Algorithmique des graphes - Cours 3

Olivier Baudon

Université de Bordeaux

24 septembre 2021

Parcours en profondeur

Principe

On examine les sommets du graphe en partant d'un sommet s. Tant que c'est possible, on "descend" dans le graphe de voisin en voisin. Sinon, on remonte jusqu'à être sur un sommet ayant encore un voisin non encore visité, et on redescend à nouveau. Si on remonte jusqu'au premier sommet et qu'aucun de ses voisins (ou successeur) n'a pas été visité, alors s'il reste encore un sommet s' non encore visité, on repart de s'.
Ce processus est répété tant qu'il existe des sommets non visités. Pour cela, on utilise une boucle et une pile. Le plus simple est d'utiliser la pile d'appels générée par des appels récursifs.
Une variable *temps*, initialisée à 0, sert à définir pour chaque sommet u deux valeurs : $d(u)$ l'heure (ou la date) de début de visite de u et $f(u)$ l'heure (ou la date) de fin de visite de u. *temps* est incrémentée avant chaque mise à jour des tableaux d et f.

Parcours en profondeur

Énoncé

Voir le document "Algorithmes de graphes" page 3.

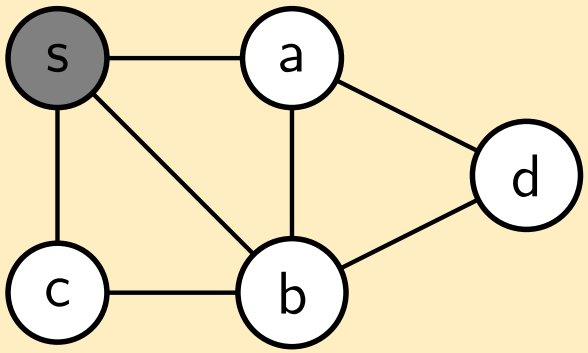
Complexité

La complexité du parcours en profondeur est en $O(n+m)$.
En effet, la procédure `VisiterPP(v)` est appelée une et une seule fois par sommet, car dès que `VisiterPP(v)` est appelée, le sommet v passe en GRIS et ne redevient jamais BLANC.
Dans cette procédure, on regarde chaque arête incidente à v ou chaque arc sortant de v. Donc le contenu de la boucle $3-8$ de `VisiterPP(v)` sera exécutée au total $2m$ fois si G est non orienté ou m fois si G est orienté.

Parcours en profondeur

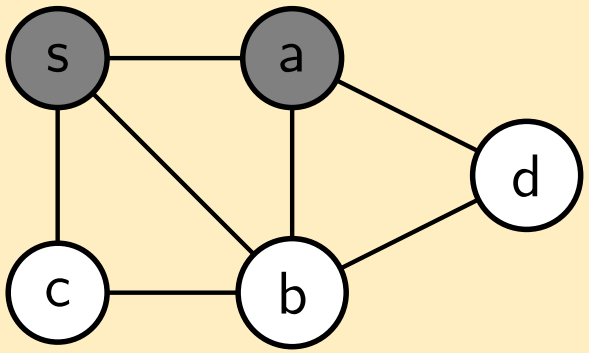
Exemple dans le cas non orienté

La couleur "NOIR" sera affichée avec du bleu afin de laisser l'étiquette lisible.



$Pile = [s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1				
$f(v)$					

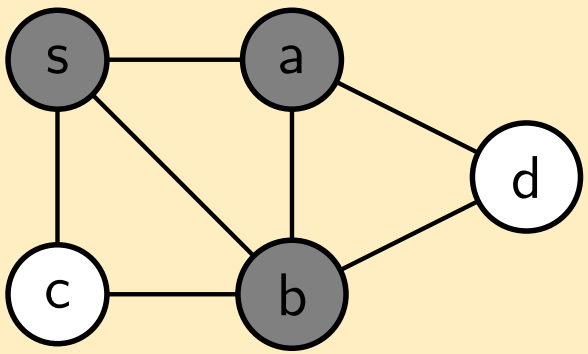


$Pile = [a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2			
$f(v)$					

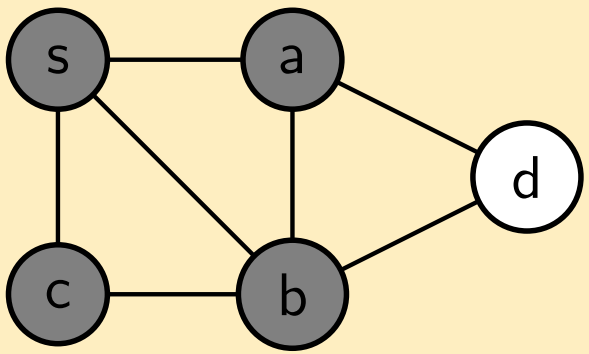
Parcours en profondeur

Exemple dans le cas non orienté



$Pile = [b, a, s]$

sommet v	s	a	b	c	d
$d(v)$	1	2	3		
$f(v)$					

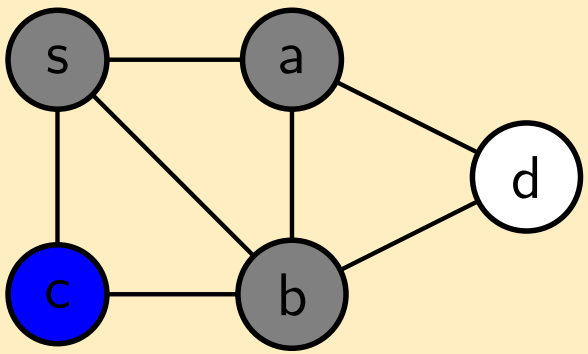


$Pile = [c, b, a, s]$

sommet v	s	a	b	c	d
$d(v)$	1	2	3	4	
$f(v)$					

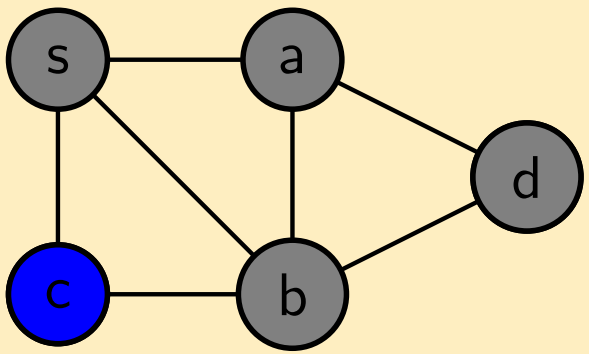
Parcours en profondeur

Exemple dans le cas non orienté



$Pile = [b, a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3	4	
$f(v)$				5	

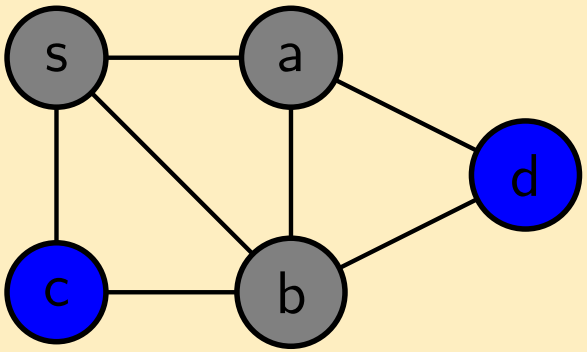


$Pile = [d, b, a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3	4	6
$f(v)$				5	

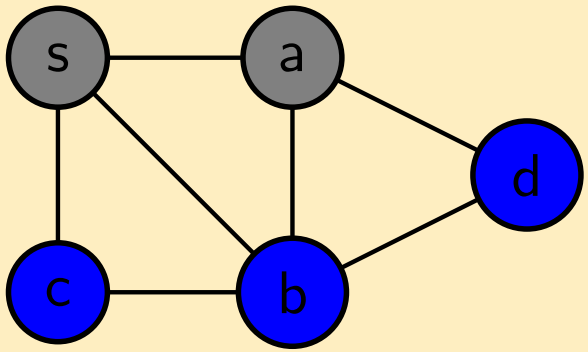
Parcours en profondeur

Exemple dans le cas non orienté



$Pile = [b, a, s]$

sommet v	s	a	b	c	d
$d(v)$	1	2	3	4	6
$f(v)$				5	7

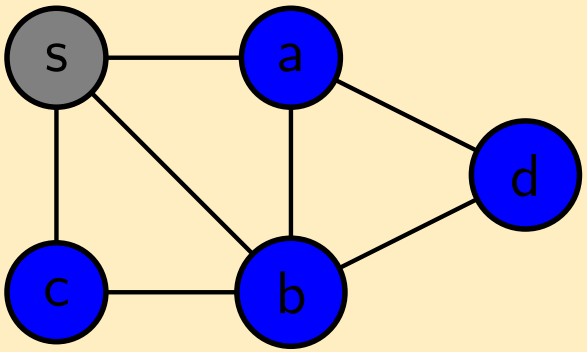


$Pile = [a, s]$

sommet v	s	a	b	c	d
$d(v)$	1	2	3	4	6
$f(v)$			8	5	7

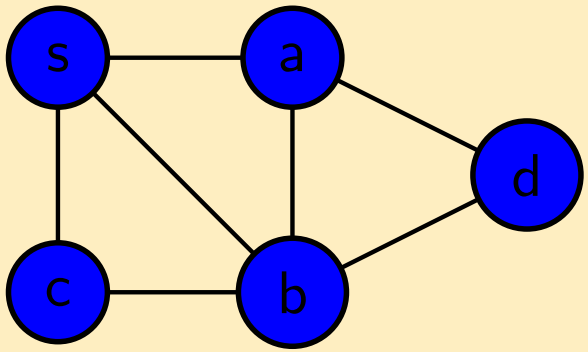
Parcours en profondeur

Exemple dans le cas non orienté



$Pile = [s]$

sommet v	s	a	b	c	d
$d(v)$	1	2	3	4	6
$f(v)$		9	8	5	7

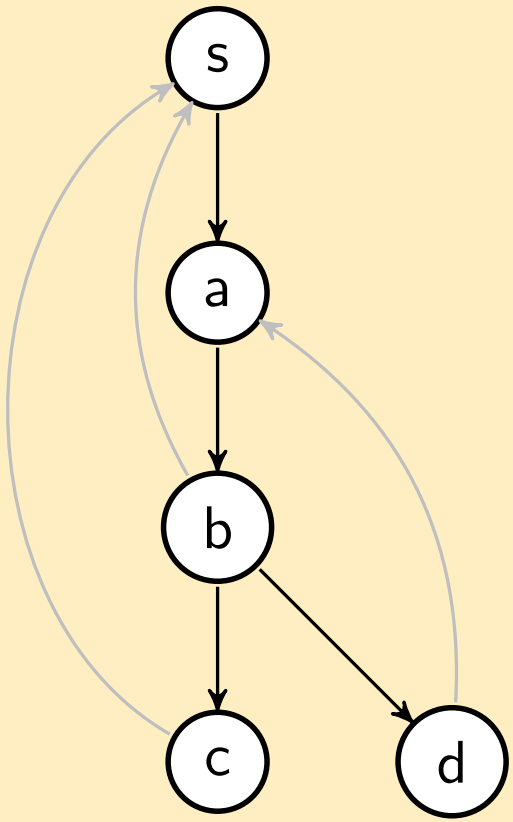


$Pile = []$

sommet v	s	a	b	c	d
$d(v)$	1	2	3	4	6
$f(v)$	10	9	8	5	7

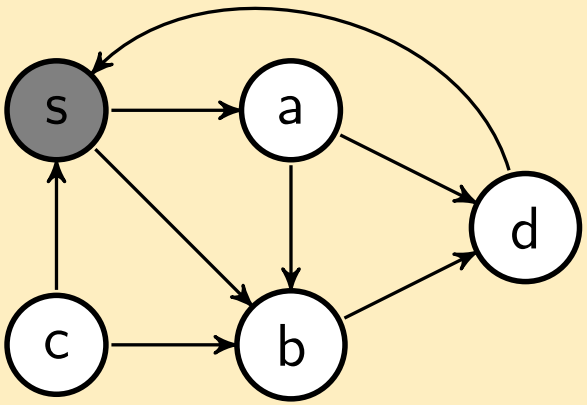
Parcours en profondeur

Arborescence du parcours en profondeur



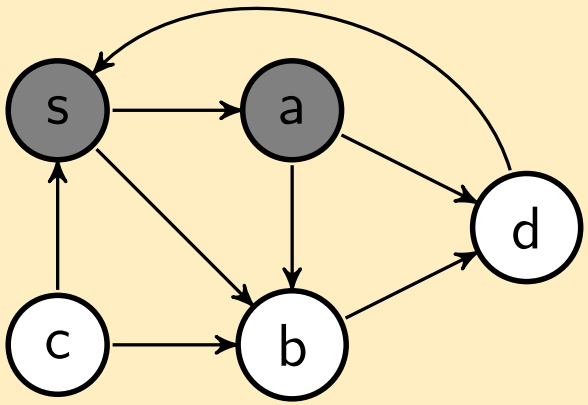
Parcours en profondeur

Exemple dans le cas orienté



$Pile = [s]$

sommet v	s	a	b	c	d
$d(v)$	1				
$f(v)$					

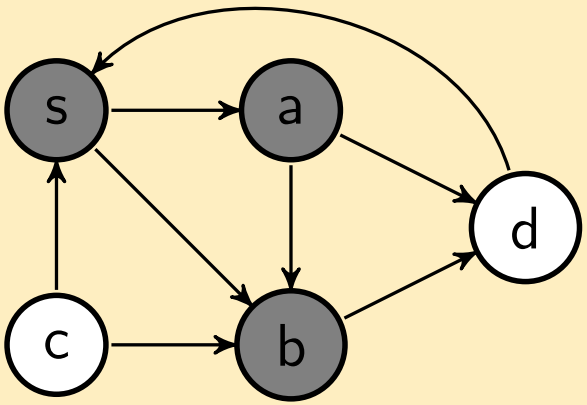


$Pile = [a, s]$

sommet v	s	a	b	c	d
$d(v)$	1	2			
$f(v)$					

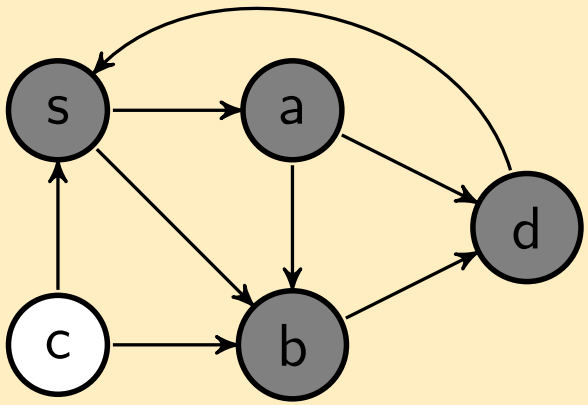
Parcours en profondeur

Exemple dans le cas orienté



$Pile = [b, a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3		
$f(v)$					

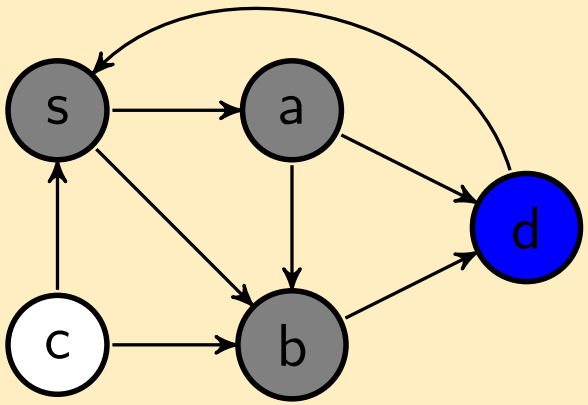


$Pile = [d, b, a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3		4
$f(v)$					

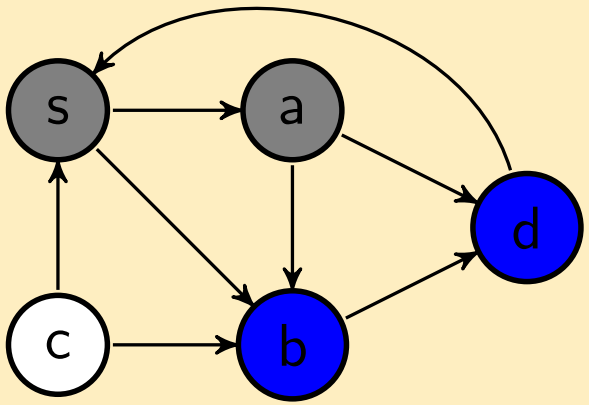
Parcours en profondeur

Exemple dans le cas orienté



$Pile = [b, a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3		4
$f(v)$					5

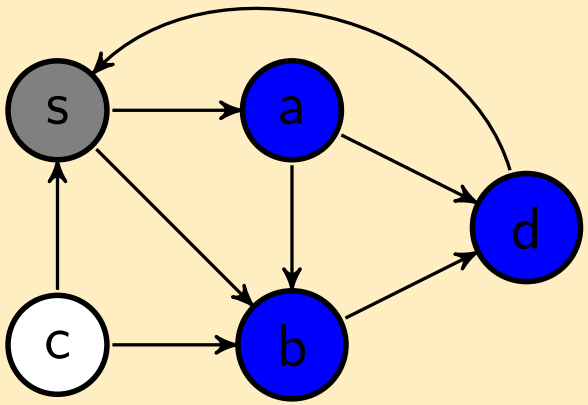


$Pile = [a, s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3		4
$f(v)$			6		5

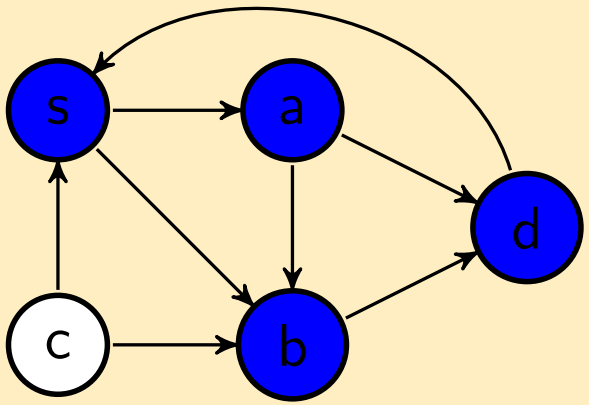
Parcours en profondeur

Exemple dans le cas orienté



$Pile = [s]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3		4
$f(v)$		7	6		5

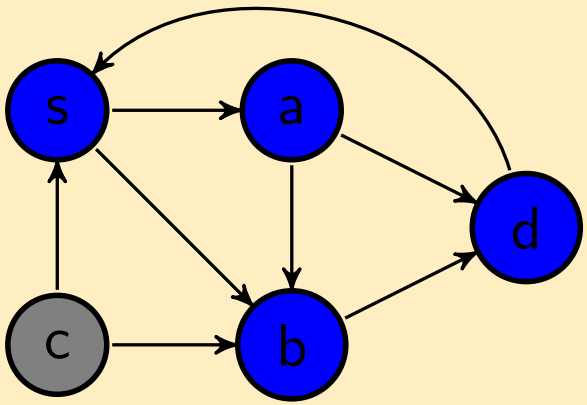


$Pile = []$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3		4
$f(v)$	8	7	6		5

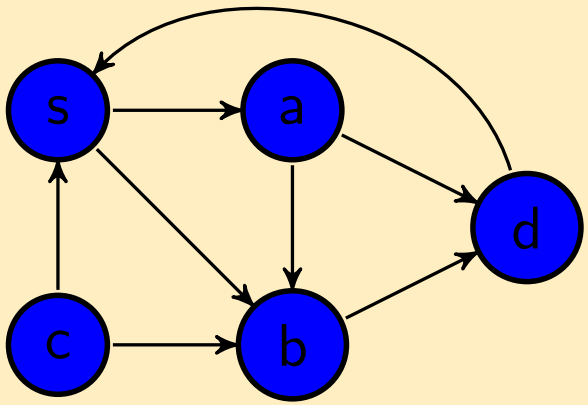
Parcours en profondeur

Exemple dans le cas orienté



$Pile = [c]$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3	9	4
$f(v)$	8	7	6		5

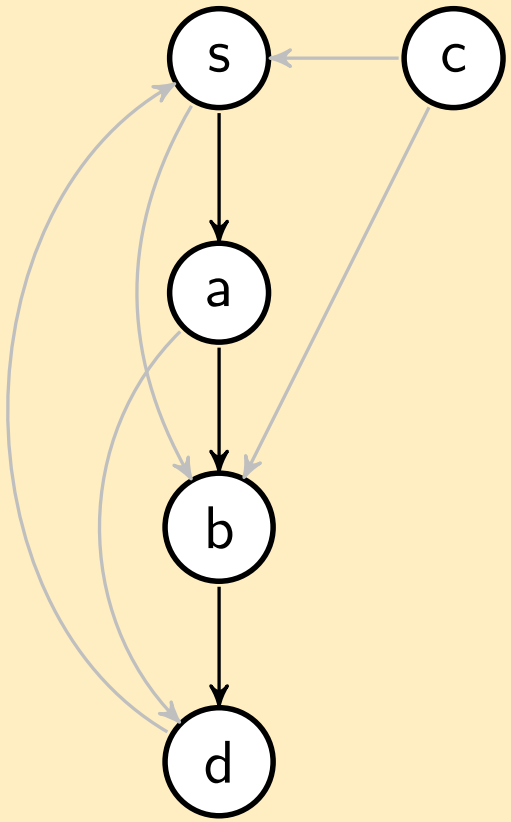


$Pile = []$

sommet v	*s*	*a*	*b*	*c*	*d*
$d(v)$	1	2	3	9	4
$f(v)$	8	7	6	10	5

Parcours en profondeur

Arborescence du parcours en profondeur



Parcours en profondeur

Type des arcs de l'arborescence

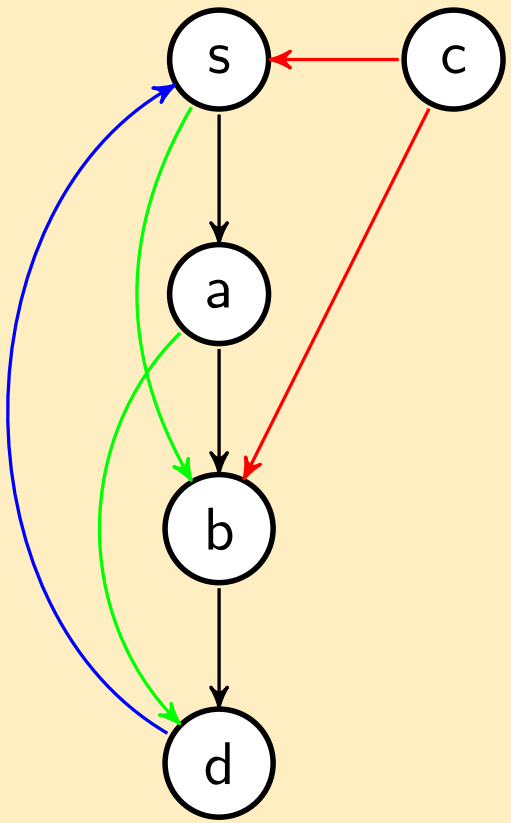
On distingue 4 types d'arcs :

- de liaison : ce sont les arcs reliant un sommet à ses fils ;
- avant : ce sont les arcs reliant un sommet à ses descendants $\neq$ de ses fils ;
- arrière ou de retour : ce sont les arcs reliant un sommet à l'un de ses ancêtres ;
- transverse : ce sont les arcs reliant un sommet aux sommets avec lesquels il n'a aucune relation dans l'arborescence.

Parcours en profondeur

Type des arcs de l'arborescence

Dans le dessin suivant, les arcs de liaison sont en noir, les arcs de retour en bleu, les arcs avant en vert et les arcs transverses en rouge.



Parcours en profondeur

Classe des arcs en fonction des couleurs des sommets

L'idée est de connaître le type d'un arc lors de sa visite en fonction de la couleur du sommet destination (le sommet origine étant toujours GRIS). Soit uv l'arc visité.

- Si v est BLANC, alors l'arc uv sera un arc de liaison.
- Si v est GRIS, alors l'arc uv sera un arc retour (aussi appelé arrière.
- Si v est NOIR, alors l'arc uv sera un arc avant ou transverse selon que u est un ancêtre ou non de v.

Parcours en profondeur

Classe des arcs dans le cas non orienté

Dans le cas du parcours en profondeur d'un graphe G non orienté, alors les arcs sont soit de liaison, soit retour. Il n'y a pas d'arc avant, ni d'arc transverse.
Justification : Lorsque l'on examine pour la première fois une arête uv, disons à partir d'un sommet u, si v n'a pas encore été visité, alors uv sera un arc de liaison.
Sinon, v est forcément un ancêtre de u car on ne peut finir de visiter v sans regarder l'arête uv. Donc v ne peut être NOIR et donc uv est un arc de retour.

Parcours en profondeur

Théorème des parenthèses

Soit u et v deux sommets de G, et $[d(u), f(u)]$, $[d(v), f(v)]$ les intervalles définis par leurs heures de début et fin de visite. On suppose que $d(u) < d(v)$. Deux cas sont possibles :

- Soit $[d(v), f(v)] \subset [d(u), f(u)]$. Dans ce cas v est un descendant de u.
- Soit $[d(v), f(v)] \cap [d(u), f(u)] = \emptyset$. Dans ce cas, aucun des deux n'est le descendant de l'autre dans l'arborescence.

Justification :

- dans le cas où $[d(v), f(v)] \subset [d(u), f(u)]$, cela signifie que v a été visité quand u était GRIS. Donc v sera un descendant de u et on finira la visite de v avant celle de u.
- Si $[d(v), f(v)] \cap [d(u), f(u)] = \emptyset$, alors comme $d(v) > d(u)$, on a $f(v) > d(v) > f(u) > d(u)$ et donc, on aura fini de visiter u avant de commencer la visite de v.

Parcours en profondeur

Théorème du chemin blanc

Dans une forêt obtenu par un parcours en profondeur d'un graphe $G = (V, E)$, un sommet v est un descendant d'un sommet u si et seulement si lors du début de visite du sommet u, il existe une chaîne ou un chemin reliant u à v composé exclusivement de sommets blancs.

Théorème du chemin blanc - Justification

si v est un descendant de u, les sommets reliant u à v dans l'arborescence étaient tous BLANC au moment du début de la visite de u.
Inversement, supposons qu'il existe un chemin composé de sommets blancs entre u et v à l'instant $d(u)$, et que v ne soit pas un descendant de u. Choisissons v de telle sorte qu'il soit le sommet de ce chemin blanc le plus proche de u qui ne soit pas un de ses descendants. Et soit w son prédécesseur sur le chemin blanc. w sera bien un descendant de u. Or si v n'est pas un descendant de u et donc de w, quand on finit de visiter w, v sera encore blanc et devrait donc être visité via l'arête wv. Donc v sera un fils de w et donc un descendant de u.

Parcours en profondeur

Applications

Il existe de nombreuse applications issues du parcours en profondeur. Parmi elles, citons :

1. le tri topologique ;
2. le calcul des composantes fortement connexes ;
3. la recherche de points d'articulation (voir feuille TD 4 exo 4) ;
4. le test de planarité d'un graphe (Hopcroft et Tarjan).

A noter que dans tous les cas cités ci-dessus, les algorithmes ainsi obtenus sont en temps linéaire, i.e. en $O(n + m)$.

Tri topologique

Définition

Le tri topologique d'un graphe G orienté est un ordonnancement de ses sommets de telle sorte que si $v_1, \ldots, v_n$ est l'ordre obtenu, alors :
$\forall v_i, v_j \in V(G), v_i v_j \in E(G) \Rightarrow i < j$

Théorème

Un graphe G orienté admet un tri topologique si et seulement si il est sans circuit.
Justification :
$\Rightarrow$ (par contradiction)
soit $v_1 v_2 \ldots v_{k-1} v_1$ un circuit de G. Si G admet un tri topologique, alors v_1 doit être rangé avant v_2 car il existe un arc $v_1 v_2$, v_2 avant $v_3, \ldots v_{k-1}$ avant v_1, ce qui est impossible.
Donc "G graphe orienté admet un tri topologique" $\Rightarrow$ G est sans circuit.

Tri topologique

Justification (suite)

$\Leftarrow$

A l'inverse, si G est sans circuit, montrons que G admet un tri topologique.

On peut montrer qu'il existe un sommet de G de degré entrant 0. En effet, si l'on considère un chemin P de longueur maximale dans G, disons $P = v_0, v_1, \ldots v_k$, alors v_0 est de degré entant 0 dans G. Sinon, soit il a un prédécesseur hors de $\{v_1, \ldots v_k\}$ et dans ce cas, P n'est pas un chemin de longueur maximale, soit il a un prédécesseur parmi $\{v_1, \ldots v_k\}$ et dans ce cas, G n'est pas sans circuit.

On prend le sommet v de degré entrant 0 comme le premier sommet dans le tri topologique. $G - v$ est forcément sans circuit, donc possède également un sommet de degré entrant 0, que l'on prendra en deuxième position. On répète cette opération jusqu'à ce que tous les sommets de G aient été placés.

Tri topologique

Algorithme (basé sur le parcours en profondeur)

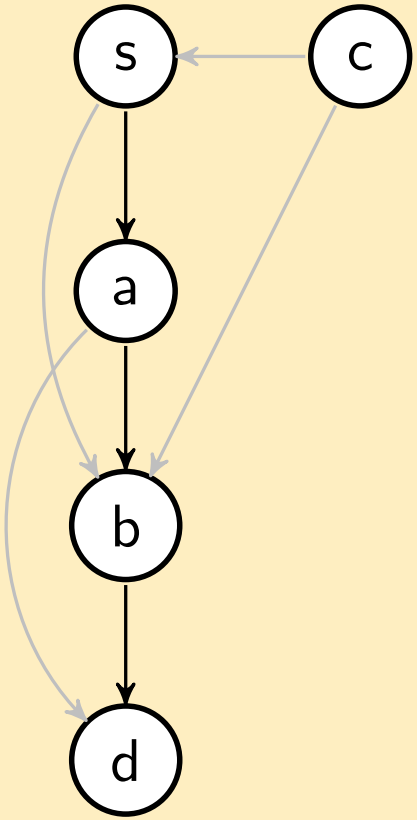
Voir le document "Algorithmes de graphes" page 4.

Complexité

Il est facile de constater que la complexité du tri topologique basé sur le parcours en profondeur est identique à celle du parcours en profondeur.
Donc le tri topologique est en $O(n + m)$

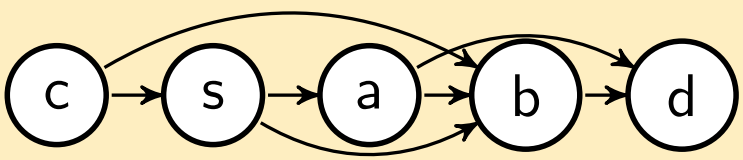
Tri topologique

Exemple



sommet v	s	a	b	c	d
$d(v)$	1	2	3	9	4
$f(v)$	8	7	6	10	5

Tri topologique : c, s, a, b, d



Tri topologique

Justification de l'algorithme

Théorème : un graphe G orienté est sans circuit si et seulement si le parcours en profondeur de G ne produit aucun arc de retour.
Preuve : on va en fait montrer que G possède un circuit si et seulement si son parcours en profondeur produit un arc retour.
Si lors du parcours, on a un arc retour uv, alors le graphe G possède un circuit constitué des arcs de liaisons reliant v à u complété de l'arc uv.
Si G possède un circuit $C = v_1, \dots v_k, v_1$, alors supposons (sans perte de généralité) que v_1 soit le premier sommet de C visité.
D'après le théorème du chemin blanc, v_k sera un descendant de v_1 et donc $v_k v_1$ un arc retour.

Tri topologique

Justification de l'algorithme

Montrons maintenant que l'ordre inverse de fin de visite de $PP(G)$ est bien un tri topologique de G. Il faut donc montrer que $uv \in E(G) \Rightarrow f(u) > f(v)$

En effet, soit uv un arc de G.

Si uv est un arc de liaison ou avant, alors v est un descendant de u et donc $f(v) < f(u)$ car $[d(v), f(v)] \subset [d(u), f(u)]$.

Si uv est un arc transverse, alors $d(v) < f(v) < d(u) < f(u)$.

Donc dans tous les cas, on aura bien $uv \in E(G) \Rightarrow f(u) > f(v)$.

Composantes fortement connexes

Énoncé de l'algorithme

Voir le document "Algorithmes de graphes" page 4.

Complexité

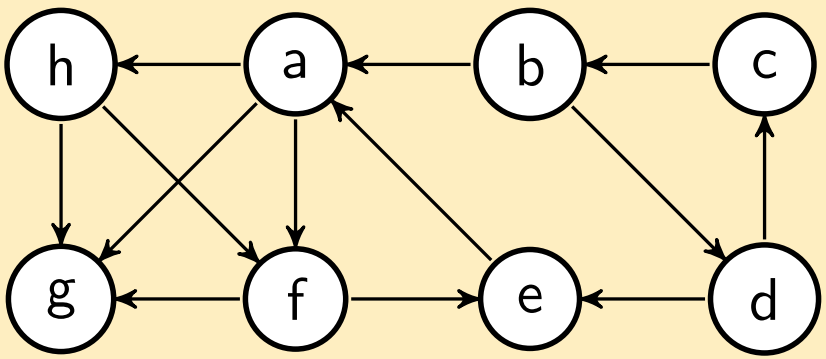
L'algorithme exécute deux parcours en profondeur, soit une complexité de $O(n+m)$. De plus, le calcul du graphe inversé G^{-1} est également en $O(n+m)$ en utilisant par exemple l'algorithme suivant où $Adj_G(v)$ désigne la liste des successeurs du sommet v dans le graphe G.

Inverser(G) :

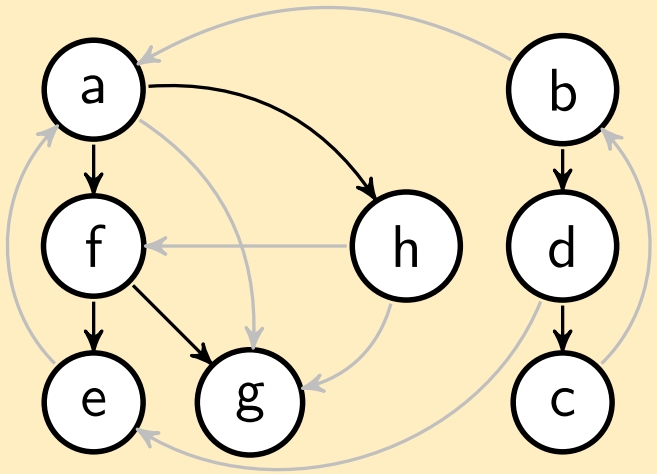
1: $G^{-1} \leftarrow (V(G), \emptyset)$
2: **pour tout** $v \in V(G)$ **faire**
3: **pour tout** $w \in Adj_G(v)$ **faire**
4: $INSERER(v, Adj_{G^{-1}}(w)$
5: **fin pour**
6: **fin pour**
7: **retourner** G^{-1}

Composantes fortement connexes

Exemple



v :	a	b	c	d	e	f	g	h
$d(v)$	1	11	13	12	3	2	5	8
$f(v)$	10	16	14	15	4	7	6	9

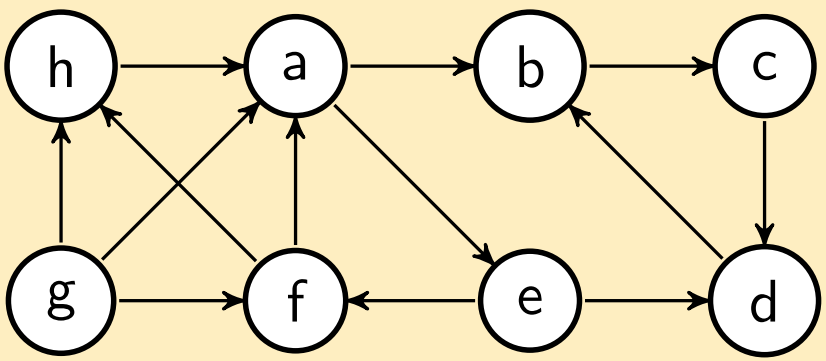


Ordre : b, d, c, a, h, f, g, e

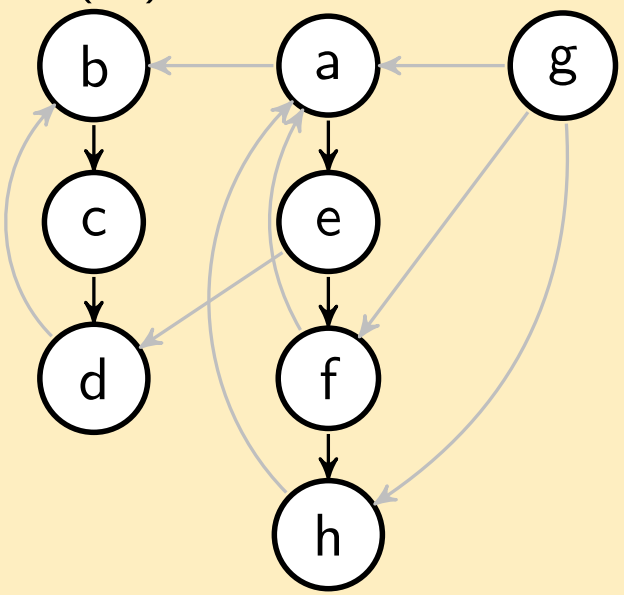
Composantes fortement connexes

Exemple (Suite)

On calcule d'abord le graphe inverse G^{-1}, puis on exécute $PP(G^{-1})$ en traitant dans la boucle 6 de $PP(G^{-1})$ les sommets dans l'ordre inverse de f obtenu lors de $PP(G)$.



Ordre : b, d, c, a, h, f, g, e



CFC : $\{b, c, d\}, \{a, e, f, h\}, \{g\}$.