Cursul 9

Traversarea grafurilor

noiembrie 2020

G cu n noduri și m muchii

- Cu liste de muchii și liste de noduri
- Cu liste de adiacență
- Cu matrice de adiacență A_G de dimensiune $n \times n$
- Cu matrice de incidență M_G de dimensiune $n \times m$
- Cu matrice de ponderi W_G de dimensiune $n \times n$

- Reprezentarea cu listă de muchii
  - Adecvată pentru reprezentarea grafurilor simple fără noduri izolate, cu $m \ll n^2$
  - Complexitate spațială (memorie ocupată): $O(m)$
- Reprezentarea cu liste de adiacență
  - Permite enumerarea rapidă a vecinilor unui nod
  - Complexitate spațială (memorie ocupată): $O(n+m)$
- Reprezentarea cu matrice de adiacență $A_G = (a_{ij})$ sau cu matrice de ponderi $W_G = (w_{ij})$
  - Test rapid de conectivitate directă între 2 noduri: $O(1)$
    - $\nexists (v_i, v_j) \in E$ dacă $a_{ij} = 0$ sau dacă $w_{ij} = \infty$
  - Complexitate spațială (memorie ocupată): $O(n^2)$
    - reprezentare neadecvată când $m \ll n^2$
- Reprezentarea cu matrice de incidență M_G
  - Complexitate temporală: $O(n \cdot m)$

Reprezentarea arborilor cu rădăcină

Funcția predecesor $\mathtt{p}[x]$

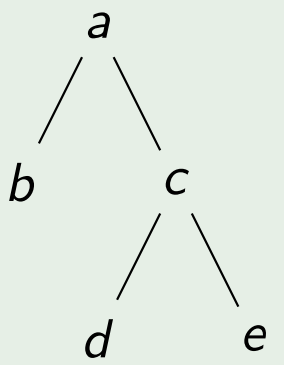
Arbore cu rădăcină = digraf G obținut dintr-un arbore $T = (V, E)$:

- alegem un nod $s \in V$ = rădăcina digrafului
- toate muchiile se orientează să meargă dinspre părinte spre fii

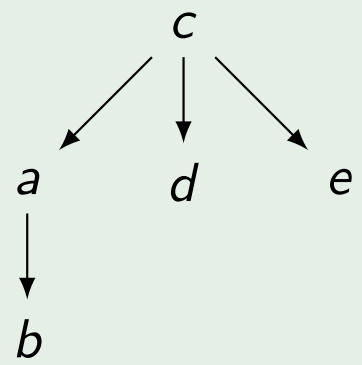
Reprezentarea cu predecesor:

$\mathtt{p}[s] = \mathtt{null}$
dacă $x \in V - \{s\}$: $\mathtt{p}[x]$ = părintele, sau predecesorul direct al lui x

Exemplu



Arbore cu rădăcina c



$\mathtt{p}[c] = \mathtt{null}$

$\mathtt{p}[a] = \mathtt{p}[d] = \mathtt{p}[e] = c$

$\mathtt{p}[b] = a$

Se vizitează nodul s.

- Vizitarea unui nod x se face astfel:
  1. Se marchează x ca nod vizitat.
  2. Se vizitează recursiv toți vecinii nevizitați ai lui x. De obicei, pentru fiecare vecin y care se vizitează se setează $\mathrm{p}[y] = x$ pentru a reține faptul că traversarea s-a făcut de la x la y.

Se vizitează nodul s.

- Vizitarea unui nod x se face astfel:
  1. Se marchează x ca nod vizitat.
  2. Se vizitează recursiv toți vecinii nevizitați ai lui x. De obicei, pentru fiecare vecin y care se vizitează se setează $\mathrm{p}[y] = x$ pentru a reține faptul că traversarea s-a făcut de la x la y.

  $\Rightarrow$ un arbore de traversare în adâncime cu rădăcina s.

Se vizitează nodul s.

- Vizitarea unui nod x se face astfel:
  1. Se marchează x ca nod vizitat.
  2. Se vizitează recursiv toți vecinii nevizitați ai lui x. De obicei, pentru fiecare vecin y care se vizitează se setează $p[y] = x$ pentru a reține faptul că traversarea s-a făcut de la x la y.

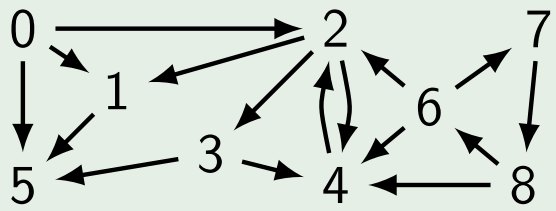
$\Rightarrow$ un arbore de traversare în adâncime cu rădăcina s.

```
private void dfs(Graph G, int i) { // iniţiază traversarea de la i
   vizitat[i] = true;
   for (int j : G.adj(i))
       if (!vizitat[j]) {
          p[j] = i;     // reţine că s-a traversat muchia i−j
          dfs(G,j);
       }
}
```

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

Exemplu

G:



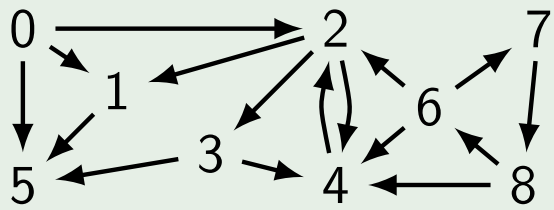
$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

$\Rightarrow$ arborele de traversare în adâncime:

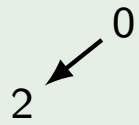
1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

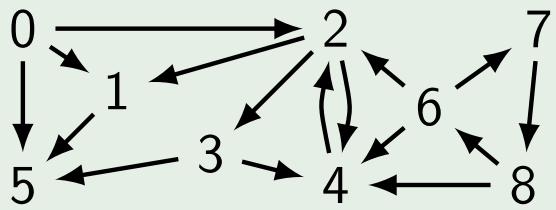


$\Rightarrow$ arborele de traversare în adâncime:

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

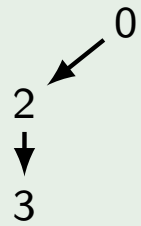
Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

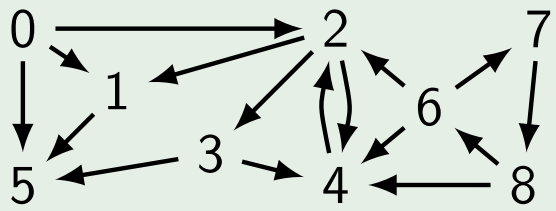
$\Rightarrow$ arborele de traversare în adâncime:



1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

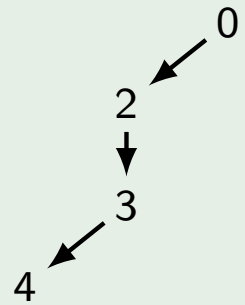
Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

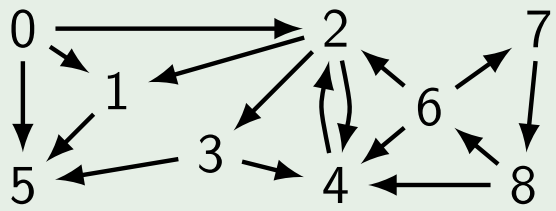
$\Rightarrow$ arborele de traversare în adâncime:



1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

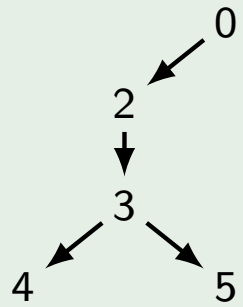
Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

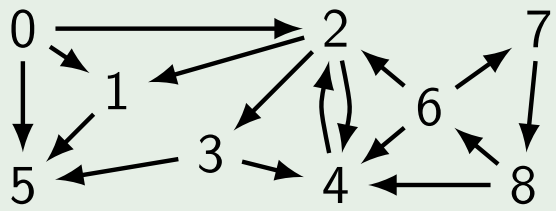
$\Rightarrow$ arborele de traversare în adâncime:



1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

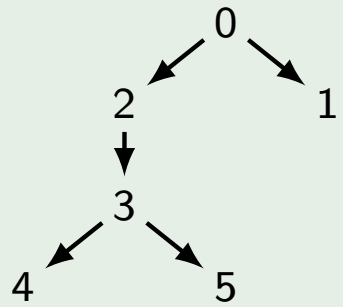
Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

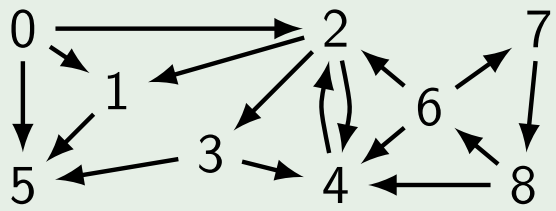
$\Rightarrow$ arborele de traversare în adâncime:



1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

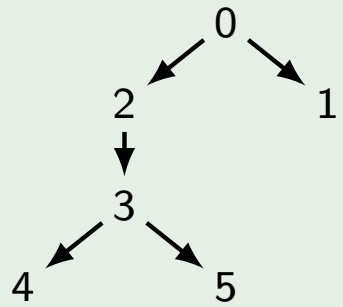
Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

$\Rightarrow$ arborele de traversare în adâncime:

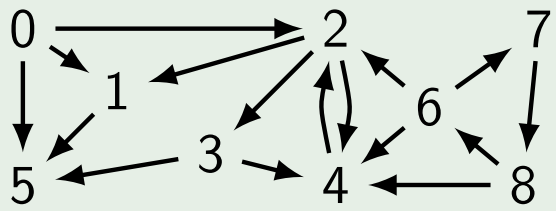


- $\{x \mid 0 \rightsquigarrow x\} = \{0, 1, 2, 3, 4, 5\}$

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de la s la x.

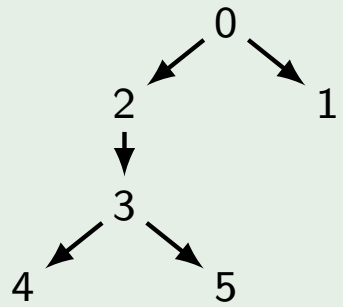
Exemplu

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

$\Rightarrow$ arborele de traversare în adâncime:



- $\{x \mid 0 \rightsquigarrow x\} = \{0, 1, 2, 3, 4, 5\}$
- s-au găsit drumurile $[0], [0, 1], [0, 2], [0, 2, 3], [0, 2, 3, 4], [0, 2, 3, 5]$

Traversarea în lățime de la un nod sursă s se face în runde:

- în prima rundă vizităm s
- în fiecare rundă următoare vizităm vecinii nevizitați ai nodurilor vizitate în runda precedentă.

```
private void bfs(Graph G, int s) {
   vizitat[s] = true;        // vizitează nodul sursă
   Queue<Integer> Q = new Queue<Integer>();
   Q.enqueue(s);             // şi il pune in coadă
   while(!Q.isEmpty()) {
      int v = Q.dequeue();
      for(int w : G.adj(v))
         if(!vizitat[w]) {
            p[w] = v;
            vizitat[w] = true;
            Q.enqueue(w);
         }
   }
}
```

Traversarea în lățime

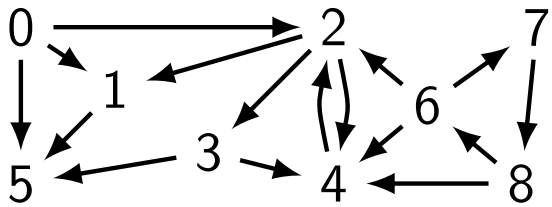
Proprietăți

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x.

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x.

Exemplu ilustrat de traversare în lățime de la nodul 0

G:

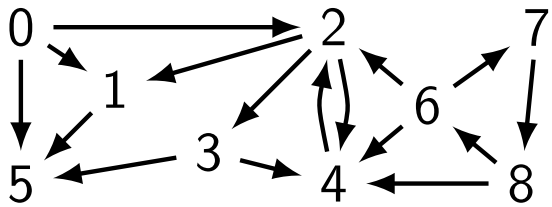


$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x.

Exemplu ilustrat de traversare în lățime de la nodul 0

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

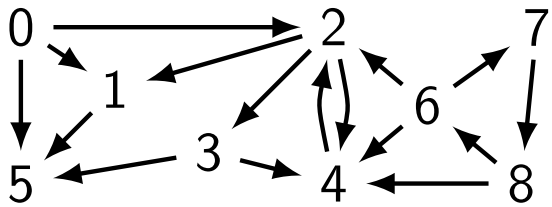
0

$\Rightarrow$ arborele de traversare în lățime:

1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x.

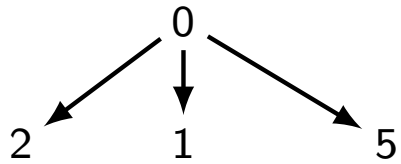
Exemplu ilustrat de traversare în lățime de la nodul 0

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

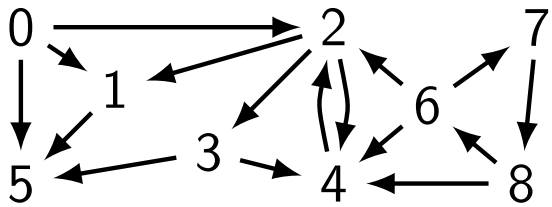
$\Rightarrow$ arborele de traversare în lățime:



1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x.

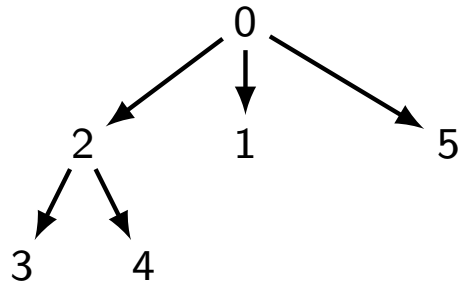
Exemplu ilustrat de traversare în lățime de la nodul 0

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

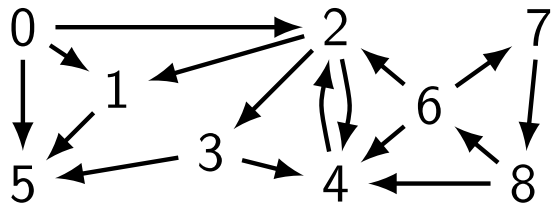
$\Rightarrow$ arborele de traversare în lățime:



1. Arborele de traversare cu rădăcina s conține toate nodurile x la care se poate ajunge din s (adică $s \rightsquigarrow x$)
2. $\forall x$: lista de noduri pe ramura de la s la x este un drum de lungime minimă la s la x.

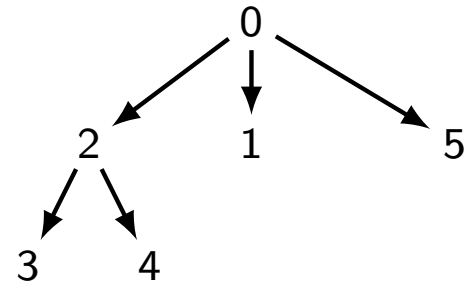
Exemplu ilustrat de traversare în lățime de la nodul 0

G:



$\texttt{adj}[0] = [2, 1, 5]$, $\texttt{adj}[1] = [5]$, $\texttt{adj}[2] = [3, 1, 4]$,
$\texttt{adj}[3] = [4, 5]$, $\texttt{adj}[4] = [2]$, $\texttt{adj}[5] = []$,
$\texttt{adj}[6] = [4, 2]$, $\texttt{adj}[7] = [8]$, $\texttt{adj}[8] = [6]$.

$\Rightarrow$ arborele de traversare în lățime:



Cele mai scurte căi găsite sunt $[0], [0, 2], [0, 2, 3], [0, 2, 4], [0, 1], [0, 5]$.

Ordini de traversare în adâncime

Traversarea în adâncime a tuturor nodurilor unui graf

- Produce o pădure de arbori de traversare în adâncime
- Definește trei ordini de traversare:

  Preordine: se adaugă nodurile într-o coadă înaintea apelului recursiv al lui `dfs()`. Avem $x <_{\text{pre}} y$ dacă x apare înaintea lui y în coadă.

  Postordine: se adaugă nodurile în o coadă de noduri după apelul recursiv al lui `dfs()`. Avem $x <_{\text{post}} y$ dacă x apare înaintea lui y în coadă.

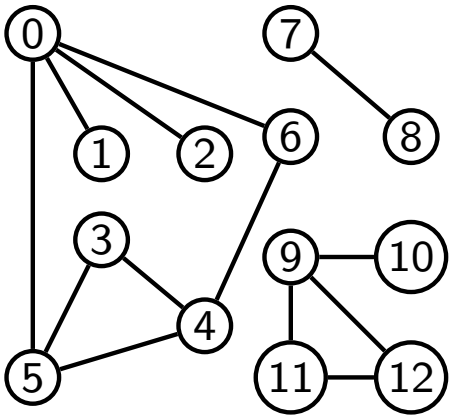
  Postordine inversă: Avem $x <_{\text{revpost}} y$ dacă $y <_{\text{post}} x$. Deci postordinea inversă se poate calcula punând nodurile în o stivă după apelul recursiv al lui `dfs()`. Avem $x <_{\text{revpost}} y$ dacă x apare deasupra lui y în stivă.

Ordini de traversare în adâncime

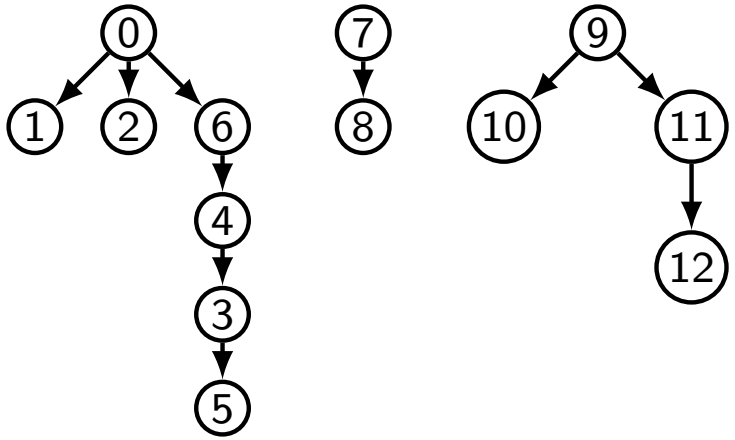
Exemplu ilustrat

$$\begin{aligned}
&\texttt{adj}[0] = [1, 2, 5, 6], && \texttt{adj}[1] = [0], && \texttt{adj}[2] = [0], && \texttt{adj}[3] = [4, 5],\\
&\texttt{adj}[4] = [3, 5, 6], && \texttt{adj}[5] = [0, 3, 4], && \texttt{adj}[6] = [0, 4], && \texttt{adj}[7] = [8],\\
&\texttt{adj}[8] = [7], && \texttt{adj}[9] = [10, 11, 12], && \texttt{adj}[10] = [9], && \texttt{adj}[11] = [9, 12],\\
&\texttt{adj}[11] = [9, 12], && \texttt{adj}[12] = [9, 11]. &&
\end{aligned}$$

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce

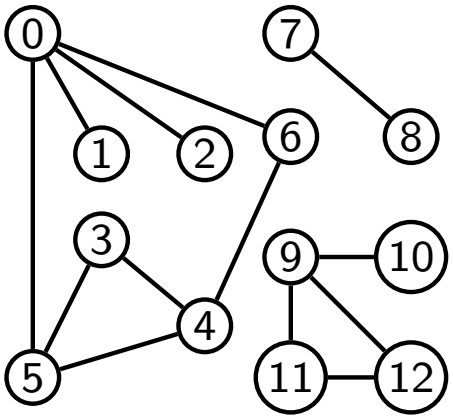


Ordini de traversare în adâncime

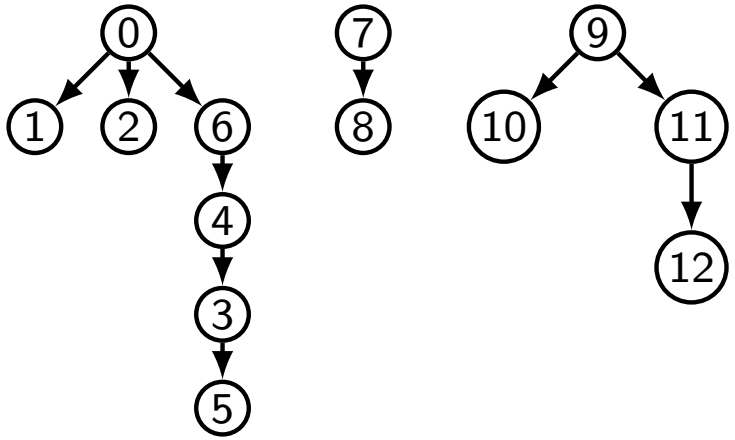
Exemplu ilustrat

$$\begin{aligned}
&\texttt{adj}[0] = [1, 2, 5, 6], && \texttt{adj}[1] = [0], && \texttt{adj}[2] = [0], && \texttt{adj}[3] = [4, 5],\\
&\texttt{adj}[4] = [3, 5, 6], && \texttt{adj}[5] = [0, 3, 4], && \texttt{adj}[6] = [0, 4], && \texttt{adj}[7] = [8],\\
&\texttt{adj}[8] = [7], && \texttt{adj}[9] = [10, 11, 12], && \texttt{adj}[10] = [9], && \texttt{adj}[11] = [9, 12],\\
&\texttt{adj}[11] = [9, 12], && \texttt{adj}[12] = [9, 11]. &&
\end{aligned}$$

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



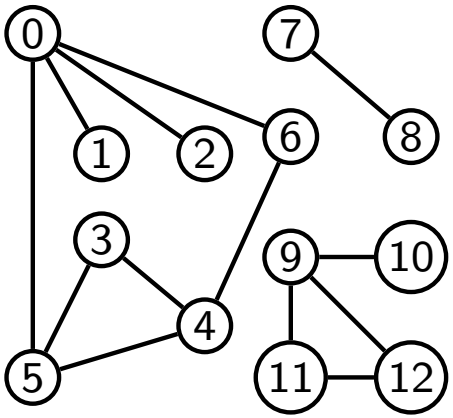
Preordine: $[0, 1, 2, 6, 4, 3, 5, 7, 8, 9, 10, 11, 12]$

Ordini de traversare în adâncime

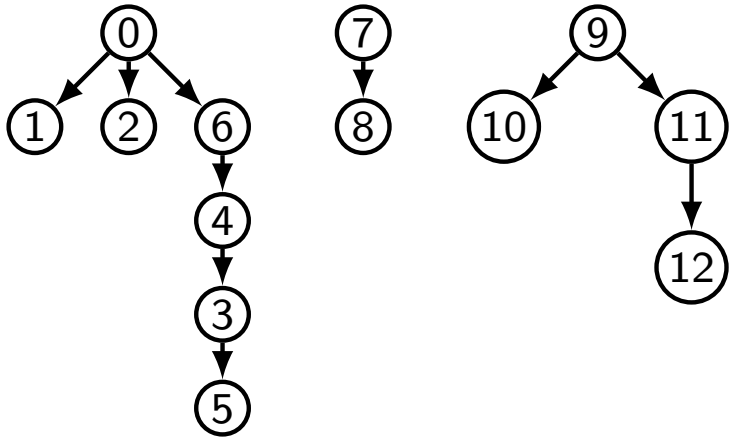
Exemplu ilustrat

$$\begin{aligned}
&\texttt{adj}[0] = [1, 2, 5, 6], && \texttt{adj}[1] = [0], && \texttt{adj}[2] = [0], && \texttt{adj}[3] = [4, 5],\\
&\texttt{adj}[4] = [3, 5, 6], && \texttt{adj}[5] = [0, 3, 4], && \texttt{adj}[6] = [0, 4], && \texttt{adj}[7] = [8],\\
&\texttt{adj}[8] = [7], && \texttt{adj}[9] = [10, 11, 12], && \texttt{adj}[10] = [9], && \texttt{adj}[11] = [9, 12],\\
&\texttt{adj}[11] = [9, 12], && \texttt{adj}[12] = [9, 11]. &&
\end{aligned}$$

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



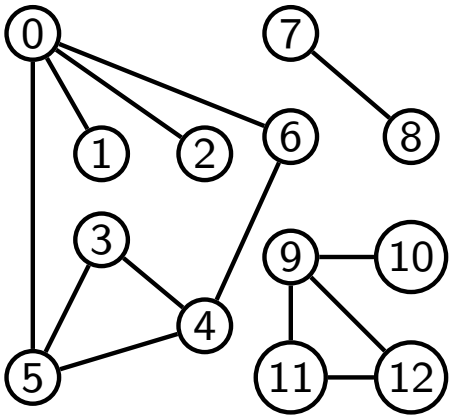
Preordine: $[0, 1, 2, 6, 4, 3, 5, 7, 8, 9, 10, 11, 12]$
Postordine: $[1, 2, 5, 3, 4, 6, 0, 8, 7, 10, 12, 11, 9]$

Ordini de traversare în adâncime

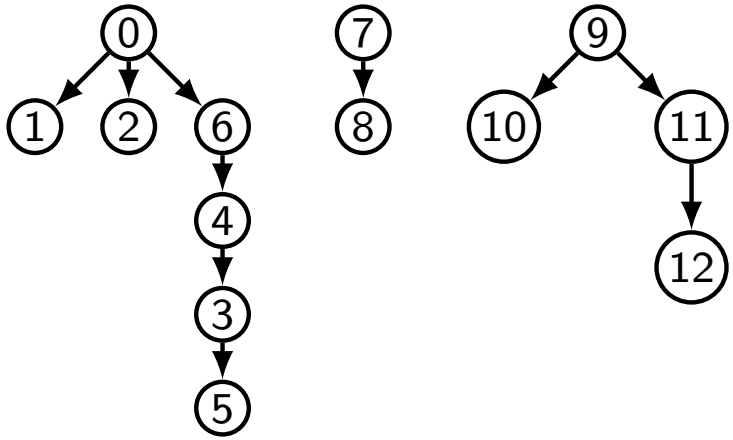
Exemplu ilustrat

$\texttt{adj}[0] = [1, 2, 5, 6]$, $\texttt{adj}[1] = [0]$, $\texttt{adj}[2] = [0]$, $\texttt{adj}[3] = [4, 5]$,
$\texttt{adj}[4] = [3, 5, 6]$, $\texttt{adj}[5] = [0, 3, 4]$, $\texttt{adj}[6] = [0, 4]$, $\texttt{adj}[7] = [8]$,
$\texttt{adj}[8] = [7]$, $\texttt{adj}[9] = [10, 11, 12]$, $\texttt{adj}[10] = [9]$, $\texttt{adj}[11] = [9, 12]$,
$\texttt{adj}[11] = [9, 12]$, $\texttt{adj}[12] = [9, 11]$.

Traversarea în adâncime a tuturor nodurilor grafului neorientat



produce



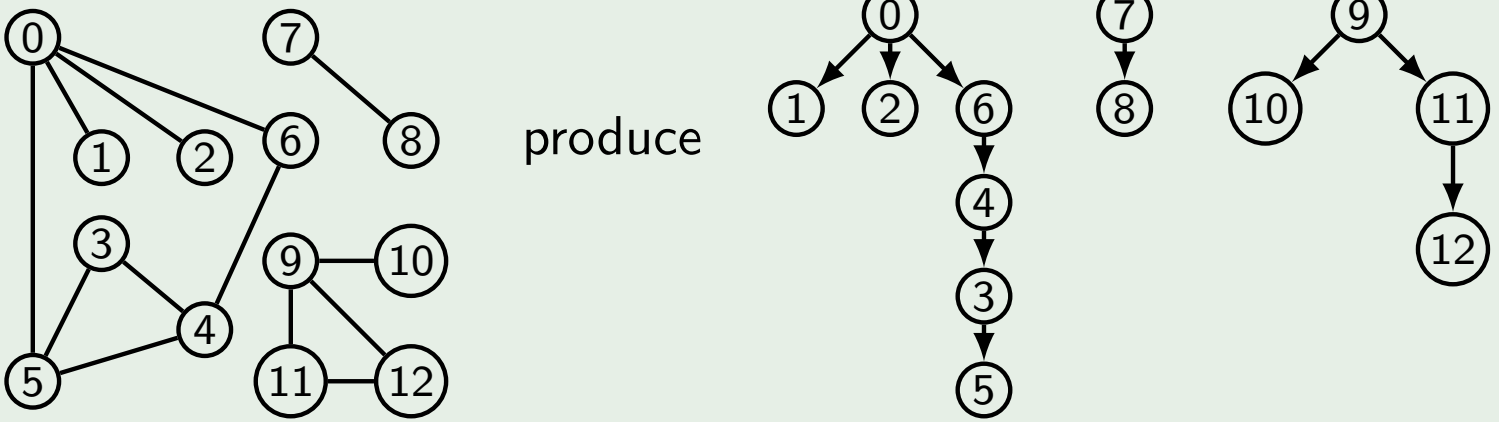
Preordine: $[0, 1, 2, 6, 4, 3, 5, 7, 8, 9, 10, 11, 12]$
Postordine: $[1, 2, 5, 3, 4, 6, 0, 8, 7, 10, 12, 11, 9]$
Postordine inversă: $[9, 11, 12, 10, 7, 8, 0, 6, 4, 3, 5, 2, 1]$

public	class CC	
	`CC(Graph G)`	Constructor pentru componentele conexe ale grafului neorientat `G`
`boolean`	`connected(int i,int j)`	Există drum de la `i` la `j`?
`int`	`count()`	Numărul de componente conexe
`int`	`id(int v)`	Identificatorul de componentă al nodului `v` (între 0 şi `count()-1`)

```
public class CC {
   private boolean[] vizitat;
   private int[] id;
   private int count;
   public CC(Graph G) {
      vizitat = new boolean[G.V()]; id = new int[G.V()];
      for (int s=0; s<G.V(); s++) if (!vizitat(s)) { dfs(G,s); count++; }
   }
   private void dfs(Graph G, int i) {
        vizitat[i] = true; id[i] = count;
        for (int j : G.adj(i)) if (!vizitat[j]) dfs(G,j);
   }
   ...
}
```

Exemplu

Traversarea în adâncime a tuturor nodurilor grafului neorientat



- Nodurile $x \in \{0, 1, 2, 6, 4, 3, 5\}$ din primul arbore fac parte din prima componentă conexă; pentru ele setăm $\texttt{id}[x] = 0$.
- Nodurile $x \in \{7, 8\}$ din al doilea arbore fac parte din a doua componentă conexă; pentru ele setăm $\texttt{id}[x] = 1$.
- Nodurile $x \in \{9, 10, 11, 12\}$ din al treilea arbore fac parte din a treia componentă conexă; pentru ele setăm $\texttt{id}[x] = 2$.

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \to v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \to v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \to v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \to v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \to v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \to v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Observații

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \to v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \to v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \to v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Observații

1. G are un ciclu dacă și numai dacă are o muchie de întoarcere.

Traversarea în adâncime a tuturor nodurilor unui digraf produce o pădure de arbori de căutare în adâncime.

Arcele care nu apar ca muchii în arbori sunt de 3 feluri:

Muchii de întoarcere: sunt arcele $u \to v$ de la un nod u la un predecesor de-al lui în un arbore de căutare în adâncime.

Muchii de salt înainte: sunt arcele $u \to v$ de la un nod u la un succesor de-al lui în un arbore de căutare în adâncime.

Muchii transversale: sunt arce $u \to v$ de două feluri: (1) între noduri pe ramuri diferite din același arbore, sau (2) de la un nod u la un nod v într-un arbore construit anterior.

Observații

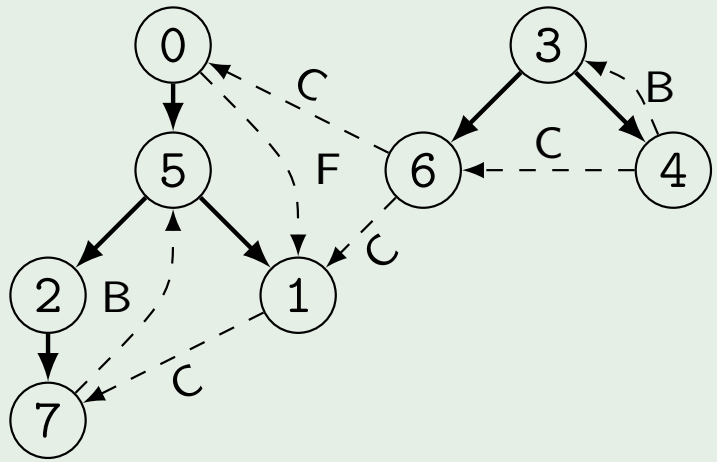
1. G are un ciclu dacă și numai dacă are o muchie de întoarcere.
2. Vom folosi abrevierea DAG (engl. *directed acyclic graph*) pentru un digraf fără cicluri.

Exemplu

Traversarea în adâncime a digrafului reprezentat cu

$$\texttt{adj}[0] = [5, 1], \quad \texttt{adj}[1] = [7], \quad \texttt{adj}[2] = [7], \quad \texttt{adj}[3] = [6, 4],$$
$$\texttt{adj}[4] = [3, 6], \quad \texttt{adj}[5] = [2, 1], \quad \texttt{adj}[6] = [0, 1], \quad \texttt{adj}[7] = [5].$$

produce 2 arbori de căutare:



Muchiile de întoarcere au fost etichetate cu B, cele de salt înainte cu F, și cele transversale cu C.

Un API java pentru detecția ciclurilor în digrafuri:

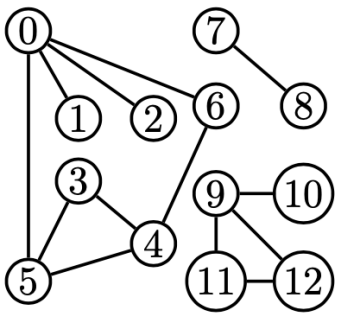
`public`	`class DCycle`	
	`DCycle(Digraph G)`	Constructor de detecţie a ciclurilor
`boolean`	`hasCycle(E e)`	Are `G` un ciclu?
`Iterable<Integer>`	`cycle()`	Nodurile din un ciclu (dacă există unul)

- Detalii de implementare se găsesc în materialul extins de curs.

Traversarea în adâncime a tuturor nodurilor unui graf G produce o pădure de arbori de căutare în adâncime.

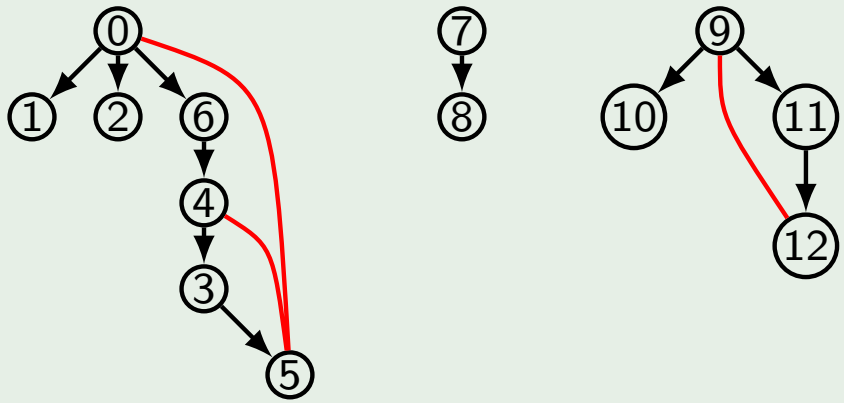
- Toate muchiile lui G care nu apar ca muchii în arbori sunt între un nod x și un predecesor indirect al lui x într-un arbore.

Exemplu



$\texttt{adj}[0] = [1, 2, 5, 6], \texttt{adj}[1] = [0], \texttt{adj}[2] = [0],$
$\texttt{adj}[3] = [4, 5], \texttt{adj}[4] = [3, 5, 6], \texttt{adj}[5] = [0, 3, 4],$
$\texttt{adj}[6] = [0, 4], \texttt{adj}[7] = [8], \texttt{adj}[8] = [7],$
$\texttt{adj}[9] = [10, 11, 12], \texttt{adj}[10] = [9], \texttt{adj}[11] = [9, 12],$
$\texttt{adj}[12] = [9, 11].$

Traversarea în adâncime produce o pădure cu 3 arbori



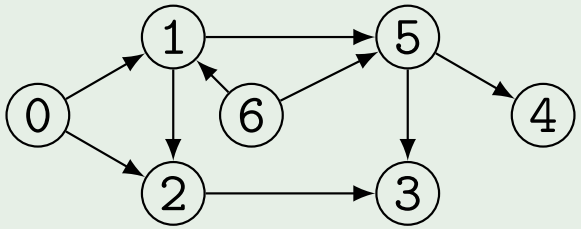
- În literatură, arborii de traversare în adâncime împreună cu muchiile de întoarcere se numesc palmieri (engl. *palm trees*).
- Un graf neorientat are un ciclu dacă și numai dacă există o muchie de întoarcere într-un palmier.
- Un API java de detecție a ciclurilor în grafuri neorientate este descris în materialul extins de curs.

`public`	`class CC`	
	`...`	
`boolean`	`hasCycle(int c)`	Există ciclu în componenta cu identificatorul `c`?
`Iterable<Integer>`	`cycle(int c)`	Nodurile din un ciclu al componentei conexe `c` (dacă există unul)

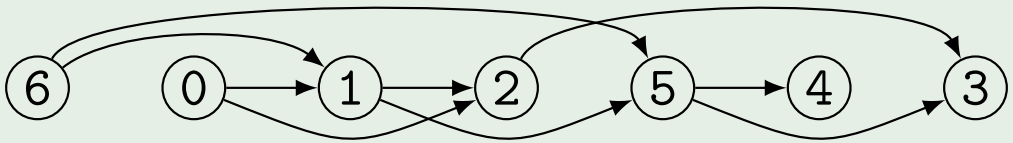
Sortare topologică a unui DAG (V, E) = enumerare $[x_1, x_2, \ldots, x_n]$ a tuturor nodurilor din V astfel încât $\forall e \in E : e = (x_i \to x_j)$ cu $i < j$.

$\Rightarrow$ dacă redesenăm graful cu nodurile $x_1, x_2, \ldots, x_n$ ordonate de la stânga la dreapta pe o dreaptă imaginară, atunci toate arcele sunt orientate de la stânga la dreapta.

Exemplu (Un DAG cu o sortare topologică)



are sortarea topologică $[6, 0, 1, 2, 5, 4, 3]$:

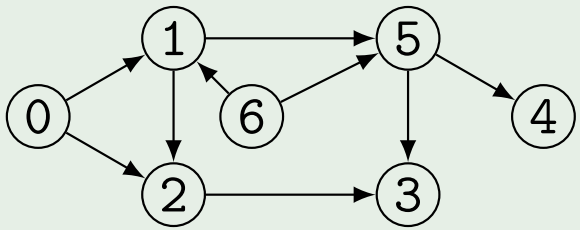


Observație

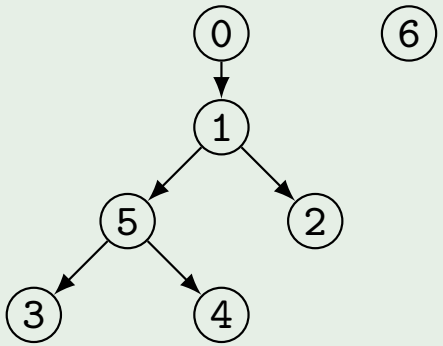
Enumerarea nodurilor în postordine inversă produsă de traversarea în adâncime este o sortare topologică.

Exemplu

Arborii de traversare în adâncime produși de traversarea în adâncime a digrafului



sunt



Postordine: $[3, 4, 5, 2, 1, 0, 6]$

Postordine inversă: $[6, 0, 1, 2, 5, 4, 3]$

Un API java pentru sortarea topologică a unui DAG:

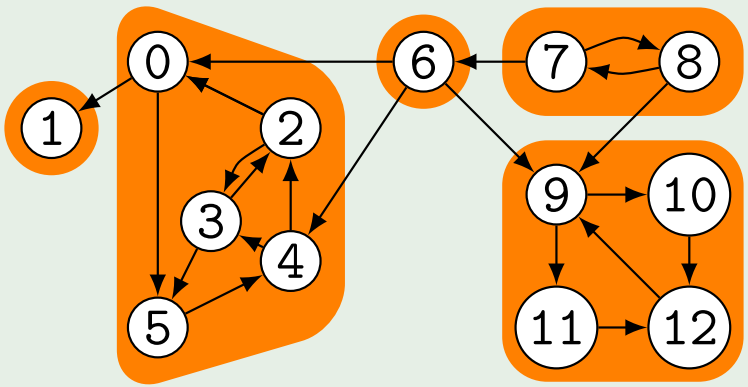
public	class Topologic	
	`Topologic(Digraph G)`	Crează un obiect pentru sortarea topologică a nodurilor unui digraf `G`
`boolean`	`isDag()`	Este aciclic?
`Iterable<Integer>`	`order()`	Nodurile sortate topologic.

Fie $G = (V, E)$ un digraf. O componentă tare conexă a lui G este o clasă de echivalență a relației de echivalență

$x \sim_{sc} y$ dacă și numai dacă $x \rightsquigarrow y$ și $y \rightsquigarrow x$.

Exemplu (Componentele tare conexe ale unui digraf)

$\texttt{adj}[0] = [1, 5]$, $\texttt{adj}[1] = []$, $\texttt{adj}[2] = [0, 3]$, $\texttt{adj}[3] = [2, 5]$,
$\texttt{adj}[4] = [2, 3]$, $\texttt{adj}[5] = [4]$, $\texttt{adj}[6] = [0, 4, 9]$, $\texttt{adj}[7] = [6, 8]$,
$\texttt{adj}[8] = [7, 9]$, $\texttt{adj}[9] = [10, 11]$, $\texttt{adj}[10] = [12]$, $\texttt{adj}[11] = [12]$,
$\texttt{adj}[12] = [9]$.



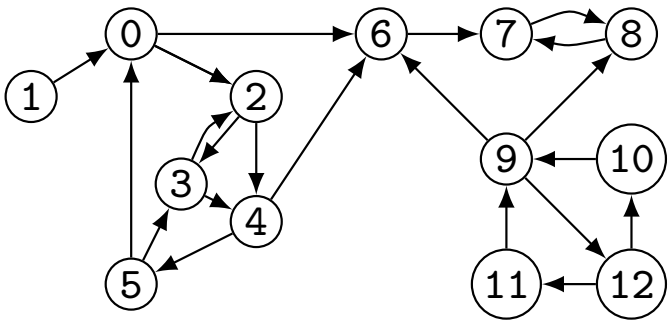
$\{1\}$
$\{0, 2, 3, 4, 5\}$
$\{6\}$
$\{7, 8\}$
$\{9, 10, 11, 12\}$

Algoritmul lui Kosaraju

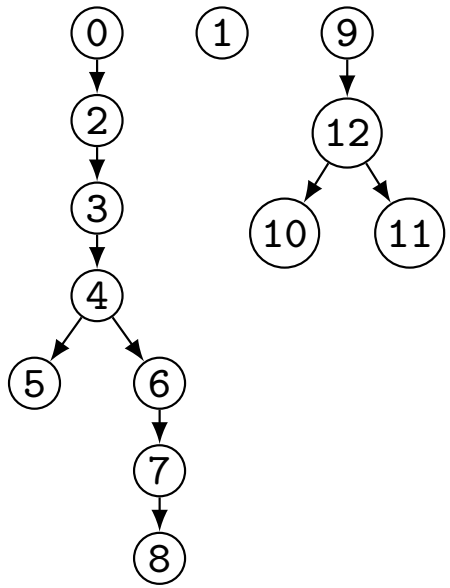
1. Calculează postordinea inversă a nodurilor din graful invers G^r.
2. Traversează toate nodurile lui G în adâncime, însă în ordinea calculată în pasul 1.
3. Toate nodurile din un arbore de căutare în adâncime calculat în felul acesta formează o componentă tare conexă a lui G.

Considerăm digraful G din exemplul precedent. Digraful invers G^r are reprezentarea

$$\texttt{adj}[0] = [2, 6], \quad \texttt{adj}[1] = [0], \quad \texttt{adj}[2] = [3, 4], \quad \texttt{adj}[3] = [2, 4],$$
$$\texttt{adj}[4] = [5, 6], \quad \texttt{adj}[5] = [0, 3], \quad \texttt{adj}[6] = [7], \quad \texttt{adj}[7] = [8],$$
$$\texttt{adj}[8] = [7], \quad \texttt{adj}[9] = [6, 8, 12], \quad \texttt{adj}[10] = [9], \quad \texttt{adj}[11] = [9],$$
$$\texttt{adj}[12] = [10, 11]$$



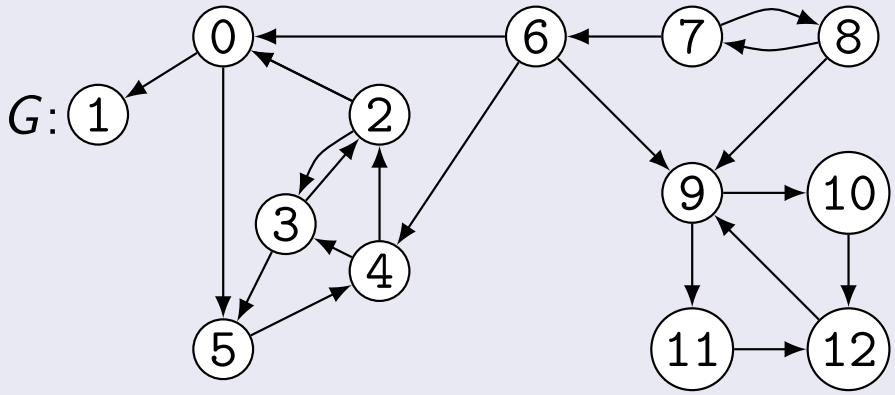
iar traversarea în adâncime produce arborii



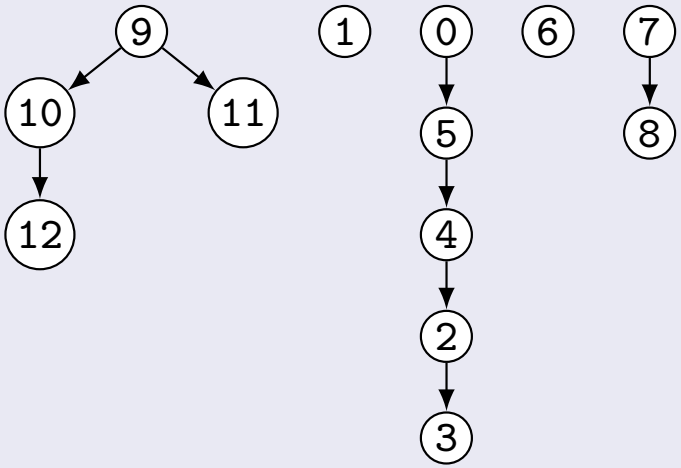
$\Rightarrow$ postordinea inversă $[9, 12, 11, 10, 1, 0, 2, 3, 4, 6, 7, 8, 5]$

Algoritmul lui Kosaraju ilustrat (continuare)

Traversarea în adâncime a lui

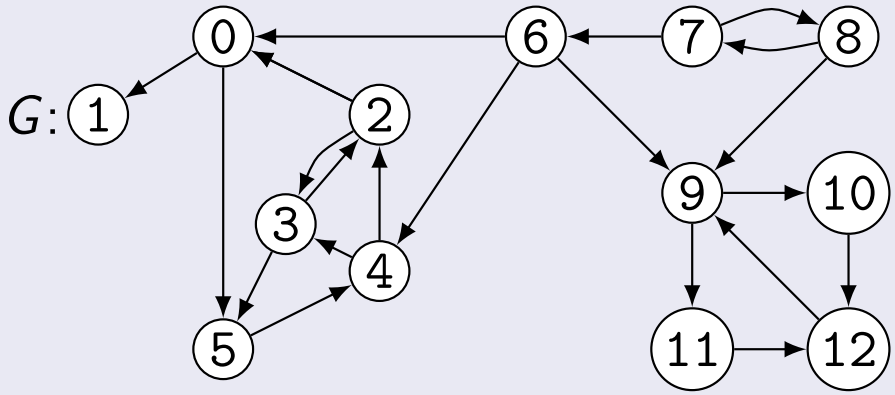


în ordinea $[9, 12, 11, 10, 1, 0, 2, 3, 4, 6, 7, 8, 5]$ produce pădurea de arbori

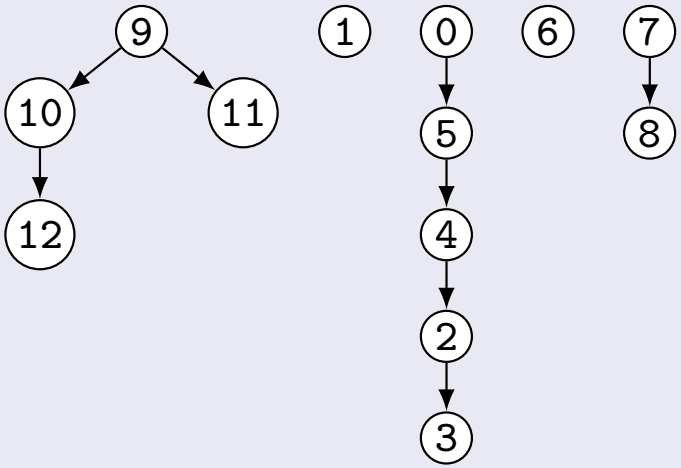


Algoritmul lui Kosaraju ilustrat (continuare)

Traversarea în adâncime a lui



în ordinea $[9, 12, 11, 10, 1, 0, 2, 3, 4, 6, 7, 8, 5]$ produce pădurea de arbori



Rezultă componentele tare conexe
$\{9, 10, 11, 12\}$
$\{1\}$
$\{0, 5, 4, 2, 3\}$
$\{6\}$
$\{7, 8\}$

Alte aplicații ale traversării în adâncime

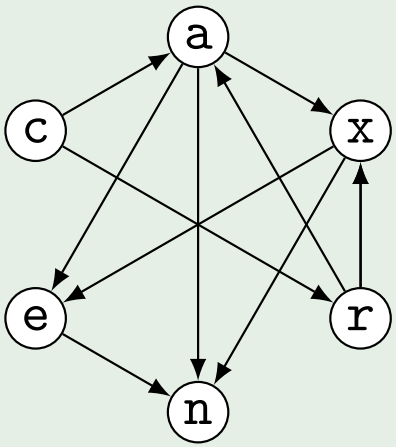
- Determinarea de drumuri elementare de lungime maximă de la un nod sursă în un DAG.
- ...

Exemplu

Listele de adiacență

$$\texttt{adj[a]} = [\texttt{e}, \texttt{x}, \texttt{n}], \quad \texttt{adj[c]} = [\texttt{a}, \texttt{r}], \quad \texttt{adj[x]} = [\texttt{e}, \texttt{n}],$$
$$\texttt{adj[r]} = [\texttt{a}, \texttt{x}], \quad \texttt{adj[e]} = [\texttt{n}], \quad \texttt{adj[n]} = []$$

reprezintă DAG-ul



Alte aplicații ale traversării în adâncime

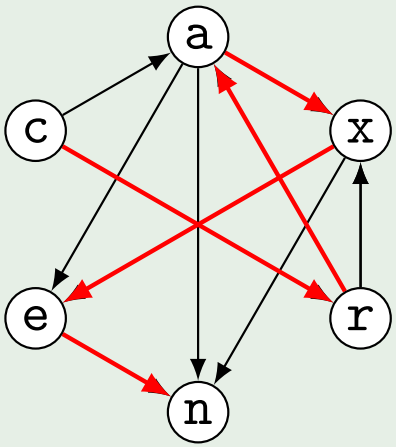
- Determinarea de drumuri elementare de lungime maximă de la un nod sursă în un DAG.
- ...

Exemplu

Listele de adiacență

$$\texttt{adj[a]} = [\texttt{e}, \texttt{x}, \texttt{n}], \quad \texttt{adj[c]} = [\texttt{a}, \texttt{r}], \quad \texttt{adj[x]} = [\texttt{e}, \texttt{n}],$$
$$\texttt{adj[r]} = [\texttt{a}, \texttt{x}], \quad \texttt{adj[e]} = [\texttt{n}], \quad \texttt{adj[n]} = []$$

reprezintă DAG-ul



Drumuri elementare de lungime maximă de la sursa c:

[c], [c, r],
[c, r, a], [c, r, a, x]
[c, r, a, x, e]
[c, r, a, x, e, n]