M1 Informatique – Graphes
Arbres et arborescences

Irena.Rusu@univ-nantes.fr

LS2N, bât. 34, bureau 303

tél. 02.51.12.58.16

- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

$G=(S,A)$ graphe non-orienté

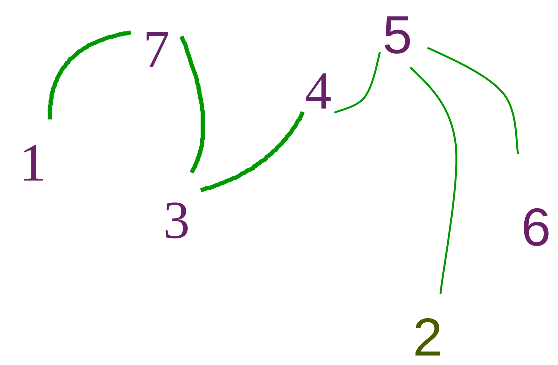
G est un arbre s'il est connexe et sans cycles.

Proposition. Un arbre à n sommets ($n \geq 2$) a au moins deux sommets de degré 1.

Theorème. Soit G un graphe non-orienté à n sommets ($n \geq 1$). Les propriétés suivantes sont équivalentes :

G est connexe et sans cycles

G est sans cycles et a n-1 arêtes



G est connexe et a n-1 arêtes

G est sans cycles et si on ajoute une arête on forme un cycle

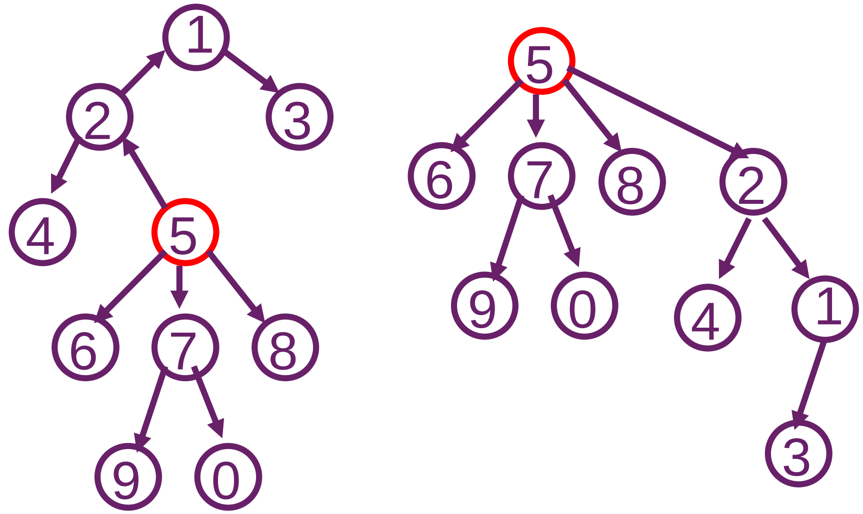
G est connexe et si on lui retire une arête il n'est plus connexe

pour tous s,s' de S ($s \neq s'$), il existe un et un seul chemin joignant s et s'.

- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

G=(S,A) graphe orienté

G est une arborescence si son graphe support est un arbre et tous ses sommets ont le degré entrant 1 sauf un, appelé racine, pour lequel le degré entrant est 0.



Proposition. Si *G* est un arbre fini, pour tout sommet s de *G* il existe une orientation qui en fait une arborescence de racine s.

$G = (S, A)$

Parcourir G = visite de tous les sommets
et/ou de tous les arcs

Algorithme de base pour
- recherche de cycles
- tri topologique
- recherche des composantes connexes
- actions sur les sommets (coloration, ...)
  sur les arcs (valuation, ...)

- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

G=(S,A) un graphe, s un sommet de *G*

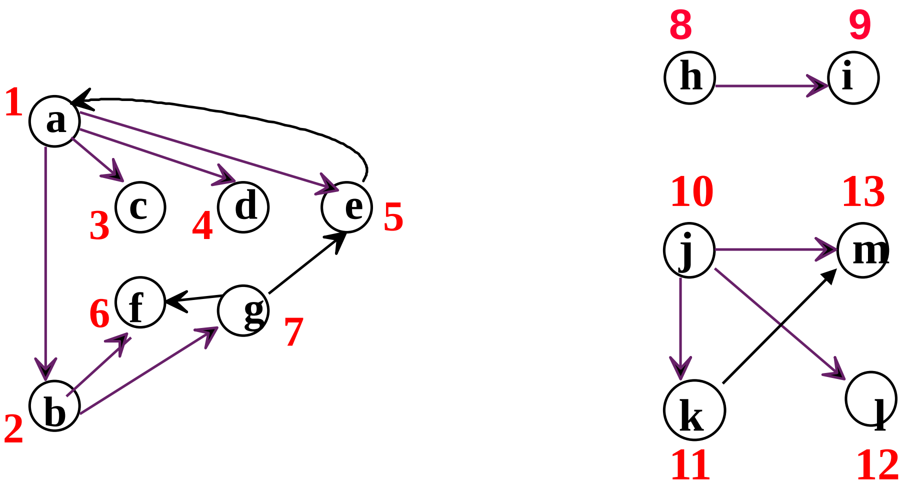
Caractéristiques du parcours en largeur :

- emprunte tous les arcs de *G* pour découvrir progressivement les sommets accessibles depuis *s*

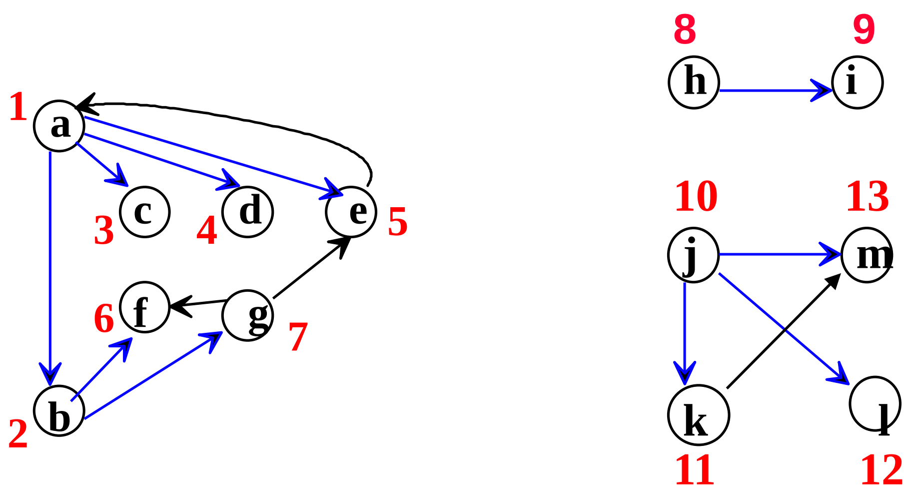
- construit une arborescence en largeur ayant la racine *s* et contenant tous les sommets accessibles depuis *s*

- pour tout *v* accessible depuis *s*, le chemin de *s* à *v* dans l'arborescence est le plus court chemin de *s* à *v* dans *G*

- découvre tous les sommets à distance k de s avant de découvrir ceux à distance k+1.



Ordre du parcours : a b c d e f g h i j k l m



Ordre du parcours : a b c d e f g h i j k l m

Exploration du graphe entier

```
pour chaque sommet s de G  faire
        visité[s] ← faux ;
pour chaque sommet s de G  faire
        si non visité [s] alors Larg (s) ;
```

Procédure Larg (*s* sommet de *G*) ;

```
début
        File ← Ajouter (File-vide, s) ;
        tant que non  Vide (File)  faire {
                s' ← Premier (File) ; File ← Enlever (File, s') ;
                si non visité [s'] alors {
                        visité [s'] ← vrai ;
                        pour  t ← premier au dernier successeur de s'
faire
                                si non visité [ t ] alors
                                        File ← Ajouter (File, t) ;
                }       }
fin
```

T (« pour chaque sommet ») = $O(|S|)$

T(« tant que non Vide (File) faire ») n'est significatif que lorsque pour est exécuté.

Matrice d'adjacence

T (« pour $t \leftarrow$ premier au dernier successeur de s' ») =

$T\Big($« pour chaque sommet t
tel que $M[s,t] = 1$ »$\Big)$ $= O(|S|)$

$\Rightarrow$ parcours en $O(|S|^2)$

Listes de successeurs

T (« pour $t \leftarrow$ premier au dernier successeur de s' ») = $O(|A(s)|)$

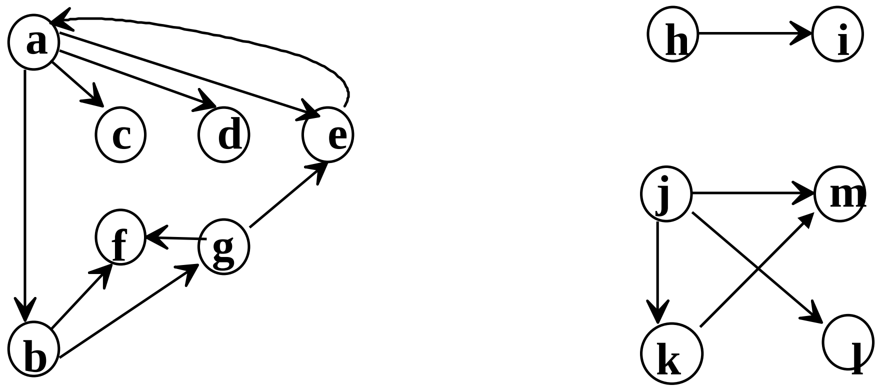
$\Rightarrow$parcours en $O(|S| + |A|)$

- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

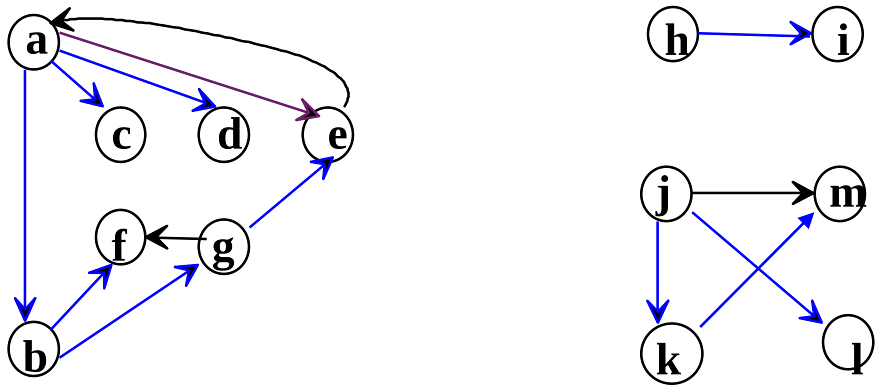
G=(S,A) graphe, *s* sommet

Caractéristiques du PP:

- Emprunte les arcs de *G* pour découvrir progressivement tous les sommets de *G* accessibles depuis *s*
- Construit une arborescence ayant la racine *s* et contenant tous les sommets accessibles depuis *s*
- Explore d'abord les candidats accessibles depuis les sommets déjà visités les plus profonds



Ordre du parcours : a b f g e c d h i j k m l



Ordre du parcours : a b f g e c d h i j k m l

Exploration du graphe entier

```
Exploration du graphe entier
    pour chaque sommet  s de G  faire
        visité[s] ← faux ;
    pour chaque sommet s de G  faire
        si non visité [s] alors Prof (s) ;

procédure  Prof (s sommet de G) ;
début
    visité[s] ← vrai ;
    pour chaque t successeur de s faire
        si non  visité[ t ]  alors Prof ( t ) ;
fin
```

T (« pour chaque sommet ») = $O(|S|)$

Matrice d'adjacence

T (« pour chaque *t* successeur de *s* ») =

$T\big($« pour chaque sommet *t*
tel que $M[s,t] = 1$ » $\big)$ = $O(|S|)$

$\Rightarrow$ parcours en $O(|S|^2)$

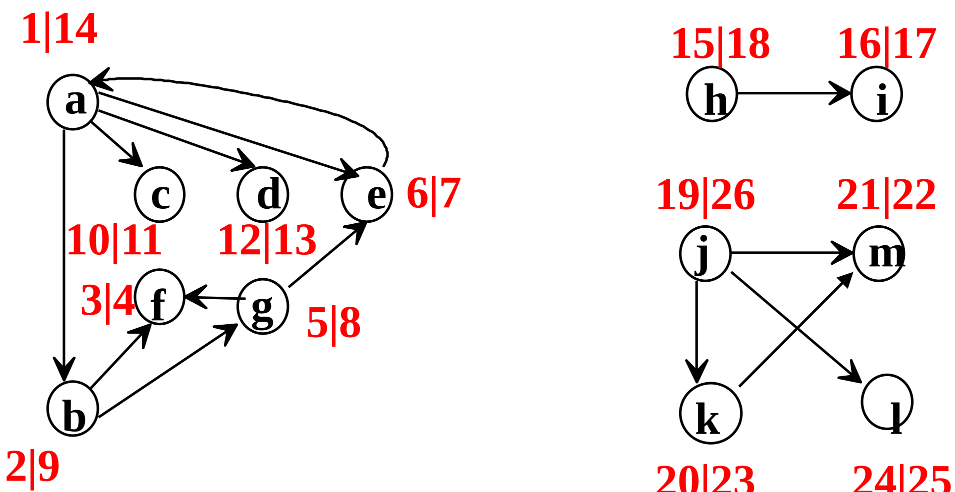
Listes de successeurs

T (« pour chaque *t* successeur de *s* ») = $O(|A(s)|)$

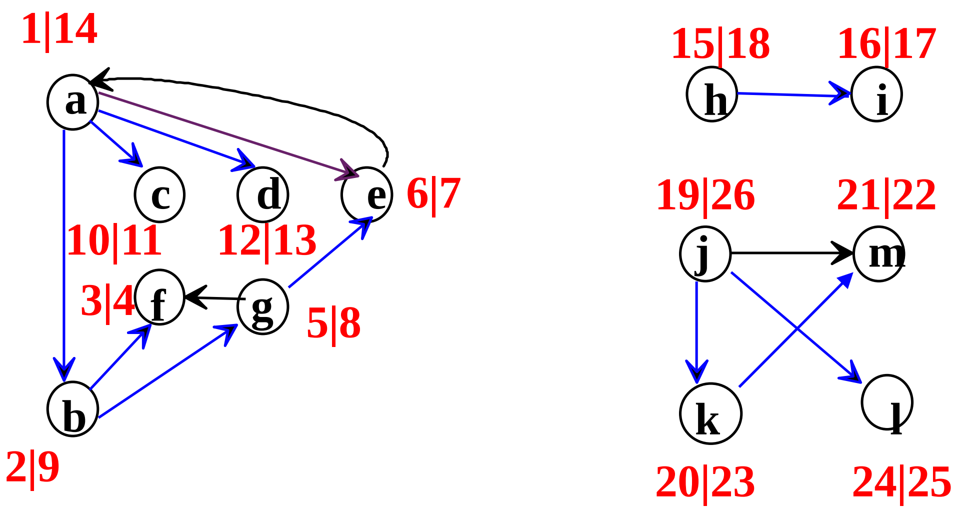
$\Rightarrow$parcours en $O(|S| + |A|)$

```
procedure ProfNum (G graphe)
début
        pour chaque sommet s de G  faire
                d [s]  ←  0 ; f [s] ←  0 ;
        nb ← 0 ;
        pour chaque sommet s de G  faire
                si d [s] = 0  alors  Num (s)
fin

procédure  Num (s sommet de G) ;
début
        nb ← nb + 1 ; d [s] ← nb ;
        pour chaque t successeur de  s faire
                si d [ t ] = 0  alors  Num ( t ) ;
         nb ← nb+1 ; f [s] ← nb

fin
```

temps = O (|*S*| + |*A*|)
avec listes de successeurs

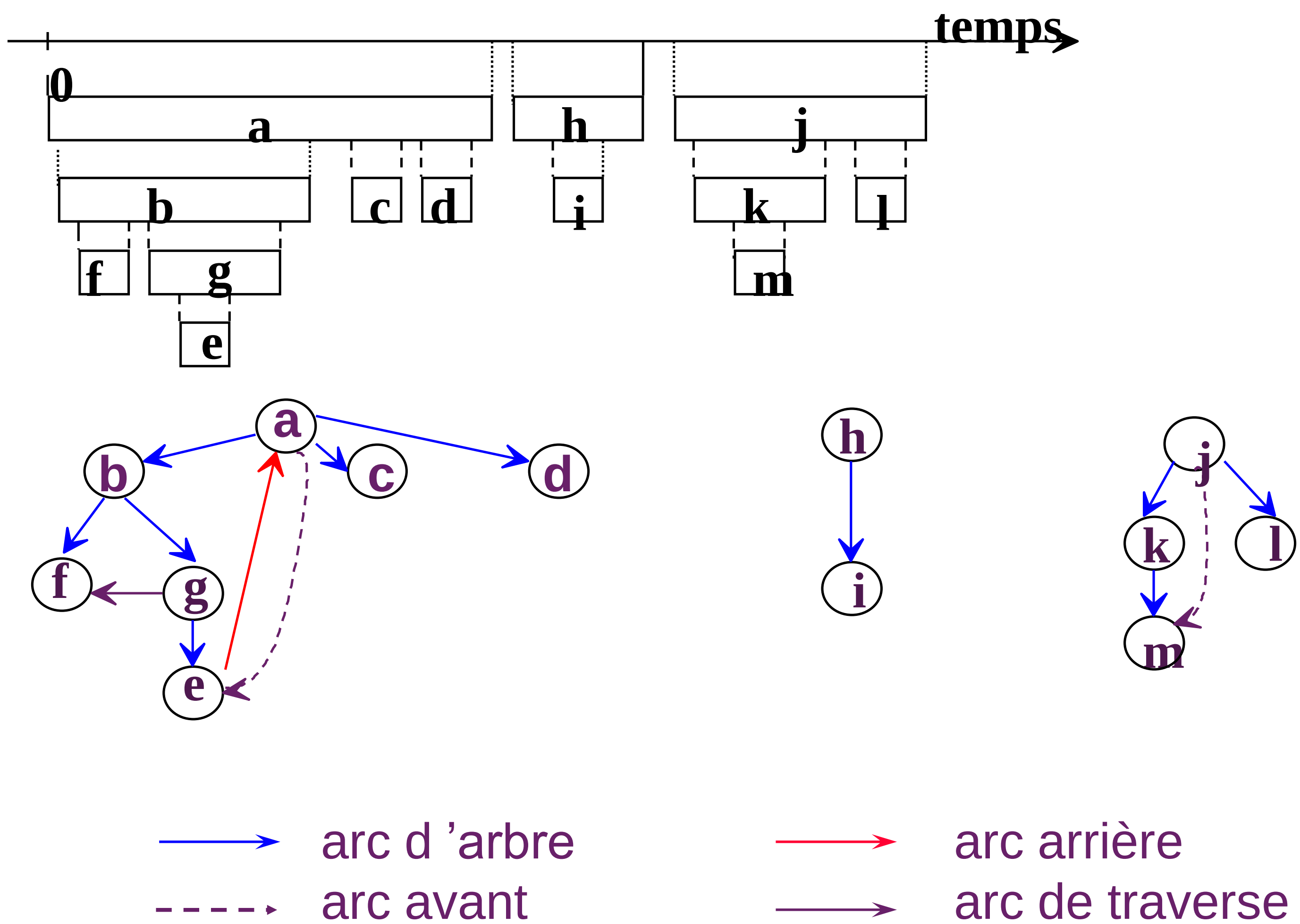


Ordre du parcours : a b f g e c d h i j k m l



Ordre du parcours : a b f g e c d h i j k m l

Procédure Prof (*s* sommet de *G*) ;   /* version itérative */
début
    Pile ← Empiler (Pile-vide, *s*) ;
    tant que non vide (Pile) faire {
        s' ← Elt (sommet (Pile)) ; Pile ← Dépiler (Pile) ;
        si non visité [*s'*]   alors {
            visité [*s'*] ← vrai ;
            pour *t* ← dernier au premier successeur de *s'* faire
                si non visité [*t*] alors
                    Pile ← Ajouter (Pile, *t*) ;
        }
    }
fin


temps
0
a
h
j
b
c
d
i
k
l
f
g
m
e
a
b
c
d
f
g
e
h
i
j
k
l
m
arc d 'arbre
arc arrière
arc avant
arc de traverse

- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

Proposition

G possède un circuit ssi il existe un arc arrière dans un parcours en profondeur de G

$d(s)$: date de début d'exécution de Prof(s)
$f(s)$: date de fin d'exécution de Prof(s)

(s,t) arc de G est un

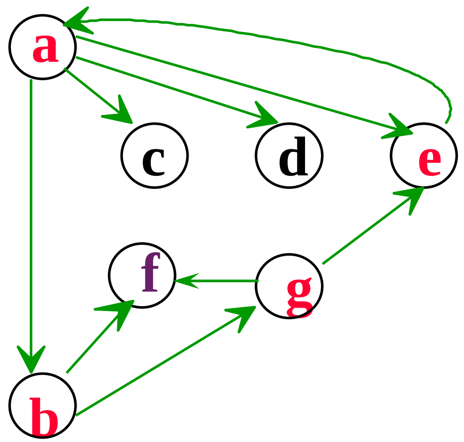
- arc d'arbre ou arc avant ssi $d(s) < d(t) < f(t) < f(s)$
- arc arrière ssi $d(t) < d(s) < f(s) < f(t)$
- arc de traverse ssi $f(t) < d(s)$

Au cours d'une exploration :

état [*s*] = noir *s* non visité

état [*s*] = rouge *s* en cours de visite (dans Pile ou File)

état [*s*] = bleu *s* déjà visité

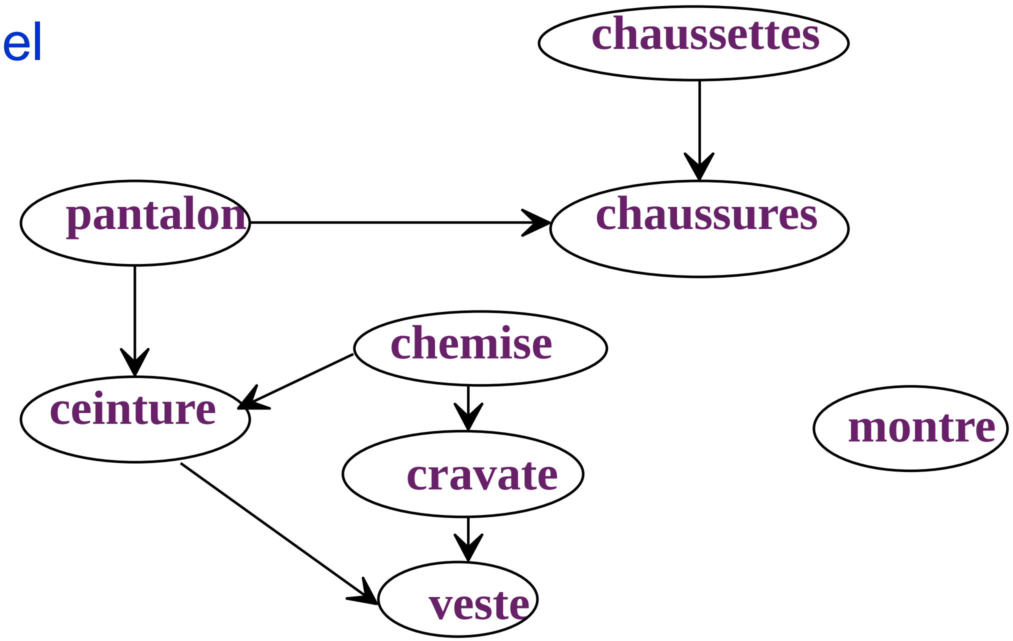


Pendant la visite du sommet e, on détecte un cycle passant par l 'arc (e, a) car a est aussi en cours de visite

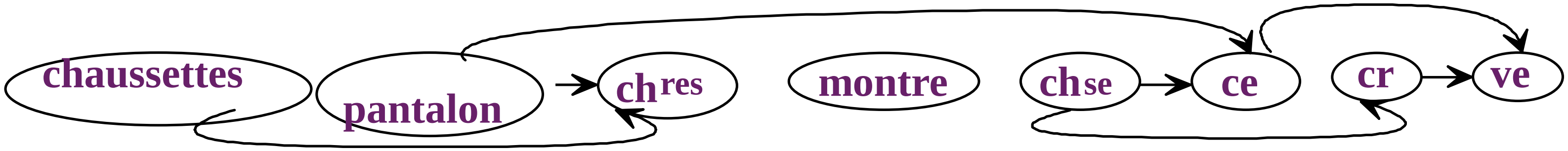
- Arbres
- Arborescences et parcours
  - Parcours en largeur
  - Parcours en profondeur
- Détection d’un cycle/circuit
- Tri topologique

Prolongement d'un ordre partiel
en un ordre total

graphe sans circuit, *i.e.*
pas d 'arc arrière



Ordre possible d'habillage:



Ordre final : ordre décroissant des $f(s)$ dans un parcours en profondeur
(dans l'exemple, parcours : chemise, cravate, veste, ceinture, montre, pantalon, chaussures, chaussettes)

```
fonction Tri-topologique (G graphe acyclique) : liste ;
début
        pour chaque sommet s de G faire
                visité [s] ← faux ;
        L ← liste-vide ;
        pour chaque sommet s de G faire
                si non visité [s] alors Topo (s) ;
        retour (L) ;
fin
```

```
procédure Topo (s sommet de G) ;
début
        visité [s] ← vrai ;
        pour chaque t successeur de s  faire
                si non  visité [t] alors Topo (t) ;
        Ajouter s en tête de L ;
fin
```

Temps d 'exécution : $O(|S| + |A|)$

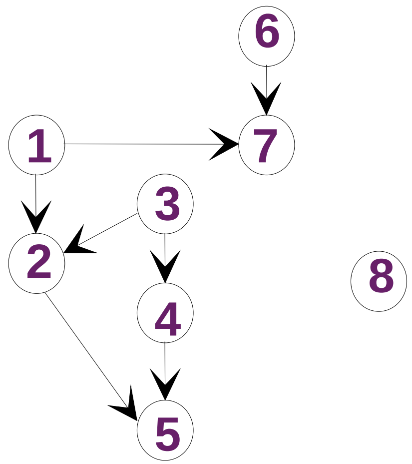
	1	2	3	4	5	6	7	8
Nb-préd	0	2	0	1	2	0	2	0

Sommets à traiter : 1 3 6 8
(sans prédécesseur)

Après traitement de 1 :

	1	2	3	4	5	6	7	8
Nb-Préd	-	1	0	1	2	0	1	0

Sommets à traiter : 3 6 8



```
Fonction Tri-topologique2 (G graphe acyclique) : liste ;
début
        F ← File-vide ;
        tant que G non vide faire
                si  tous les sommets ont un prédécesseur alors
                        « G contient un circuit » ;
                sinon {
                        s ← un sommet sans prédécesseur ;
                        G ← G diminué de s et des arcs d'origine s ;
                        F ← Ajouter (F, s) ;
                }
        retour (F) ;
fin
```

Temps d 'exécution : $O(|S| + |A|)$

- Les arbres et les arborescences sont des graphes comme les autres

- Mais attention à la représentation : celle spécifique aux arbres/arborescences (par pointeurs) est plus efficace pour ces classes de graphes et doit être prioritaire

- Les parcours de graphes généralisent les parcours d'arbres, et sont très utiles en algorithmique des graphes → les comprendre est essentiel

- Tous les algorithmes (de graphes ou non) ont besoin d'une preuve. Ceux que nous voyons ont été prouvés. Nous n'en donnons que les idées intuitives.