui devin descoperiți.
- Lista nodurilor gri existente la un moment dat se reține în o coadă $Q$ *first-in, first-out* (FIFO), în ordinea descoperirii lor.
  - Inițial, $Q =$ [s].
- Pt. fiecare nod gri $y \in Q$, se scanează nodurile adiacente la $y$. Pentru fiecare vecin alb $x$ al lui $y$:
  - $x$ se colorează gri (pt. că devine descoperit), se adaugă la $Q$, și $d[x] := d[y] + 1$, $\pi[x] := y$

Apoi, $y$ se colorează negru.

# Căutarea în lățime (BFS)

## Pseudocod

BFS($G, s$)
1. **for** fiecare nod $u \in V - \{s\}$ **do**
2. $\quad c[u] := \mathtt{w}$
3. $\quad d[u] := \infty$
4. $\quad \pi[u] :=$NIL
5. $c[u] := \mathtt{g}$
6. $d[s] := 0$
7. $\pi[s] :=$NIL
8. $Q := [s]$
9. **while** $Q \neq []$ **do**
10. $\quad u :=$Dequeue($Q$)
11. $\quad$ **for** fiecare vecin $v$ al lui $u$ **do**
12. $\quad\quad$ **if** $c[v] == \mathtt{w}$ **then**
13. $\quad\quad\quad c[v] := \mathtt{g}$
14. $\quad\quad\quad d[v] := d[u] + 1$
15. $\quad\quad\quad \pi[v] := u$
16. $\quad\quad\quad$ Enqueue($Q, v$)
17. $\quad c[u] := \mathtt{n}$

**Timp de execuție:** $O(|V| + |E|)$

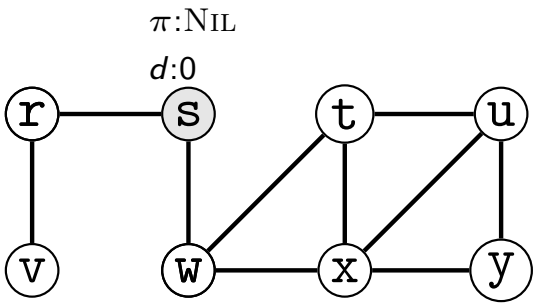
# Căutarea în lățime (BFS)

## Afișarea unei căi cu nr. minim de muchii de la nodul sursă $s$ la un nod $x$ (pseudocod)

```
AfiseazaCale(G, s, x)
1 if x == s then print s
2 else
3       if π[x] == Nil then
4         print "nu exista cale de la " s " la " x
5       else
6         AfiseazaCale(G, s, π[x])
7         print x
```

# Căutarea în lățime (BFS)

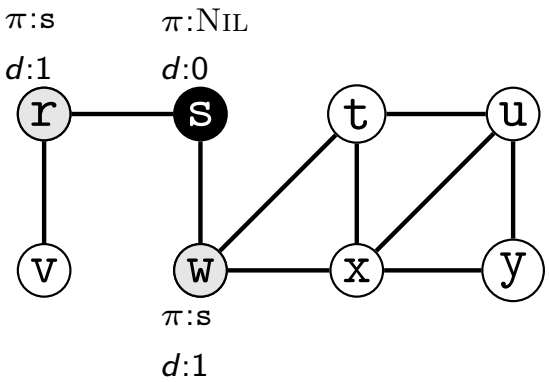
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathtt{s}]$

# Căutarea în lățime (BFS)

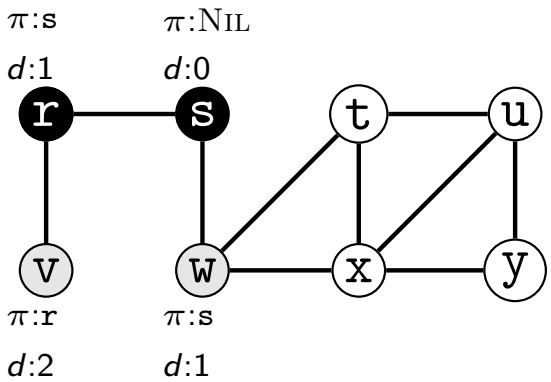
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathtt{r}, \mathtt{w}]$

# Căutarea în lățime (BFS)

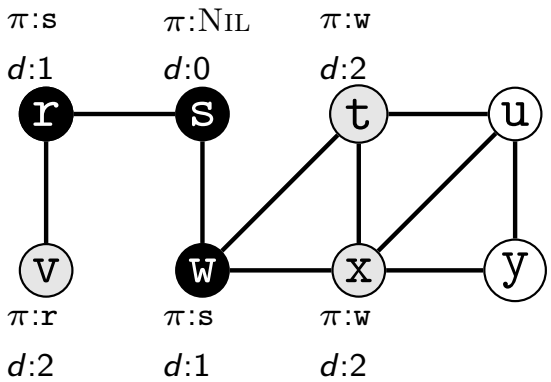
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathtt{w}, \mathtt{v}]$

# Căutarea în lățime (BFS)

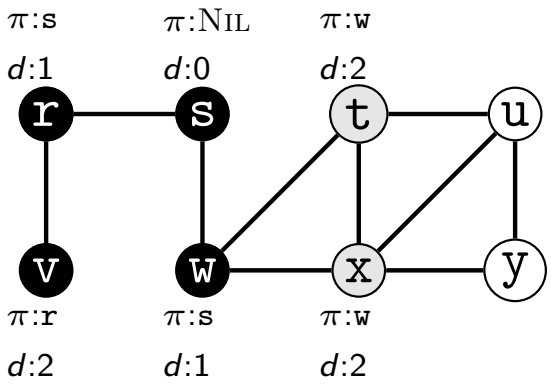
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathtt{v}, \mathtt{t}, \mathtt{x}]$

# Căutarea în lățime (BFS)

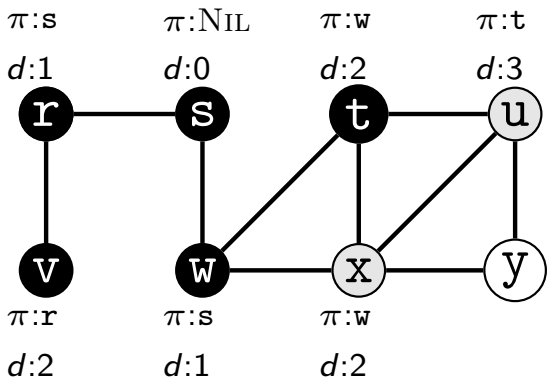
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathtt{t}, \mathtt{x}]$

# Căutarea în lățime (BFS)

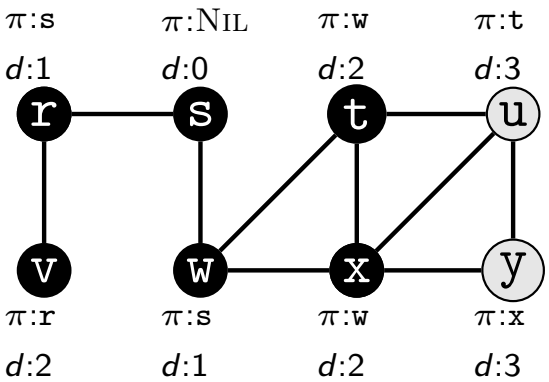
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathtt{x}, \mathtt{u}]$

# Căutarea în lățime (BFS)

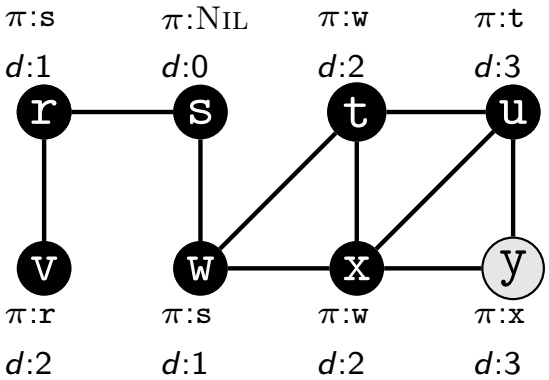
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathrm{u}, \mathrm{y}]$

# Căutarea în lățime (BFS)

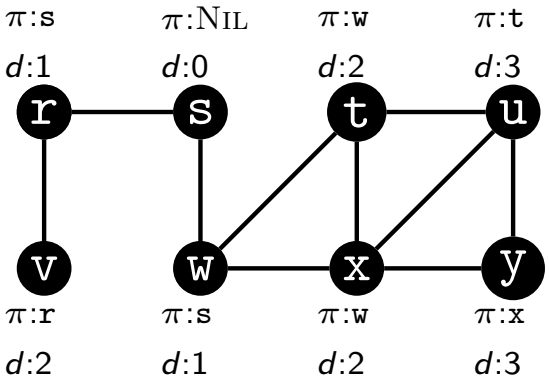
## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\mathrm{y}]$

# Căutarea în lățime (BFS)

## Exemplu ilustrat: căutare pornind de la sursa s




Conținutul listei curente:
$Q = [\ ]$

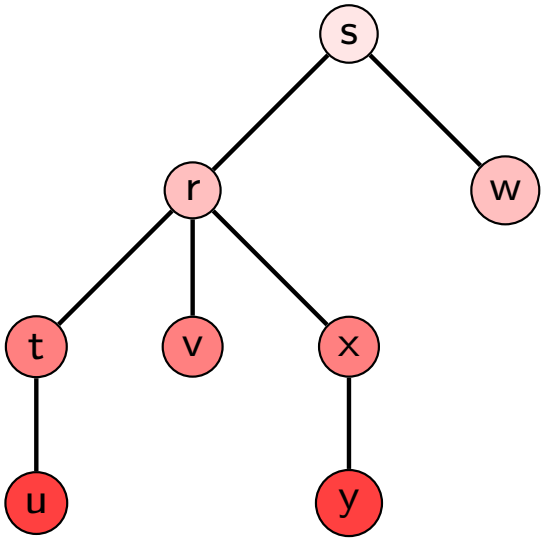
Algorimul BFS construiește arborele predecesor $G_\pi = (V_\pi, E_\pi)$ al lui $G = (V, E)$ unde

$$V_\pi := \{s\} \cup \{v \in V \mid \pi[v] \neq \text{NIL}\},$$
$$E_\pi := (\pi[v], v) \mid v \in V_\pi - \{s\}\}.$$

Proprietăți:

1. $V_\pi$ este mț. tuturor nodurilor la care se poate ajunge din $s$.
2. Fiecare ramură de la $s$ la un nod $x$ în $G_\pi$ este o cale cu nr. minim de muchii de la $s$ la $x$ în $G$.

Remarcă: $G_\pi$ pentru graful din exemplul ilustrat este




← - - - - - Nivel 1: noduri la distanță 1 de sursa s

← - - - - - Nivel 2: noduri la distanță 2 de sursa s

← - - - - - Nivel 3: noduri la distanță 3 de sursa s

# Căutarea în adâncime (**D**epth **F**irst **S**earch)

## Comparație cu căutarea în lățime (BFS)

Găsește toate nodurile la care putem ajunge de la un nod sursă $s$ cu o metodă de explorare a muchiilor grafului diferită de cea a BFS:

- Se caută mai întâi noduri nedescoperite învecinate cu cel mai recent nod descoperit.
- Dacă ultimul nod descoperit nu are vecini nedescoperiți, se revine cu căutarea la nodul descoperit anterior (*engl.* backtracking)

**Asemănări cu DFS:**

- Reține predecesorii nodurilor $x$ de-a lungul căilor găsite într-un tablou $\pi[x]$
- Pe durata căutării, distinge 3 tipuri de noduri:
  - albe (a): nodurile nedescoperite încă
  - gri (g): nodurile descoperite
  - negre (n): nodurile descoperite, cu toți vecinii descoperiți
- De regulă, presupunem că $G$ este reprezentat cu liste de adiacență.

- Pentru fiecare nod $x$ la care se poate ajunge din $s$, reține două valori temporale (numere întregi):
  - $d[x]$: momentul descoperirii nodului $x$
  - $f[x]$: momentul finalizării examinării nodului $x$ (când nodul devine negru)
- Construiește o pădure $G_\pi = (V, E_\pi)$ unde $E = \{(\pi[v], v) \mid v \in V - \{s\}\}$.
- $G_\pi$ are următoarele proprietăți:
  1. Fiecare arbore conține toate nodurile la care se poate ajunge de la rădăcina lui.
  2. Unul din arbori are rădăcina $s$.
- Diferența esențială dintre DFS și BFS este că
  - DFS reține într-o stivă $S$ nodurile gri care urmează să fie finalizate.
  - BFS le reține într-o listă FIFO $Q$.

# Căutarea în adâncime (DFS)

## Pseudocod (versiunea iterativă)

DFS($G, s$)
1. **for** fiecare nod $u \in V$ **do**
2. $\quad c[u] := \mathtt{a}$
3. $\quad \pi[u] := \textsc{Nil}$
4. $timp := 0$
5. $S := []$
6. **for** fiecare nod $u \in V$, începând cu $s$ **do**
7. $\quad$ **if** $c[u] == \mathtt{a}$ **then**
8. $\quad\quad timp := timp + 1;\ c[u] := \mathtt{g};\ d[u] := timp$
9. $\quad\quad$ Push($S, u$)
10. $\quad\quad$ **while** $S \neq []$ **do**
11. $\quad\quad\quad v :=$Pop()
12. $\quad\quad\quad$ **for** fiecare vecin $x$ al lui $v$ **do**
13. $\quad\quad\quad\quad$ **if** $c[x] == \mathtt{a}$ **then**
14. $\quad\quad\quad\quad\quad timp := timp + 1;\ c[x] := \mathtt{g};\ d[x] := timp;\ \pi[x] := v$
15. $\quad\quad\quad\quad\quad$ Push($S, x$)
16. $\quad\quad\quad timp := timp + 1;\ c[v] := \mathtt{n};\ f[v] := timp$

**Timp de execuție** $O(|V| + |E|)$

# Căutarea în adâncime (DFS)

## Pseudocod (versiunea recursivă)

DFS($G, s$)
1. **for** fiecare nod $u \in V$ **do**
2. $\quad c[u] := \mathtt{a}$
3. $\quad \pi[u] := \textsc{Nil}$
4. $timp := 0$
5. $S := []$
6. **for** fiecare nod $u \in V$, începând cu $s$ **do**
7. $\quad$ **if** $c[u] == \mathtt{a}$ **then**
8. $\quad\quad$ DFS_Visit($u$)

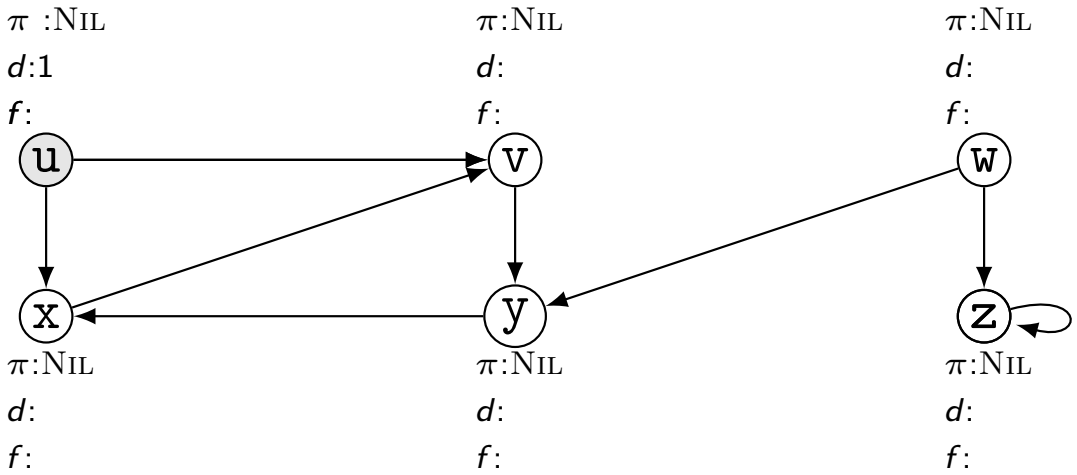
1. DFS_Visit($u$)
2. $\quad timp := timp + 1$; $c[u] := \mathtt{g}$; $d[u] := timp$
3 $\quad$ **for** fiecare vecin $v$ al lui $u$ **do**
4. $\quad\quad$ **if** $c[v] == \mathtt{a}$ **then**
5. $\quad\quad\quad \pi[v] := u$
6. $\quad\quad\quad$ DFS_Visit($v$)
7. $\quad timp := timp + 1$; $c[u] := \mathtt{n}$; $f[u] := timp$

**Remarcă:** În această versiune, utilizarea unei stive este implicită.

# Căutarea în adâncime (dfs)

## Exemplu ilustrat

Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:




Conținutul stivei curente:
$S = [\mathtt{u}]$

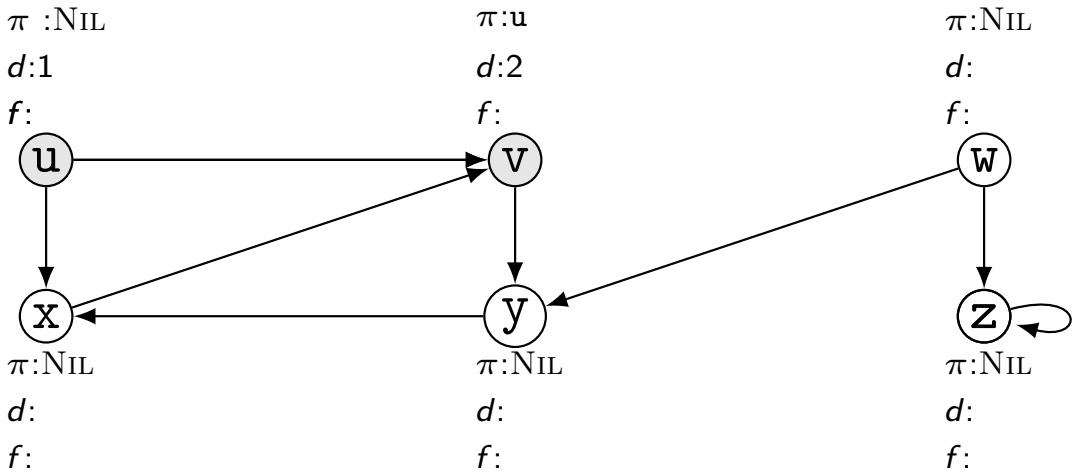
Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$

# Căutarea în adâncime (dfs)

## Exemplu ilustrat

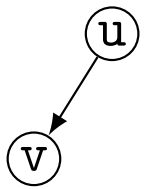
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:




Conținutul stivei curente:
$S = [\mathrm{v}, \mathrm{u}]$

Reprezentarea lui $G$ cu liste de adiacență

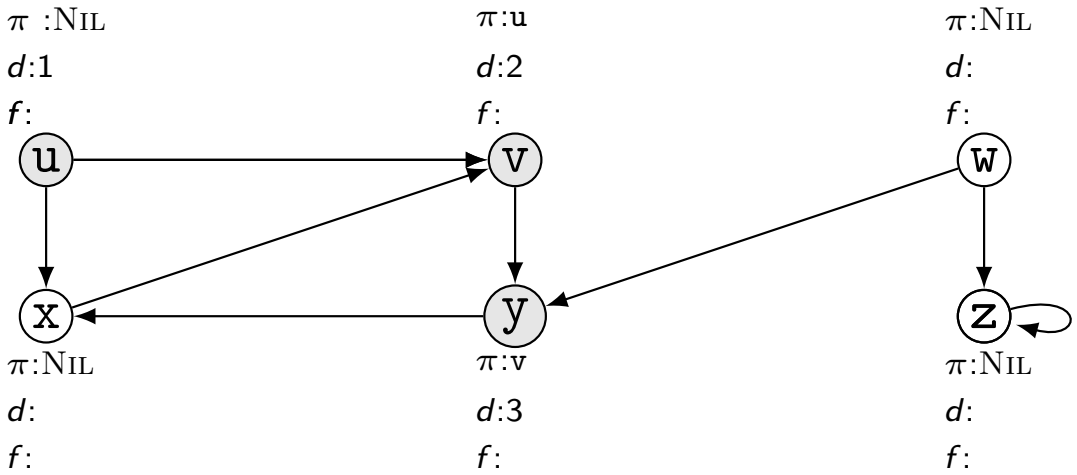
$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



# Căutarea în adâncime (dfs)

## Exemplu ilustrat

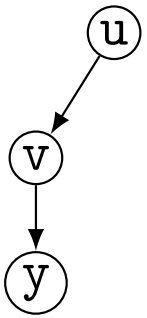
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:




Conținutul stivei curente:
$S = [\mathrm{y}, \mathrm{v}, \mathrm{u}]$

Reprezentarea lui $G$ cu liste de adiacență

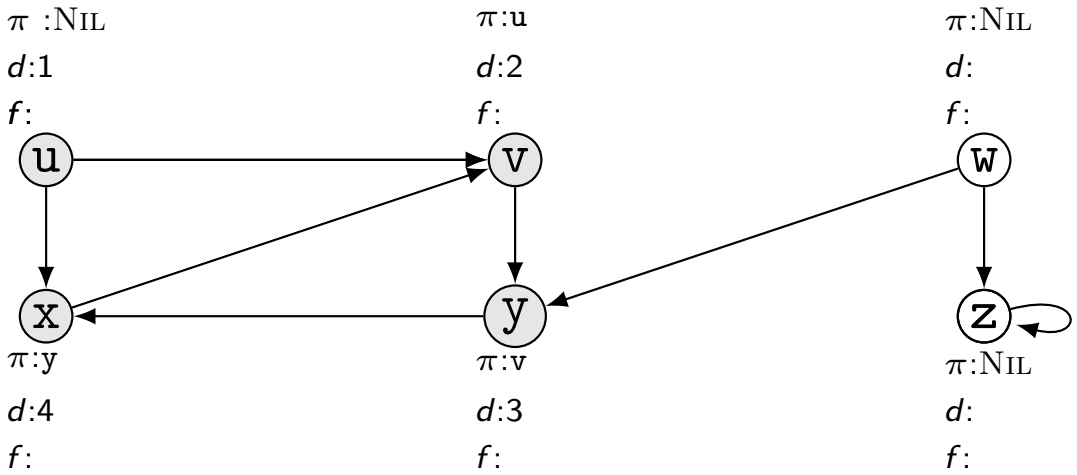
$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



# Căutarea în adâncime (dfs)

## Exemplu ilustrat

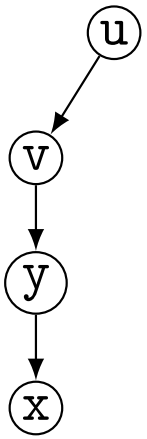
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:



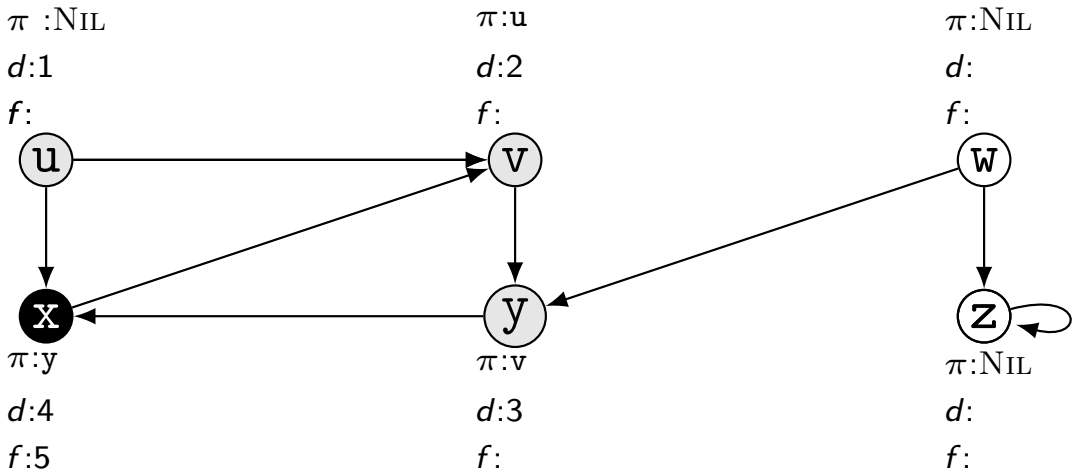

Conținutul stivei curente:
$S = [\mathrm{x}, \mathrm{y}, \mathrm{v}, \mathrm{u}]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



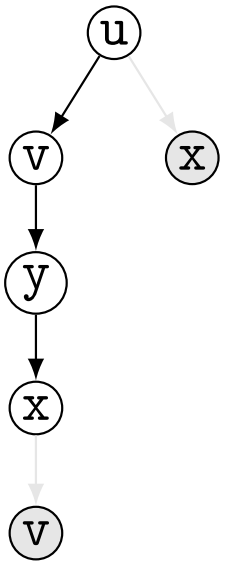
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:



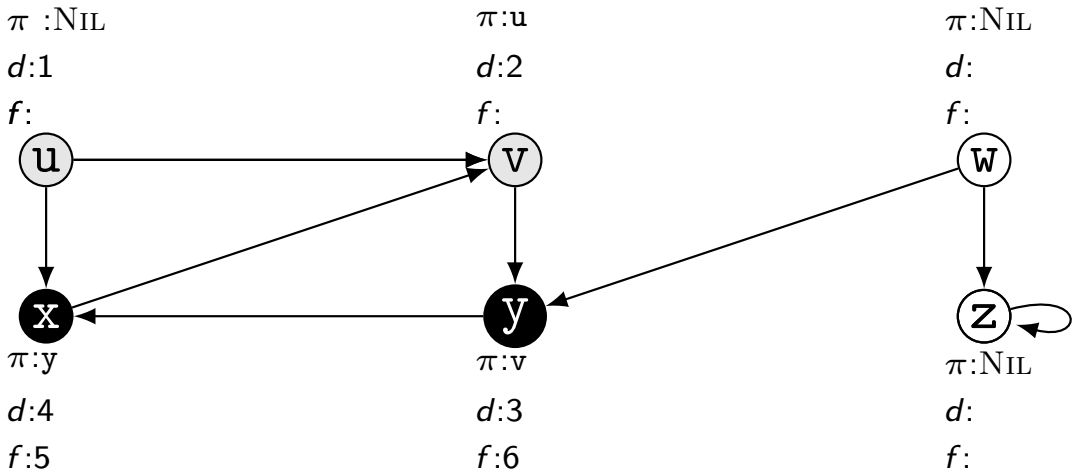

Conținutul stivei curente:
$S = [\mathrm{y}, \mathrm{v}, \mathrm{u}]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



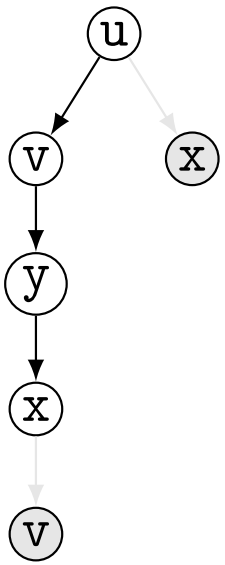
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:



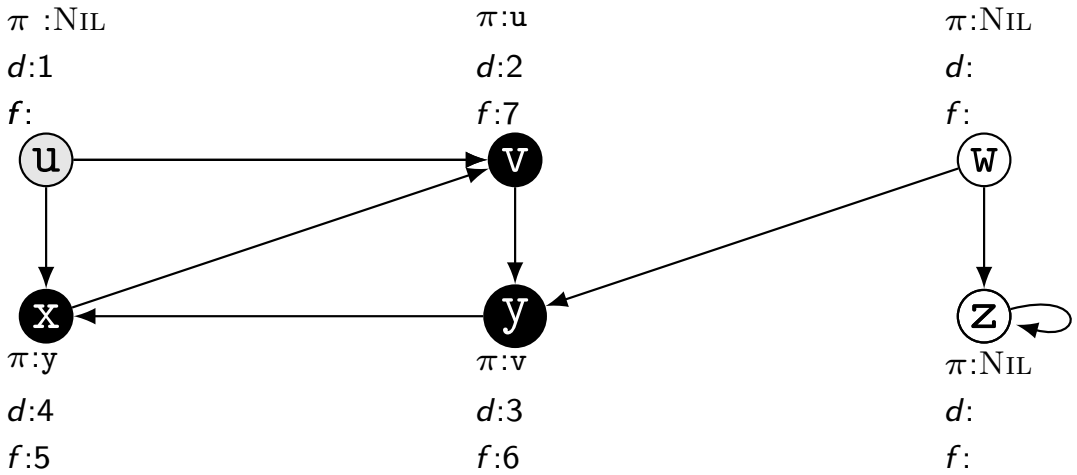

Conținutul stivei curente:
$S = [\mathrm{v}, \mathrm{u}]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



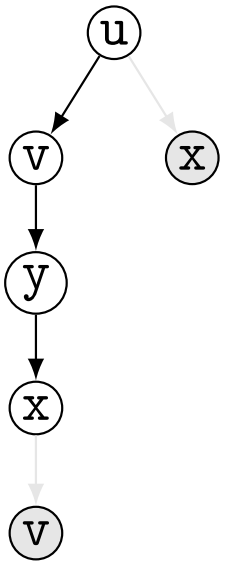
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:



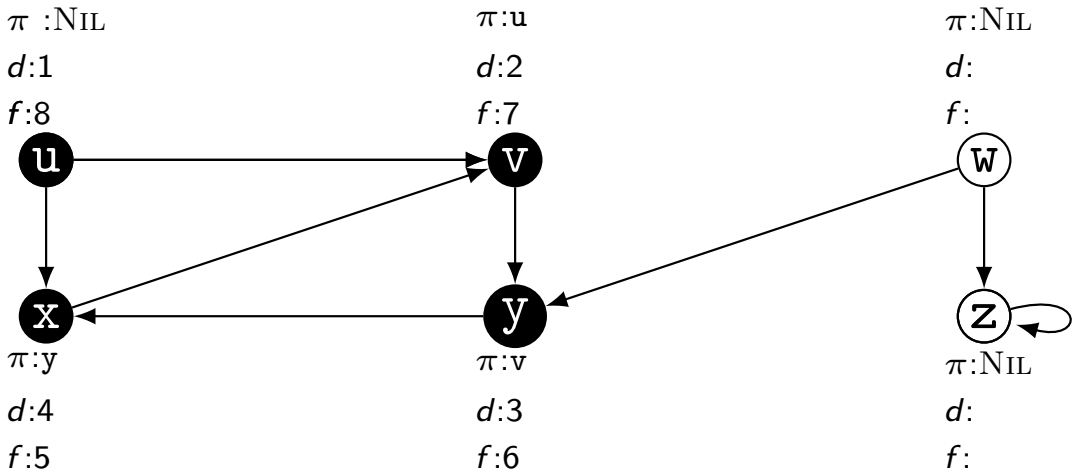

Conținutul stivei curente:
$S = [\mathrm{u}]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



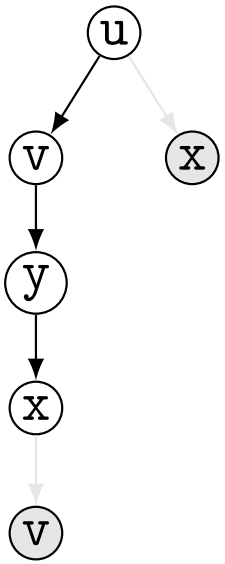
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:



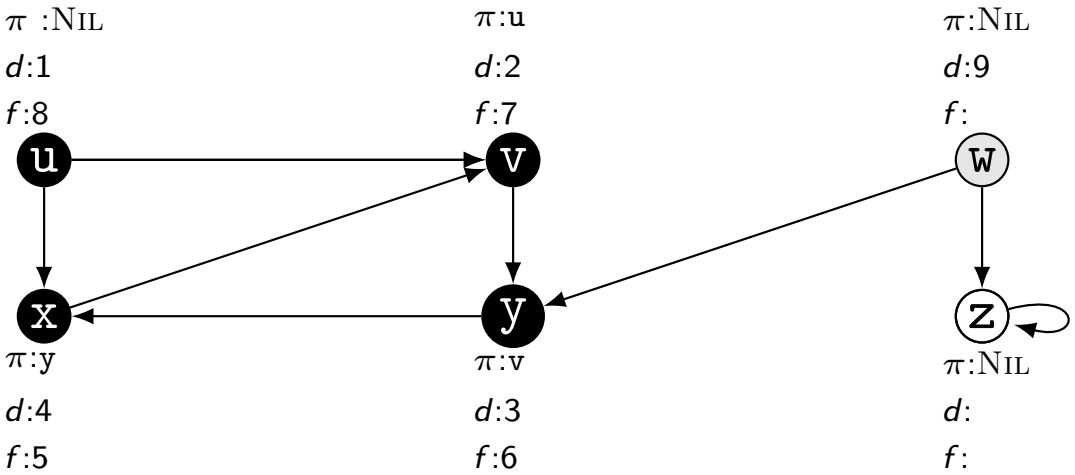

Conținutul stivei curente:
$S = [\,]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



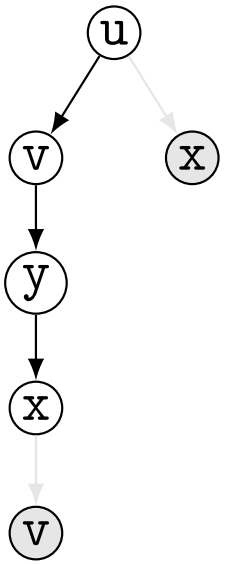
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:



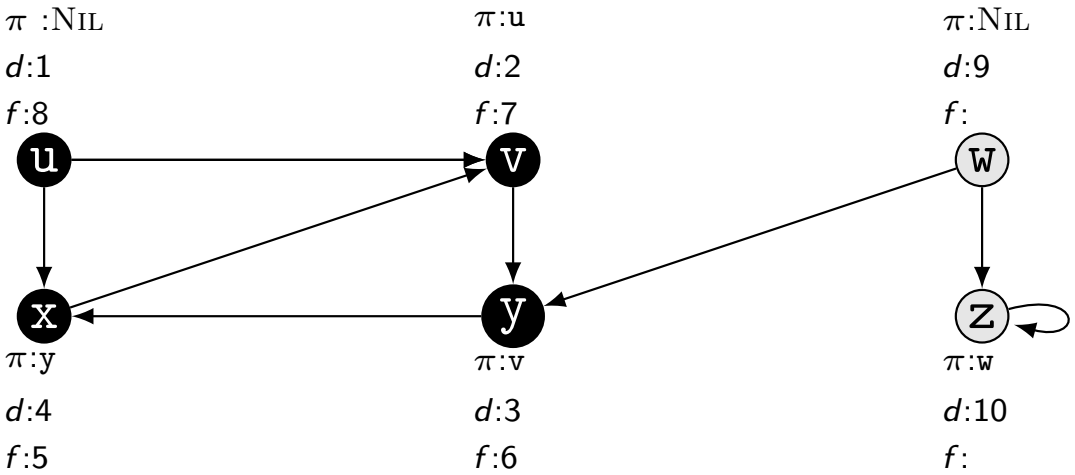

Conținutul stivei curente:
$S = [\mathtt{w}]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



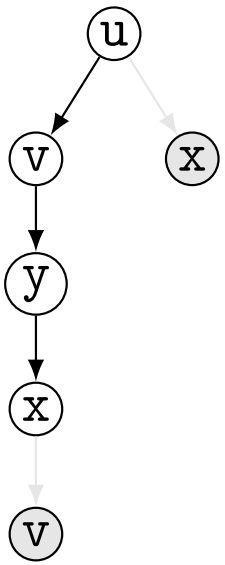

Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:

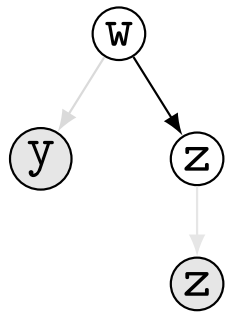



Conținutul stivei curente:
$S = [\mathtt{z}, \mathtt{w}]$

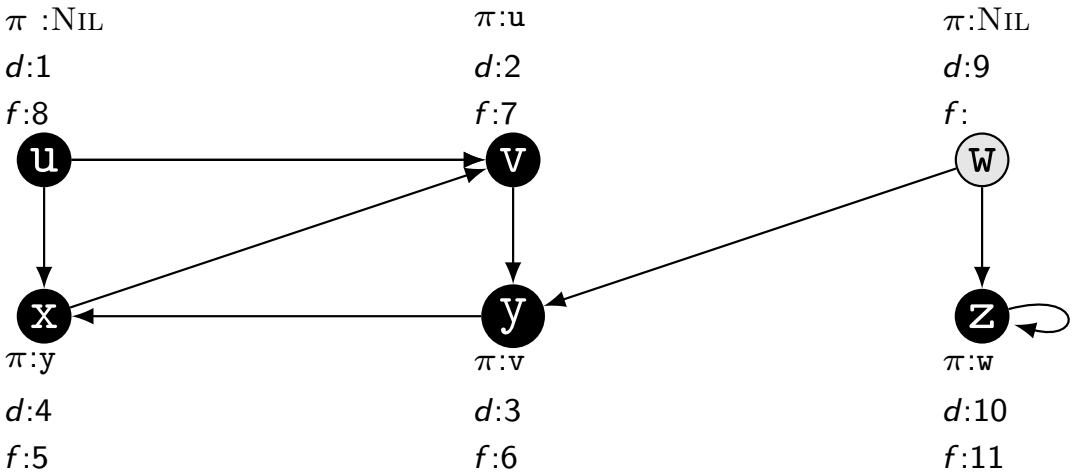
Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



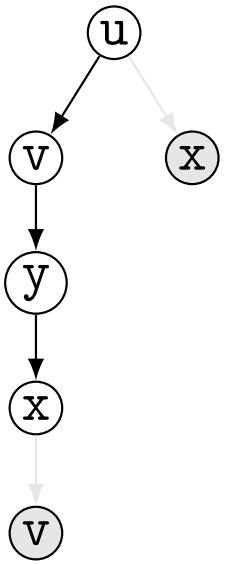
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:

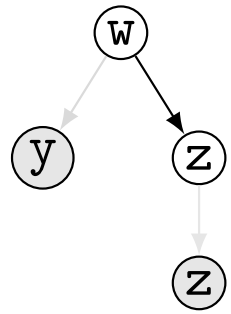



Conținutul stivei curente:
$S = [\mathtt{w}]$

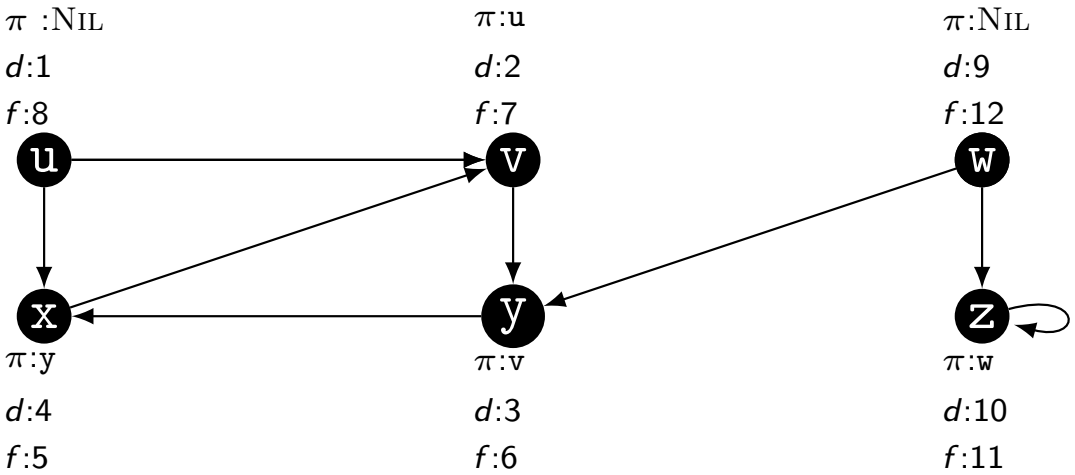
Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



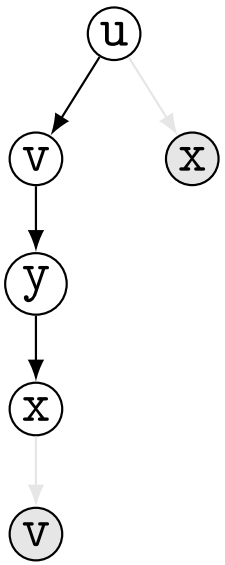
Digraf $G = (V, E)$ cu nodul sursă u ilustrat mai jos:

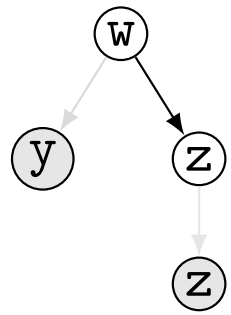



Conținutul stivei curente:
$S = [\,]$

Reprezentarea lui $G$ cu liste de adiacență

$x \mapsto [v]$
$y \mapsto [x]$
$w \mapsto [y, z]$
$u \mapsto [v, x]$
$v \mapsto [y]$
$z \mapsto [z]$



# Aplicații ale algoritmilor de căutare

- Ambii algoritmi construiesc un arbore de acoperire pentru subgraful lui $G$ ce conține toate nodurile la care se poate ajunge din nodul sursă.
- Dacă $G$ este conex, $G_\pi$ este un arbore de acoperire al lui $G$.

Aplicații ale căutării în lățime (BFS):

- Găsește căile cele mai scurte de la $s$ la nodurile la care se poate ajunge din $s$
  - Căutarea în adâncime nu garantează această proprietate.

Aplicații ale căutării în adâncime (DFS):

1. Calculul tuturor componentelor conexe ale lui $G$.
2. Detecția ciclurilor în grafuri orientate (vezi mai departe)
3. Sortarea topologică a grafurilor orientate fără cicluri (vezi mai departe)

# Sortare topologică

Terminologie:

- Un graf orientat aciclic (engl. dag) este un graf orientat fără cicluri.
- O sursă a unui dag este un nod $s$ cu $\deg^-(s) = 0$. O destinație a unui dag este un nod $t$ cu $\deg^+(t) = 0$.
- O sortare (sau numerotare) topologică a unui dag $G = (V, E)$ atribuie un întreg distinct $\texttt{I}[x]$ fiecărui nod $x$, astfel încât $\texttt{I}[u] < \texttt{I}[v]$ pentru fiecare arc $u \xrightarrow{e} v \in E$.

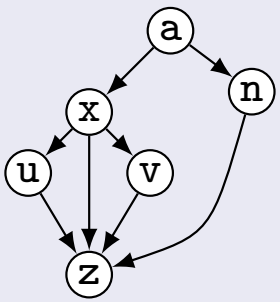
# Sortare topologică

Terminologie:

- Un graf orientat aciclic (engl. dag) este un graf orientat fără cicluri.
- O sursă a unui dag este un nod $s$ cu $\deg^-(s) = 0$. O destinație a unui dag este un nod $t$ cu $\deg^+(t) = 0$.
- O sortare (sau numerotare) topologică a unui dag $G = (V, E)$ atribuie un întreg distinct $\mathtt{I}[x]$ fiecărui nod $x$, astfel încât $\mathtt{I}[u] < \mathtt{I}[v]$ pentru fiecare arc $u \xrightarrow{e} v \in E$.

Remarcă: Orice dag poate fi desenat încât toate arcele să fie orientate în jos. Această reprezentare permite calculul unei sortări topologice a dag-ului respectiv.

## Exemplu




$\mathtt{I}[\mathtt{a}] = 1$

$\mathtt{I}[\mathtt{n}] = 2$ $\quad \mathtt{I}[\mathtt{x}] = 3$

$\mathtt{I}[\mathtt{u}] = 4$, $\mathtt{I}[\mathtt{v}] = 5$

$\mathtt{I}[\mathtt{z}] = 6$

a este sursă a dag-ului.

z este destinație a dag-ului.

Căutarea în adâncime (DFS) ne permite ca pentru un graf orientat $G = (V, E)$

1. Să detectăm dacă $G$ are cicluri sau nu.
   - $G$ are un ciclu dacă și numai dacă are o muchie $(u, v)$ cu $d[v] < d[u] < f[u] < f[v]$.
     (Vezi Cormen *et al.*: Introduction to Algorithms, cap. 22)

   $\Rightarrow$ putem verifica în timp $O(|V| + |E|)$ dacă $G$ este dag sau nu.
2. Dacă $G$ este dag (adică nu are cicluri), să calculăm o sortare topologică pentru $G$:

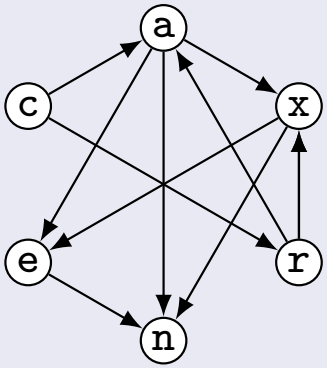
$$\texttt{I}[x] := 2 \cdot |V| - f[x] \quad \text{pentru toți } x \in V$$

   - o sortare topologică se poate găsi în timp $O(|V| + |E|)$

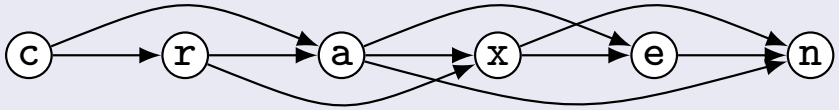
*Input:* un dag $G = (V, E)$ cu $n$ noduri, și $s \in V$
*Output:* lungimile maxime $L[v]$ de drumuri simple de la $s$

1. *se calculează o numerotare topologică* $\{\mathtt{I}[v] \mid v \in V\}$ *în timp* $O(|V| + |E|)$
2. Fie $[v_1, \ldots, v_n]$ nodurile lui $V$ în ordine topologică crescătoare și $k$ indexul pentru care $s = v_k$
3. $L[v_k] := 0$;
4. **for** $i := k + 1$ **to** $n$ do $L[v_i] := -\infty$;
5. **for** $i := k + 1$ **to** $n$
6. **for** fiecare $(v_i, w) \in E$ **do**
7. **if** $L[w] < L[v_i] + 1$ **then** $L[w] = L[v_i] + 1$;
8. **endfor**
9. **endfor**

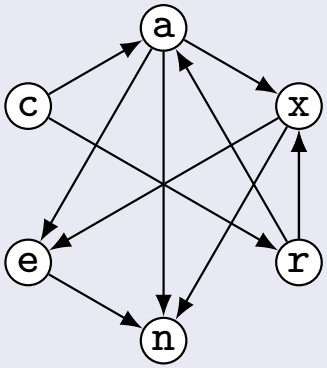
## Exemplu ilustrat



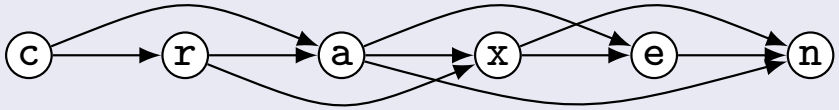

are sortarea topologică



## Exemplu ilustrat




are sortarea topologică

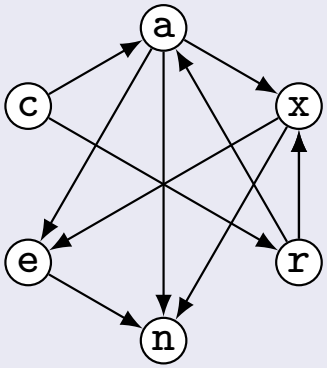



Lungimi maxime de la nodul c: $0 \quad -\infty \quad -\infty \quad -\infty \quad -\infty \quad -\infty$

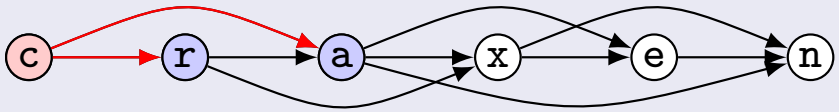
## Observații

- Pentru grafuri arbitrare, calculul celor mai lungi drumuri simple de la un nod este o problemă dificilă (NP-completă)

## Exemplu ilustrat




are sortarea topologică



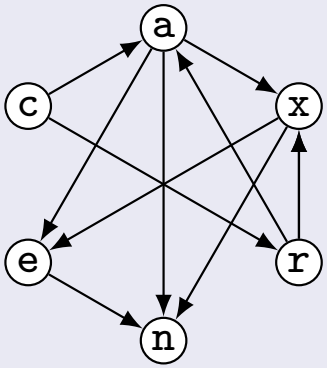

Lungimi maxime de la nodul c:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| | 1 | 1 | $-\infty$ | $-\infty$ | $-\infty$ |

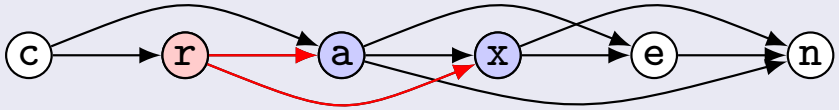
**Observații**

- Pentru grafuri arbitrare, calculul celor mai lungi drumuri simple de la un nod este o problemă dificilă (NP-completă)
- Pentru dag-uri, calculul celor mai lungi drumuri simple se poate face în timp liniar $O(|V| + |E|)$.

## Exemplu ilustrat




are sortarea topologică



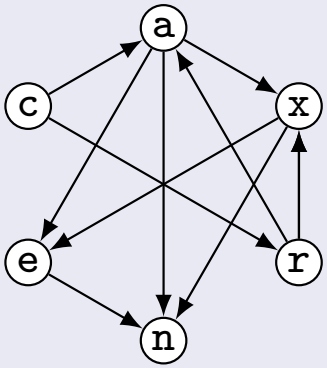

Lungimi maxime de la nodul c:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| | 1 | 1 | $-\infty$ | $-\infty$ | $-\infty$ |
| | | 2 | 2 | $-\infty$ | $-\infty$ |

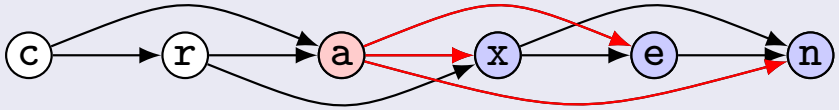
**Observații**

- Pentru grafuri arbitrare, calculul celor mai lungi drumuri simple de la un nod este o problemă dificilă (NP-completă)
- Pentru dag-uri, calculul celor mai lungi drumuri simple se poate face în timp liniar $O(|V| + |E|)$.

## Exemplu ilustrat




are sortarea topologică



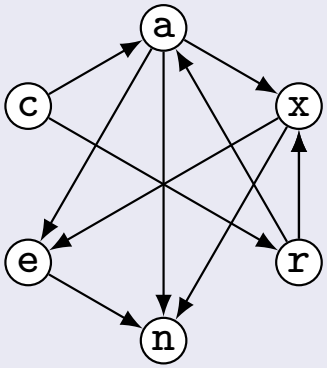

Lungimi maxime de la nodul c:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| | 1 | 1 | $-\infty$ | $-\infty$ | $-\infty$ |
| | | 2 | 2 | $-\infty$ | $-\infty$ |
| | | | 3 | 3 | 3 |

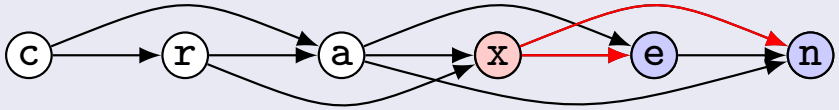
## Observații

- Pentru grafuri arbitrare, calculul celor mai lungi drumuri simple de la un nod este o problemă dificilă (NP-completă)
- Pentru dag-uri, calculul celor mai lungi drumuri simple se poate face în timp liniar $O(|V| + |E|)$.

## Exemplu ilustrat




are sortarea topologică



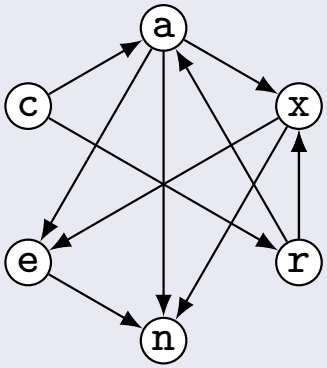

Lungimi maxime de la nodul c:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| | 1 | 1 | $-\infty$ | $-\infty$ | $-\infty$ |
| | | 2 | 2 | $-\infty$ | $-\infty$ |
| | | | 3 | 3 | 3 |
| | | | | 4 | 4 |

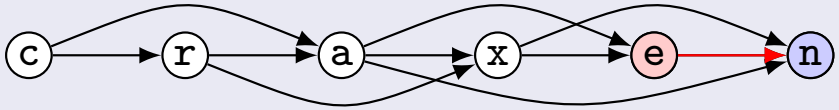
**Observații**

- Pentru grafuri arbitrare, calculul celor mai lungi drumuri simple de la un nod este o problemă dificilă (NP-completă)
- Pentru dag-uri, calculul celor mai lungi drumuri simple se poate face în timp liniar $O(|V| + |E|)$.

## Exemplu ilustrat




are sortarea topologică




Lungimi maxime de la nodul c:

| | | | | | |
|---|---|---|---|---|---|
| 0 | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ | $-\infty$ |
| | 1 | 1 | $-\infty$ | $-\infty$ | $-\infty$ |
| | | 2 | 2 | $-\infty$ | $-\infty$ |
| | | | 3 | 3 | 3 |
| | | | | 4 | 4 |
| | | | | | 5 |

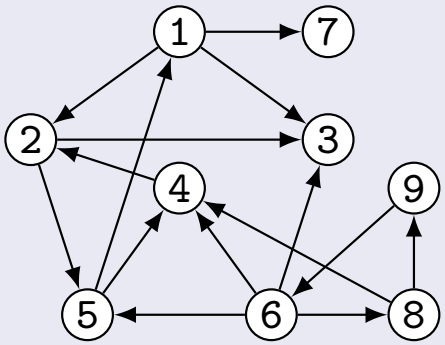
## Observații

- Pentru grafuri arbitrare, calculul celor mai lungi drumuri simple de la un nod este o problemă dificilă (NP-completă)
- Pentru dag-uri, calculul celor mai lungi drumuri simple se poate face în timp liniar $O(|V| + |E|)$.

Dacă $G$ este graf neorientat, atunci pădurea $G_\pi$ calculată de DFS este formată din arbori de acoperire pentru componentele conexe ale lui $G$.

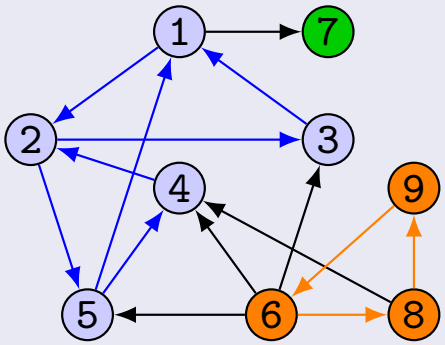
- Orice graf orientat poate fi descompus în mod unic în componente distincte.
- Dacă $G$ conține un ciclu $C_n$, atunci $C_n$ este conținut într-o componentă conexă a lui $G$.
- Dacă $G$ conține un nod $v$ cu $deg^+(v) = 0$ atunci $\{v\}$ este o componentă conexă a lui $G$.

## Exemplu



- Orice graf orientat poate fi descompus în mod unic în componente distincte.
- Dacă $G$ conține un ciclu $C_n$, atunci $C_n$ este conținut într-o componentă conexă a lui $G$.
- Dacă $G$ conține un nod $v$ cu $deg^+(v) = 0$ atunci $\{v\}$ este o componentă conexă a lui $G$.

## Exemplu




Componentele conexe sunt:

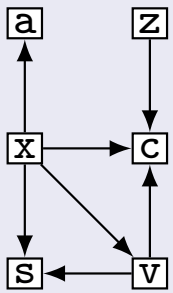
$C_1 = \{7\}$ (nod pendant)
$C_2 = \{1, 2, 3, 4, 5\}$
$C_3 = \{6, 8, 9\}$

Fie $G = (V, E)$ un graf orientat și $S \subseteq V$.
Contracția lui $S$ în $G$ este graful construit astfel:

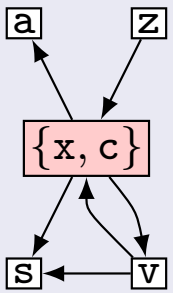
- se elimină nodurile lui $S$ din $G \Rightarrow$ graful $G_1 = G - S$
- se adaugă la $G_1$ un nod nou $x$ etichetat cu $S$, și se adaugă:
  - câte o muchie $u \xrightarrow{e} x$ pentru fiecare muchie $u \xrightarrow{e} v \in E$ cu $v \in S$.
  - câte o muchie $x \xrightarrow{e} w$ pentru fiecare muchie $v \xrightarrow{e} w \in E$ cu $v \in S$.

**Exemplu**

Contracția lui $\{x, y\}$ în




este graful orientat

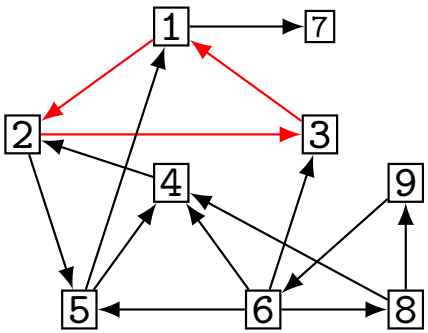


*Input:* graf orientat $G = (V, E)$
*Output:* componentele conexe ale lui $G$

1. **while** $E \neq \emptyset$ **do**
2. *Se construiește cu dfs un drum $P$ până se detectează un ciclu $C$ sau un nod destinație $t$;*
3. **if** $t$ a fost găsit **then**
   *Se reține că $\{t\}$ este componentă conexă a lui $G$;*
   *Se elimină $t$ din $G$ și $P$;*
   **endif**
4. **if** $C$ a fost găsit
   *Se contractă nodurile lui $C$ în $G$;*
   **endif**
5. **endwhile**

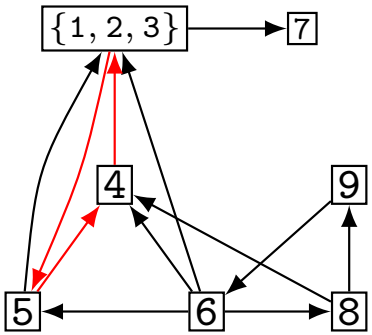
# Determinarea componentelor conexe ale unui graf orientat

## Exemplu ilustrat



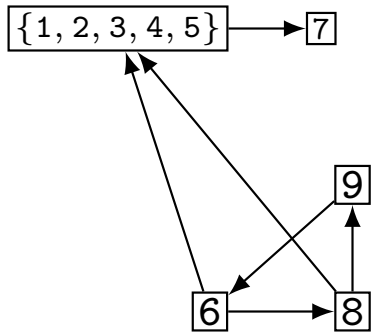

Consider ciclul (1,2,3,1)

$\Rightarrow$

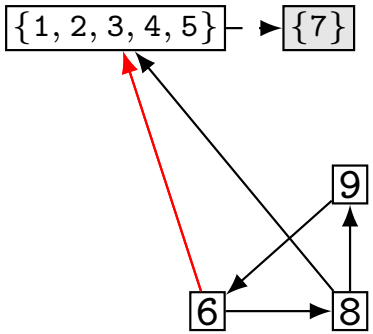



Consider ciclul
$(\{1, 2, 3\}, 5, 4, \{1, 2, 3\})$

$\Rightarrow$

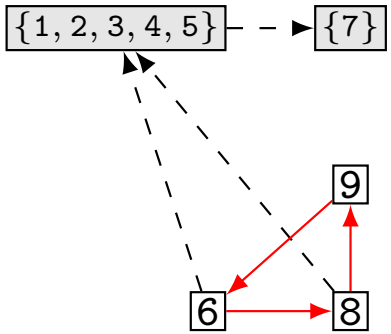



Consider drumul bfs
$(\{1, 2, 3, 4, 5\}, 7)$




Am găsit componenta $\{7\}$
Consider drumul dfs
$(6, \{1, 2, 3, 4, 5\})$

$\Rightarrow$




Am găsit componenta
$\{1, 2, 3, 4, 5\}$
Consider ciclul (9,6,8,9)

$\Rightarrow$