ALGORITHMIQUE

Gregory KUCHEROV

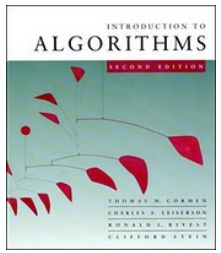
Gregory.Kucherov@univ-mlv.fr

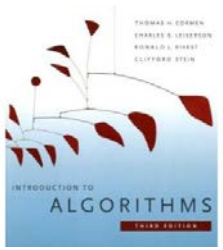
Basé sur les transparents de Maxime Crochemore

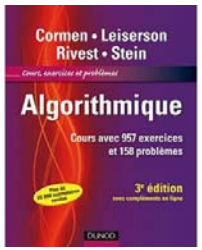
Université de Marne-la-Vallée

Plan du cours (préliminaire)

- **Graphes**
  - Représentations, explorations
  - Clôture transitive, plus courts chemins
  - Arbres couvrants, flots
- **Arbres de recherche**
  - Arbres rouges-noirs
- **Algorithmique du texte**
  - Reconnaissance de motifs
  - Recherche de mots
  - Alignements



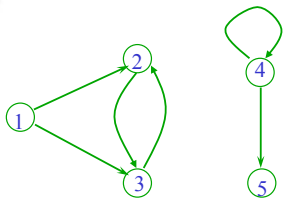




CLRS = Cormen & Leiserson & Rivest &Stein

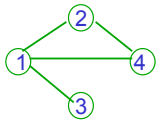
Graphes

Graphe (orienté) $G = (S, A)$
S ensemble fini des sommets
$A \subseteq S \times S$ ensemble des arcs,
i.e., relation sur S



$S = \{ 1, 2, 3, 4, 5 \}$
$A = \{ (1, 2), (1, 3), (2, 3), (3, 2), (4, 4), (4, 5) \}$

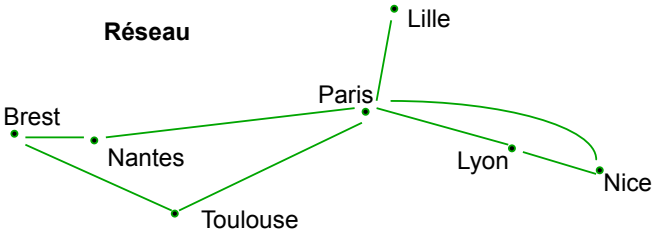
Graphe non orienté $G = (S, A)$
A ensemble des arêtes,
relation symétrique



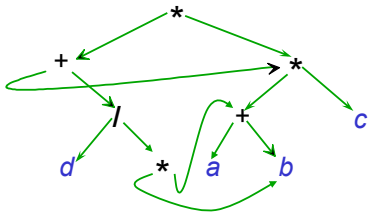
$S = \{ 1, 2, 3, 4 \}$
$A = \{ \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 4\} \}$

UMLV ©

Réseau



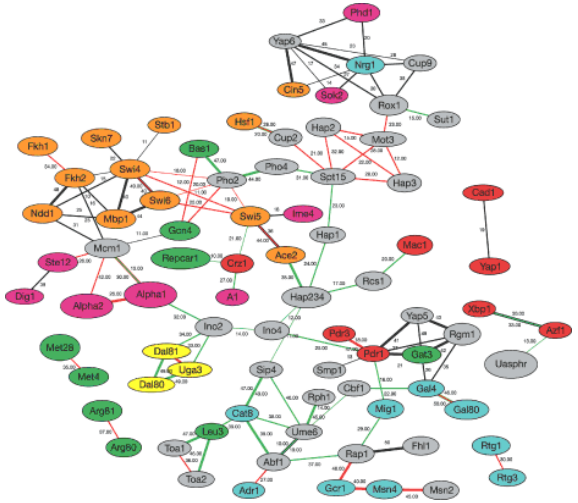
Graphe acyclique d'une expression (DAG)



((a+b)*c+d/(b*(a+b))) * (a+b)*c

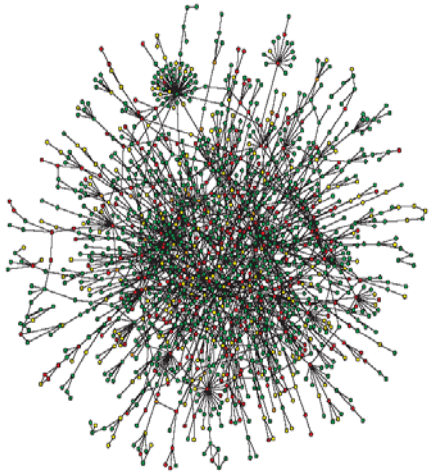
UMLV ©

Réseau de régulation de génes en biologie



UMLV ©

Réseau d'interaction de protéines (chez la levure)



UMLV ©

Algorithmes

Explorations
- Parcours en profondeur, en largeur
- Tri topologique
- Composantes fortement connexes, ...

Recherche de chemins
- Clôture transitive
- Chemin de coût minimal
- Circuits eulériens et hamiltoniens, ...

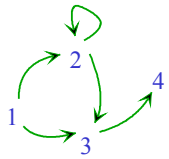
Arbres recouvrants
- Algorithmes de Kruskal et Prim

Réseaux de transport
- Flot maximal

Divers
- Coloration d'un graphe
- Test de planarité, ...

Terminologie

Graphe : $G = (S, A)$
Arc : $(s, t) \in A$ t adjacent à s, t successeur de s
Successeurs de s : $A(s) = \{ t \mid (s, t) \in A \}$
Boucle : $(t, t) \in A$

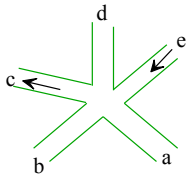


Chemins
Chemin : $c = ((s_0,s_1), (s_1,s_2), \ldots, (s_{k-1},s_k))$ où les $(s_{i-1},s_i) \in A$
origine = s_0
extrémité = s_k ((1,2), (2,2), (2,3), (3,4))
longueur = k
Circuit : chemin dont origine et extrémité coincident

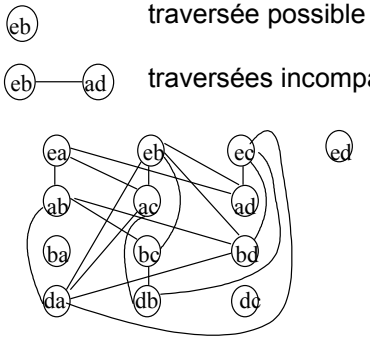
sommet, nœud = *vertex (vertices)*, arc = *arc*,
orienté = *directed*, arête = *edge*,
chemin = *path*, circuit = *circuit, cycle*

Problème de feux !

Graphe pour la modélisation
d 'un problème



sens unique

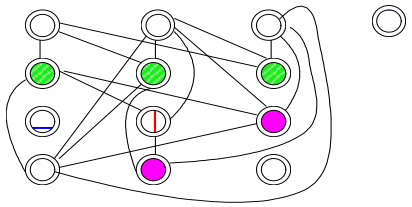


Coloration

$G = (S,A)$
coloration $f : S \rightarrow C$ telle que $(s,t) \in A \Rightarrow f(s) \neq f(t)$

$\text{Chr}(G) = \min_{f} \text{card } f(S)$, nombre chromatique de G

$\text{Chr}(G) = 4$



couleur = ensemble de traversées compatibles

Algorithme de coloration

$G = (S, A)$ $S = \{ s_1, s_2, \ldots, s_n \}$
G sans boucle !

fonction coloration-séquentielle (G graphe) : entier ;
début
  pour $i \leftarrow 1$ **à** n **faire {**
    $c \leftarrow 1$;
    tant que il existe t adjacent à s_i avec $f(t) = c$ **faire**
      $c \leftarrow c + 1$;
    $f(s_i) \leftarrow c$;
  }
  retour $\max (f(s_i), i = 1, \ldots, n)$;
fin

Temps d 'exécution : $O(n^2)$ Calcul de Chr (G) : $O(n^2 n!)$
(appliquer la fonction à toutes les permutations de S)
Aucun algorithme polynomial connu !

Représentations

UMLV ©

$G = (S, A)$ $S = \{1, 2, ..., n\}$

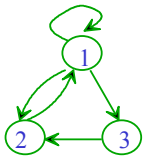
Liste des arcs
- représentation compacte
- hachage sur origine des arcs

Matrice d'adjacence
- utilisation d'opérations matricielles
- temps de traitement courant : quadratique

Listes de successeurs
- réduit la taille si $\text{card}A << (\text{card}S)^2$
- temps de traitement courant : $O(\text{card}S + \text{card}A)$

Matrices d'adjacence

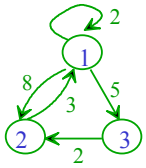
UMLV ©



$S = \{ 1, 2, 3 \}$
$A = \{ (1,1), (1, 2), (1, 3), (2, 1), (3, 2)\}$

$M = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$

$M[i, j] = 1$ ssi j adjacent à i

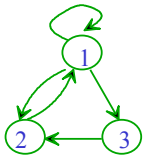


$V = \begin{pmatrix} 2 & 8 & 5 \\ 3 & 0 & 0 \\ 0 & 2 & 0 \end{pmatrix}$

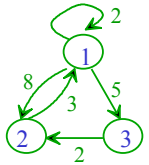
Valuation : $v : A \longrightarrow X$

Listes de successeurs

UMLV ©

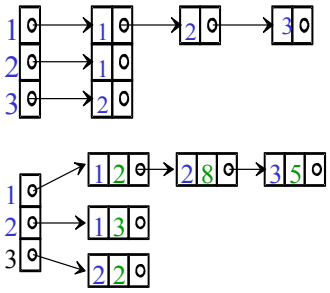


Listes des $A(s)$



Valuation : $v : A \longrightarrow X$

$S = \{ 1, 2, 3 \}$
$A = \{ (1,1), (1, 2), (1, 3), (2, 1), (3, 2)\}$



Exploration

UMLV ©

$G = (S, A)$
Explorer G = visite de tous les sommets
et 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, ...)

Parcours en profondeur ou en largeur
- extensions des parcours d'arbres

Parcours en profondeur

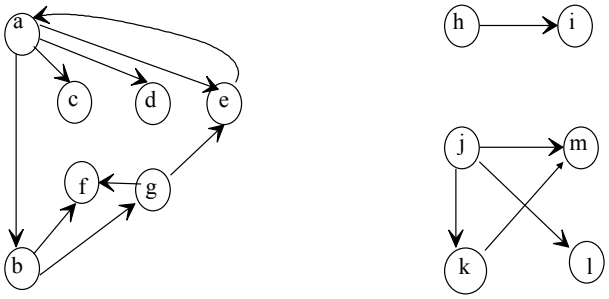
Marquage nécessaire

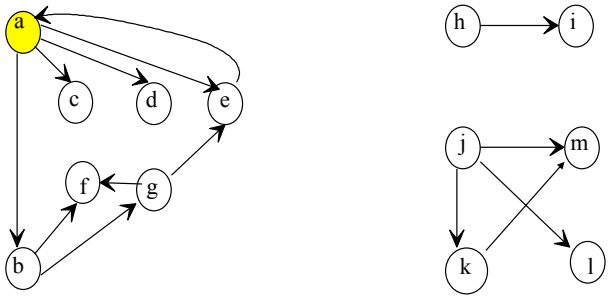
```
pour chaque sommet de s de G  faire
        visité[s] ⟵  faux ; //s est « blanc »
pour chaque sommet s de G  faire
        si non visité [s] alors Prof (s) ;

procédure  Prof (s sommet de G) ;
début
        action préfixe sur s ;
        visité[s] ⟵ vrai ; //s devient « jaune »
        pour chaque t successeur de s faire {
                action sur l'arc (s,t) ;
                si non  visité[ t ]  alors Prof ( t ) ;
        }
        action suffixe sur s ; //s devient « rouge »
fin
```

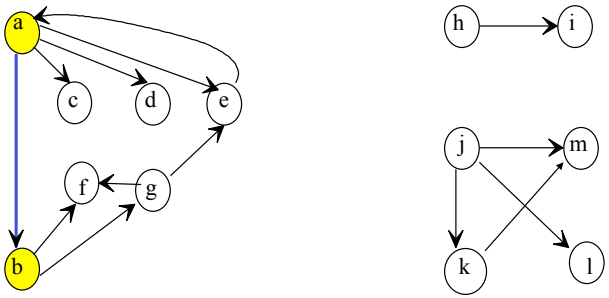

Trois états de sommets

Au cours d'une exploration :

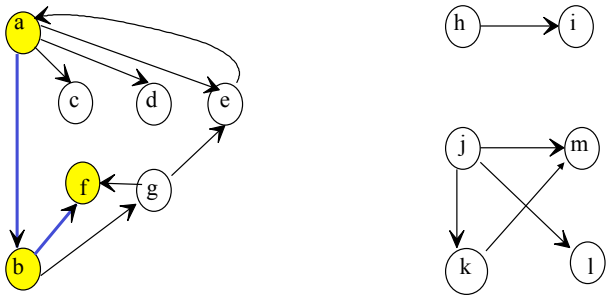
état [*s*] = blanc *s* : pas encore visité
état [*s*] = jaune *s* : en cours de visite
état [*s*] = rouge *s* : visite achevée



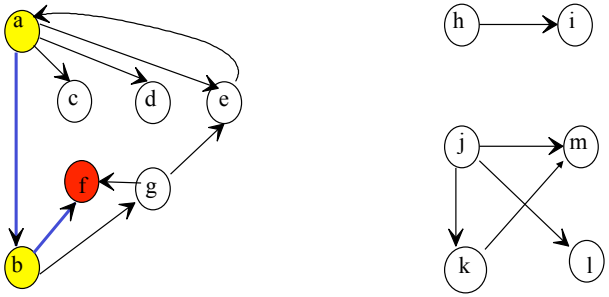



a
c
d
e
f
g
b

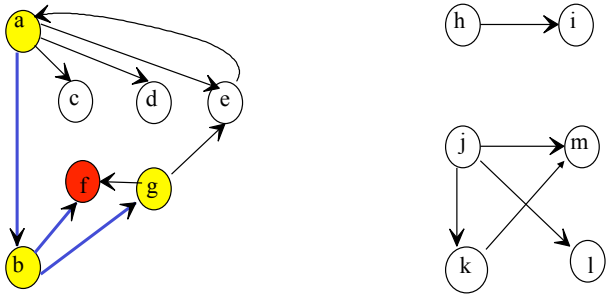
h
i
j
m
k
l


a
c
d
e
f
g
b

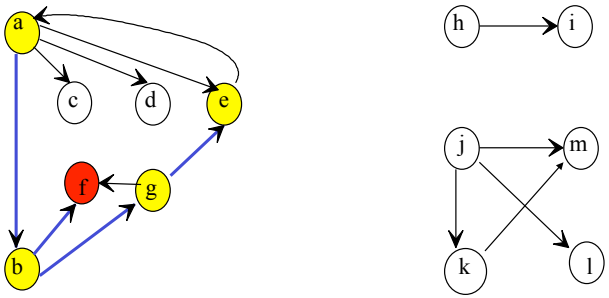
h
i
j
m
k
l


a
c
d
e
f
g
b

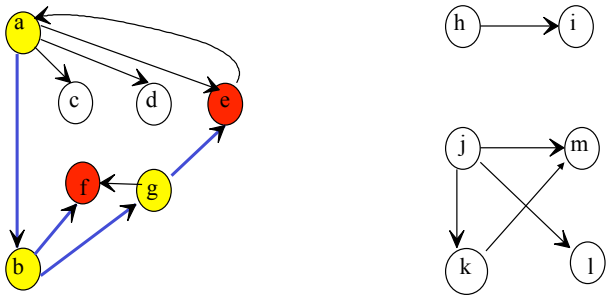
h
i
j
m
k
l

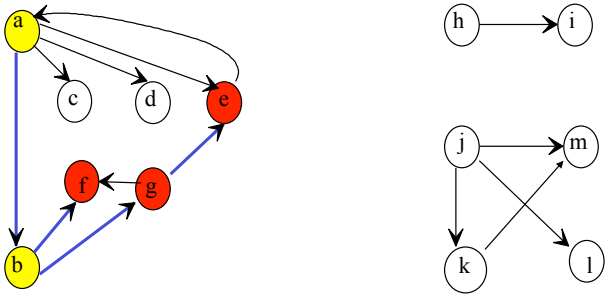

a
c
d
e
f
g
b

h
i
j
m
k
l

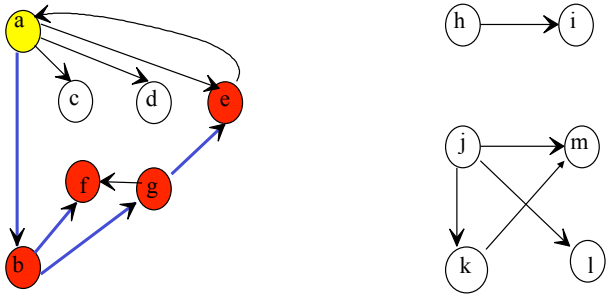

a
c
d
e
f
g
b

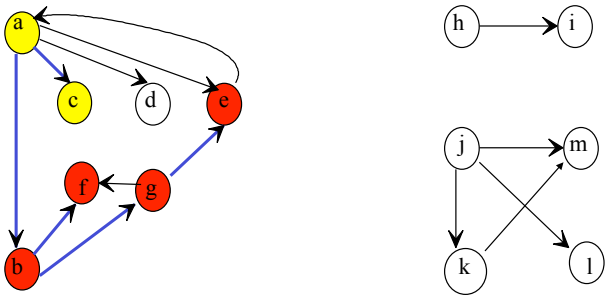
h
i
j
m
k
l


a
c
d
e
f
g
b
h
i
j
m
k
l

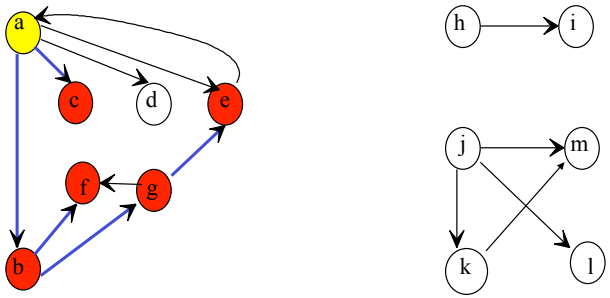

a
c
d
e
f
g
b

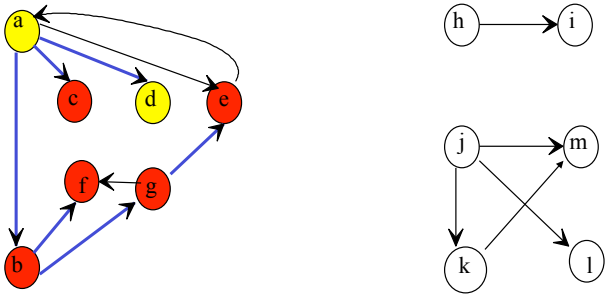
h
i
j
m
k
l


a
c
d
e
f
g
b
h
i
j
m
k
l

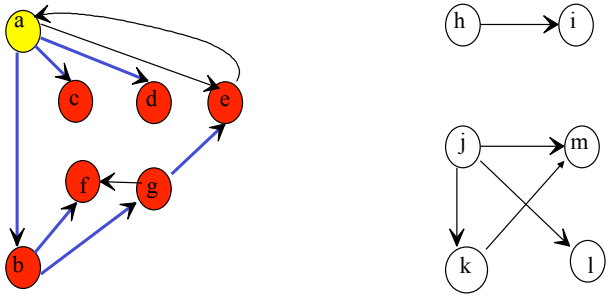

a
c
d
e
f
g
b

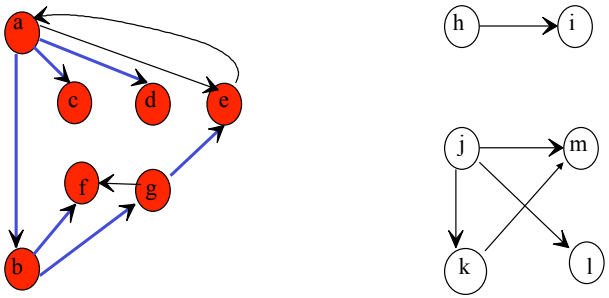
h
i
j
m
k
l


a
c
d
e
f
g
b
h
i
j
m
k
l

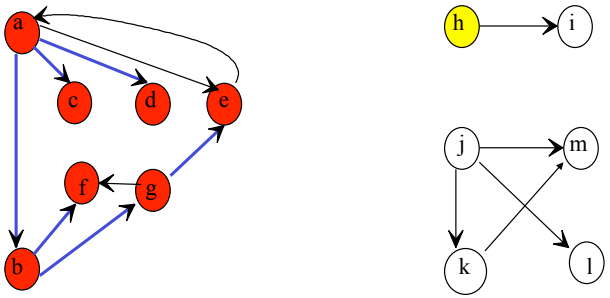

a
c
d
e
f
g
b

h
i
j
m
k
l

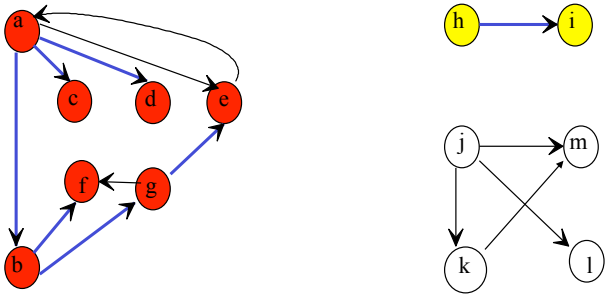

a
c
d
e
f
g
b
h
i
j
m
k
l


a
c
d
e
f
g
b

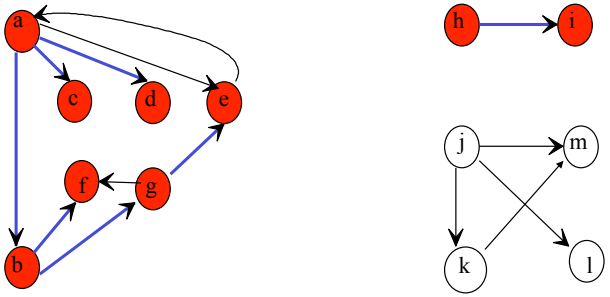
h
i
j
m
k
l


a
c
d
e
f
g
b

h
i
j
m
k
l

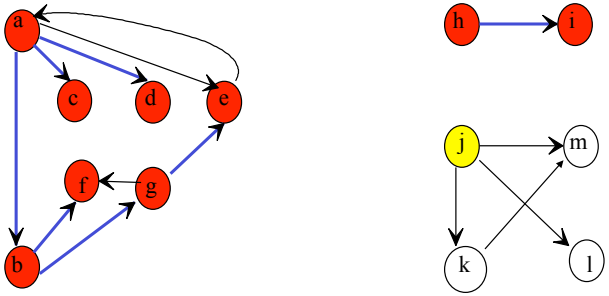

a
c
d
e
f
g
b

h
i
j
m
k
l


a
c
d
e
f
g
b

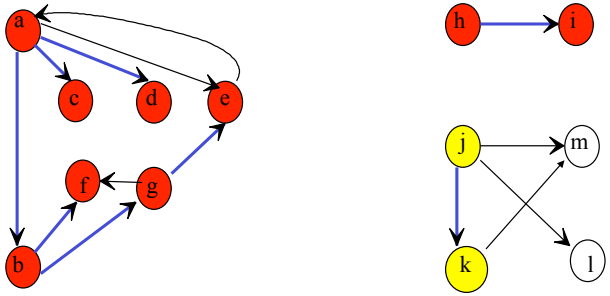
h
i
j
m
k
l

UMLV ©



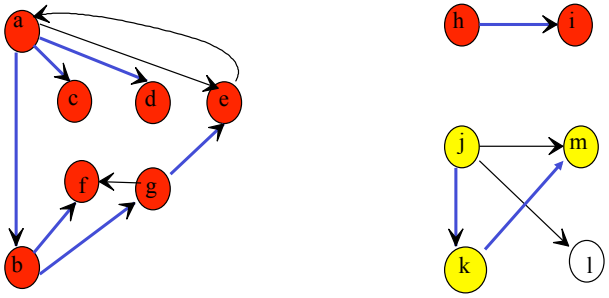
137

UMLV ©



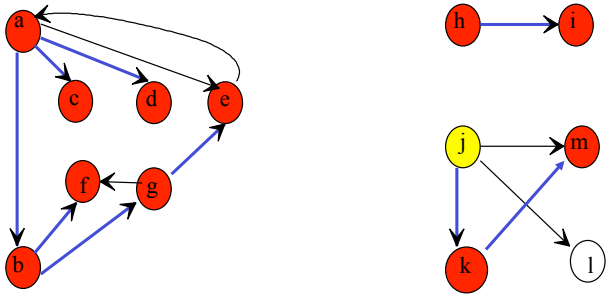
138

UMLV ©

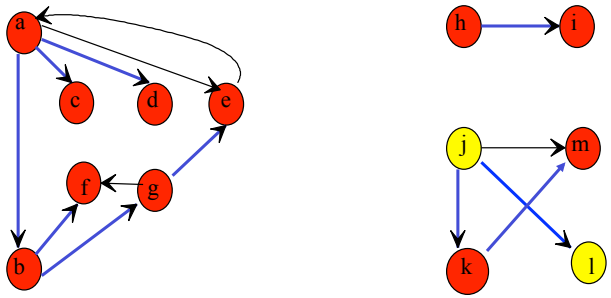


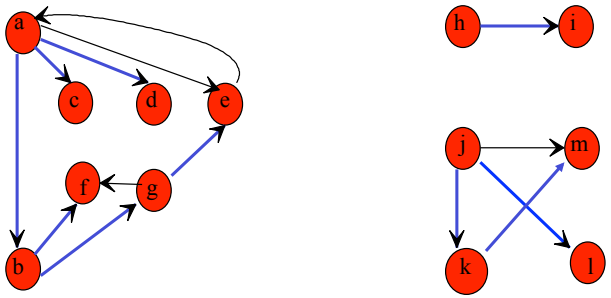
139

UMLV ©



140



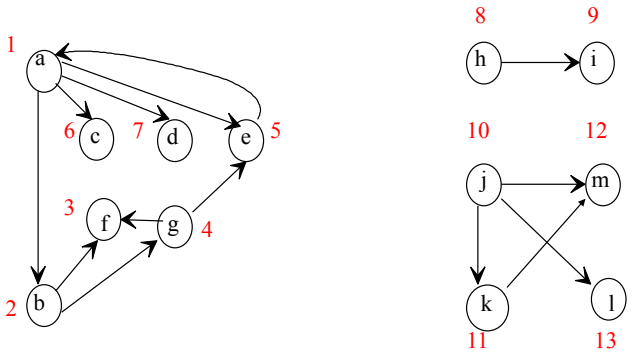


Numérotation

fonction Numérotation (*G* graphe) : table des numéros
    pour chaque sommet *s* de *G* **faire**
        no [*s*] ← 0 ;
    nb ← 0 ;
    pour chaque sommet *s* de *G* **faire**
        si *no* [*s*] = 0 **alors** Num (*s*) ;
    retour (*no*) ;
fin
procédure Num (*s* sommet de *G*) ;
début
    nb ← *nb* + 1 ; *no* [*s*] ← *nb* ;
    pour chaque *t* successeur de *s* **faire**
        si *no* [*t*] = 0 **alors** Num (*t*) ;
fin

nombre d'appels de Num = card *S*
nombre de « *no* [*t*] = 0 » dans Num = card *A*

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



Temps de parcours

T (« pour chaque sommet ») = O(card S)

Matrice d'adjacence

T (« **pour** chaque t adjacent à s ») =
T (« **pour** chaque sommet t tel que $M[s,t] = 1$ ») = O(card S)
⇒ parcours en $O((\text{card } S)^2)$

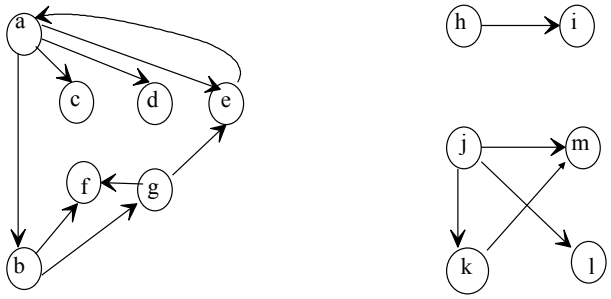
Listes de successeurs

T (« **pour** chaque t adjacent à s ») = O(card $A(s)$)
⇒ parcours en O(card S + card A)

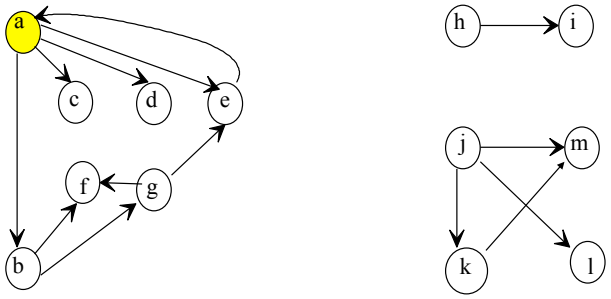
Parcours en profondeur: version itérative

```
Procédure Prof (s sommet de G) ;
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
```

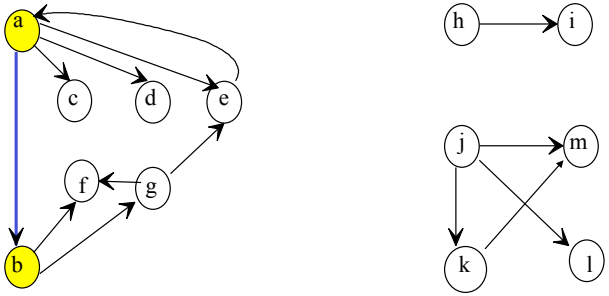
Pour "action préfixe" uniquement
Note : il peut y avoir plusieurs occurrences d'un même sommet dans la pile. Ne pas l'interdire !



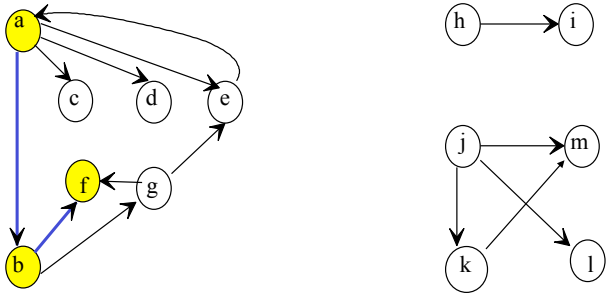
Pile : a



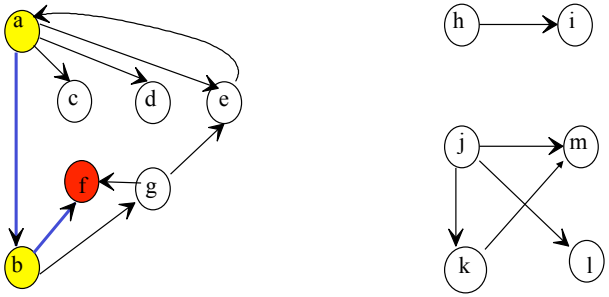
Pile : e d c b



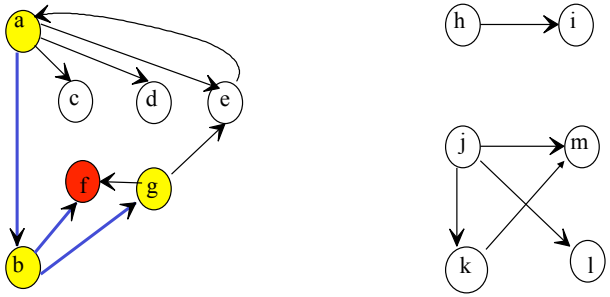
Pile : e d c g f



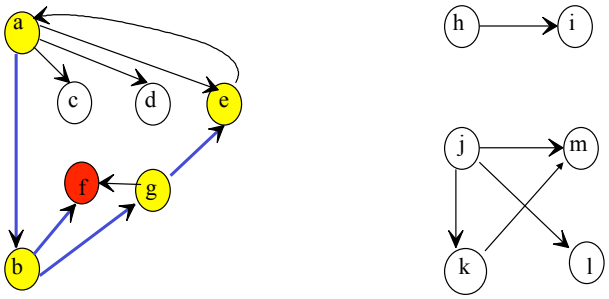
Pile : e d c g



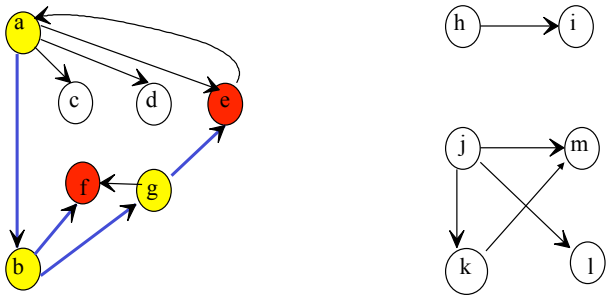
Pile : e d c g



Pile : e d c e



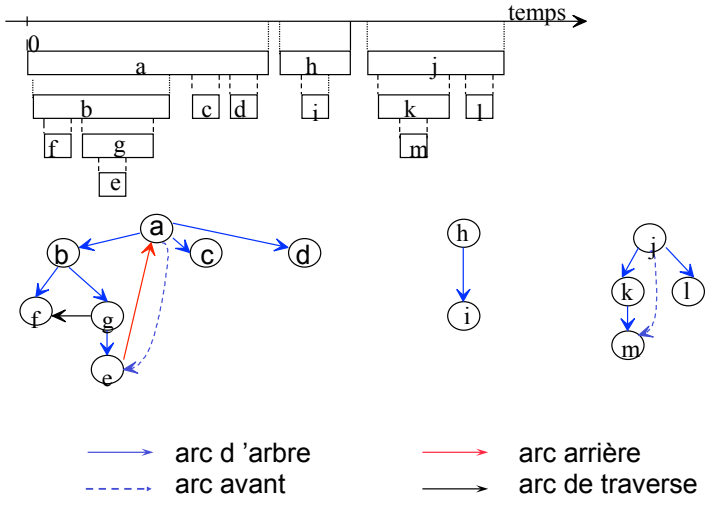
Pile : e d c



Pile : e d c

Et cetera

Forêt de l'exploration en profondeur



Détection d'un circuit

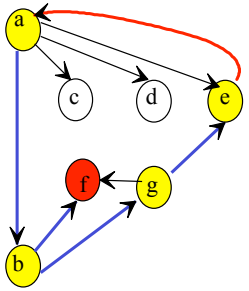
Proposition

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 $f(t) < d(s)$

Exemple

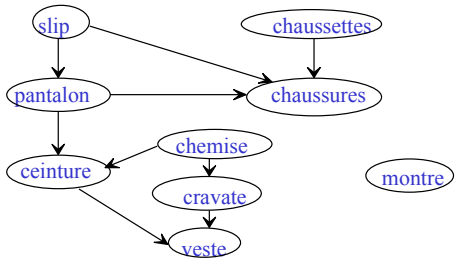


Pendant la visite du sommet e, on détecte un cycle passant par l 'arc (e, a) car a est aussi en cours de visite

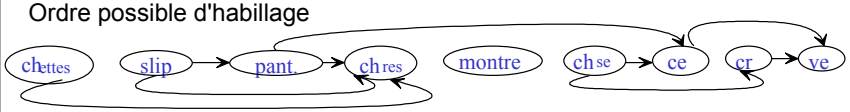
Tri topologique

Prolongement d 'un ordre partiel en un ordre total

graphe sans circuit (*DAG= Directed Acyclic Graph*), *i.e.* pas d 'arc arrière



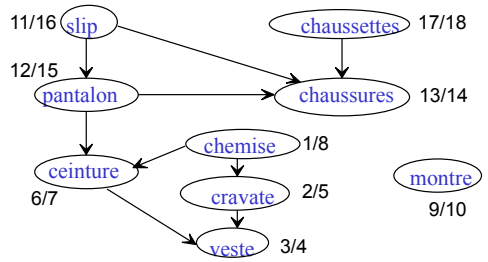
Ordre possible d'habillage



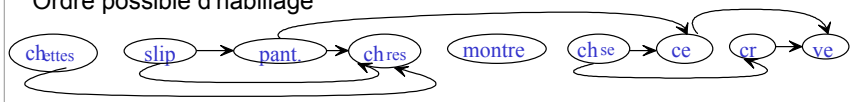
Tri topologique

Prolongement d 'un ordre partiel en un ordre total

graphe sans circuit (*DAG= Directed Acyclic Graph*), *i.e.* pas d 'arc arrière



Ordre possible d'habillage



Ordre final : ordre décroissant des $f(s)$ dans un parcours en profondeur

Tri topologique par exploration en profondeur

UMLV ©

```
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 adjacent à s  faire
                si non  visité [t] alors Topo (t) ;
        Ajouter s en tête de L ;
fin
```


Méthode itérative

UMLV ©

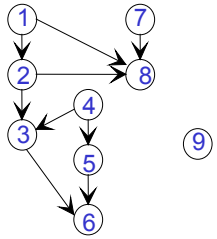
	1	2	3	4	5	6	7	8	9
Nb-préd	0	1	2	0	1	2	0	3	0

Sommets à traiter : 1 4 7 9
(sans prédécesseur)

Après traitement de 1 :

	1	2	3	4	5	6	7	8	9
Nb-Préd	-	0	2	0	1	2	0	2	0

Sommets à traiter : 4 7 9 2



Tri topologique itératif

UMLV ©

```
Fonction Tri-topologique (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
```

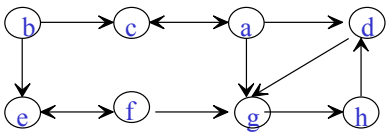
Temps d 'exécution : *O*(card *S* + card *A*)
avec gestion efficace de la liste des sommets à traiter

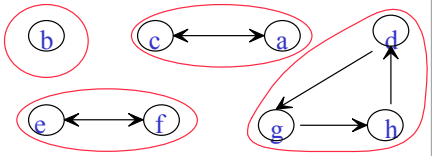
Composantes fortement connexes

UMLV ©

$G = (S, A)$ graphe
$G' = (S', A')$ sous-graphe de G ssi $S' \subseteq S$ et $A' \subseteq A \cap S' \times S'$

F **composante fortement connexe** de *G* :
F sous-graphe maximal de *G* tel que
deux sommets qlcq de *F* sont reliés par un chemin.





Calcul

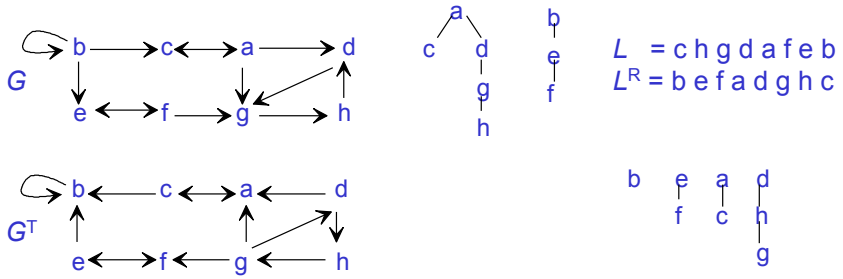
G^T = graphe transposé de G

Algorithme [Kosaraju 78] [Sharir 81]

L ← liste des sommets de G obtenus par
parcours en profondeur suffixe ;

à partir de L^R, appliquer un parcours en profondeur à G^T ;

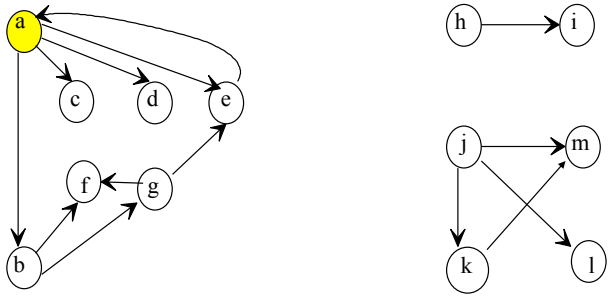
Les arbres de cette exploration sont
les composantes fortement connexes



Parcours en largeur

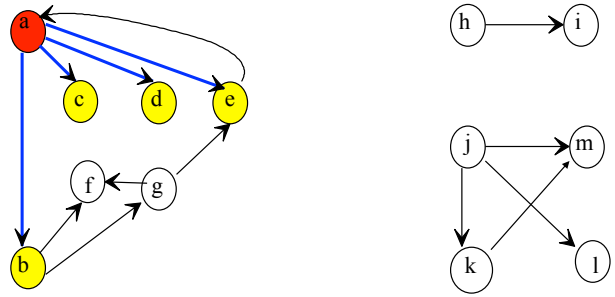
Procédure Larg (*s* sommet de *G*) ;
début
  File ← Ajouter (File-vide, *s*) ;
  tant que non Vide (File) **faire {**
    s' ← Premier (File) ; File ← Enlever (File, *s'*) ;
    si non visité [*s'*] **alors {**
      visité [*s'*] ← vrai ;
      pour *t* ← premier au dernier successeur de *s'* **faire**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

Note : il peut y avoir plusieurs occurrences d'un même sommet dans la file. On peut l'interdire !



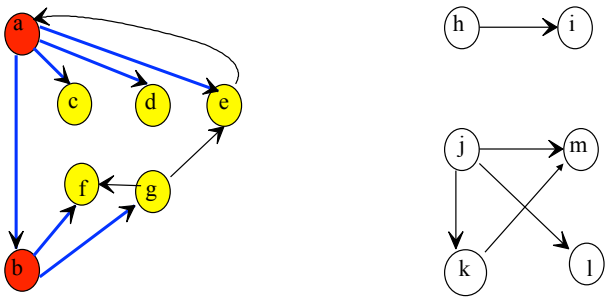
File : a

Ordre du parcours :



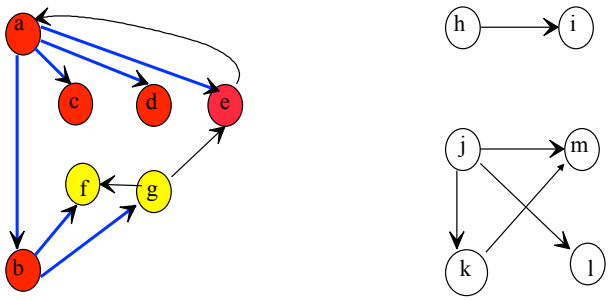
File : a b c d e

Ordre du parcours : a



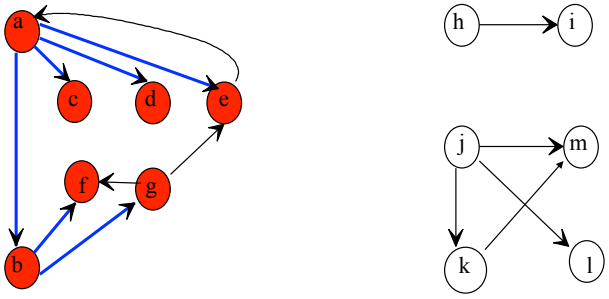
File : a b c d e f g

Ordre du parcours : a b



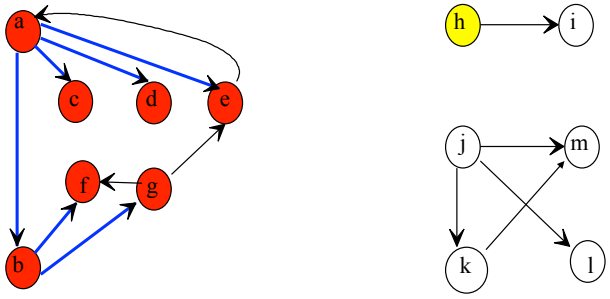
File : a b c d e f g

Ordre du parcours : a b c d e



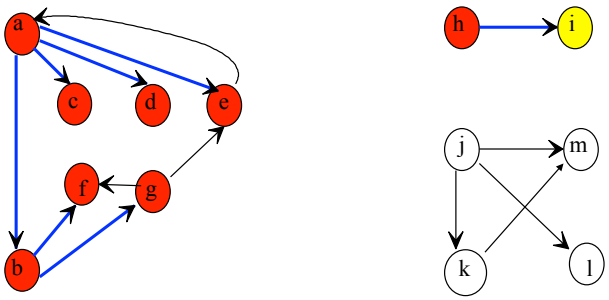
File : a b c d e f g

Ordre du parcours : a b c d f g



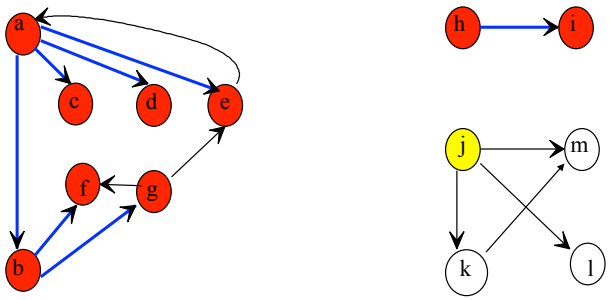
File : a b c d e f g h

Ordre du parcours : a b c d f g h



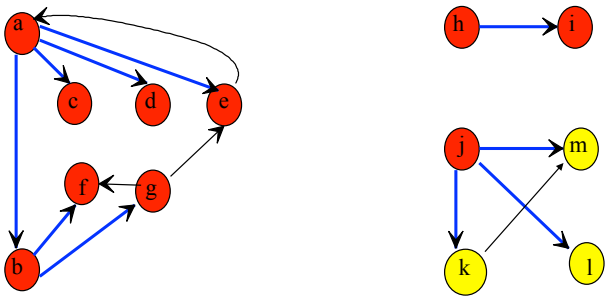
File : a b c d e f g h i

Ordre du parcours : a b c d f g h i



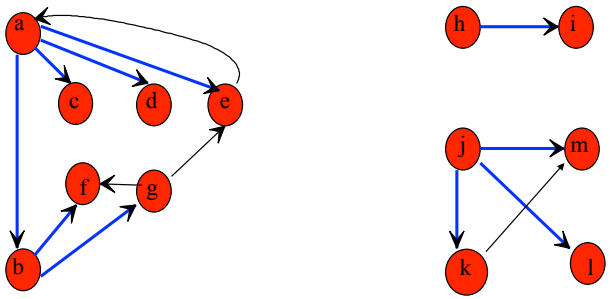
File : a b c d e f g h i j

Ordre du parcours : a b c d f g h i j



File : a b c d e f g h i j k l m

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



File : a b c d e f g h i j k l m

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

Chemin le plus court

Supposons qu'il existe un chemin de *s* à *v*

d(v) : numéro d'itération à laquelle *v* a été visité pour la première fois (devenu jaune)

Alors *d(v)* est la longueur du plus court chemin de *s* à *v*