Algorithmique des graphes

Cours 5 – Applications du parcours en profondeur

František Kardoš

frantisek.kardos@u-bordeaux.fr

Parcours en profondeur

Algorithme 1 : PP(G)

Données : graphe G

```
1  début
2      pour chaque s sommet de G
         faire
3          couleur[s] ← BLANC ;
4          π[s] ← NULL ;
5      fin
6      temps ← 0 ;
7      pour chaque s sommet de G
         faire
8          si couleur[s] = BLANC
             alors
9              Visiter(s) ;
10         fin
11     fin
12 fin
```

Algorithme 2 : Visiter(u)

```
13 début
14     couleur[u] ← GRIS ;
15     d[u] ← temps ← temps + 1 ;
16     pour chaque v voisin (sortant)
         de u faire
17         si couleur[v] = BLANC
             alors
18             π[v] ← u ;
19             Visiter(v) ;
20         fin
21     fin
22     couleur[u] ← NOIR ;
23     f[u] ← temps ← temps + 1 ;
24 fin
```

Parcours en profondeur – graphes orientés

Il existe plusieurs types d'arcs :

- *arc de liaison* relie un sommet et son père (dans le sens père $\rightarrow$ fils)

Parcours en profondeur – graphes orientés

Il existe plusieurs types d'arcs :

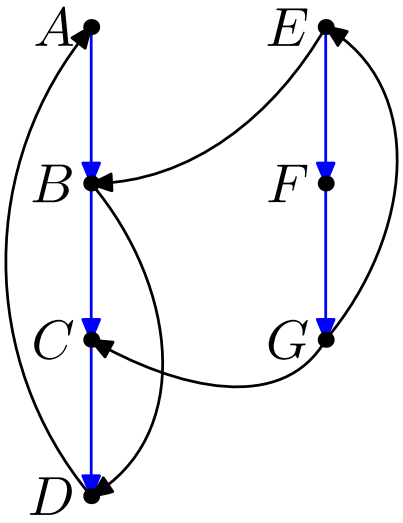
- *arc de liaison* relie un sommet et son père (dans le sens père $\rightarrow$ fils)
- *arc de retour* relie un sommet à son ancêtre (dans le sens descendant $\rightarrow$ ancêtre) – c'est un arc qui permet de remonter dans l'arbre de liaison
- *arc d'avant* relie un sommet à son descendant autre que fils (dans le sens ancêtre $\rightarrow$ descendant) – c'est un arc qui permet de redescendre dans l'arbre de liaison

Parcours en profondeur – graphes orientés

Il existe plusieurs types d'arcs :

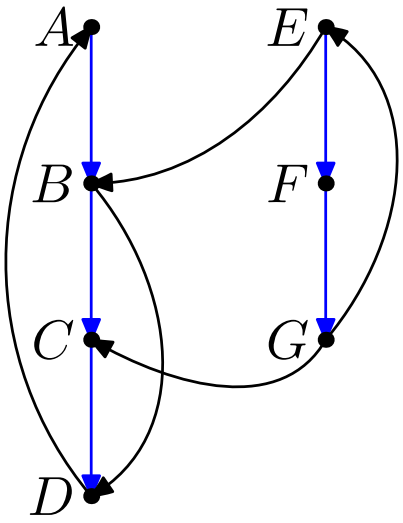
- *arc de liaison* relie un sommet et son père (dans le sens père $\rightarrow$ fils)
- *arc de retour* relie un sommet à son ancêtre (dans le sens descendant $\rightarrow$ ancêtre) – c'est un arc qui permet de remonter dans l'arbre de liaison
- *arc d'avant* relie un sommet à son descendant autre que fils (dans le sens ancêtre $\rightarrow$ descendant) – c'est un arc qui permet de redescendre dans l'arbre de liaison
- *arc transverse* relie un sommet à tout autre sommet dont la visite a bien été déjà terminée

Parcours en profondeur – graphes orientés



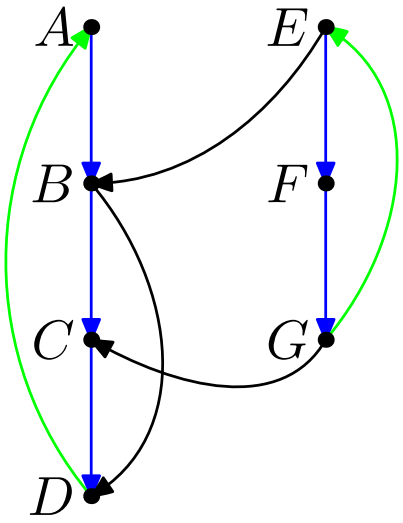
arcs de liaison

Parcours en profondeur – graphes orientés



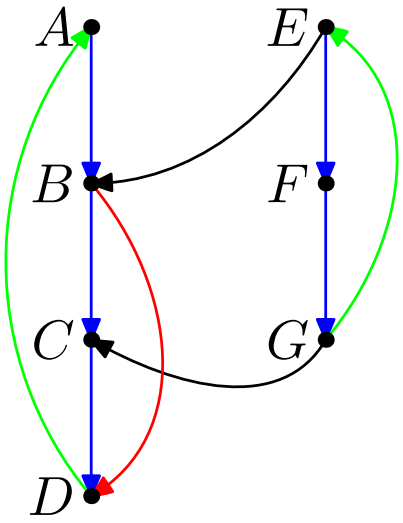
arcs de liaison
arcs de retour
arcs d'avant
arcs transverses

Parcours en profondeur – graphes orientés



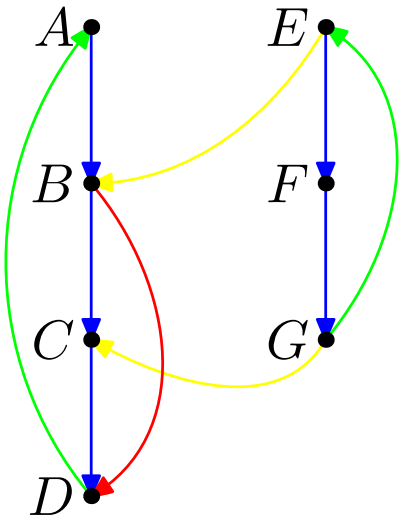
arcs de liaison
arcs de retour
arcs d'avant
arcs transverses

Parcours en profondeur – graphes orientés



arcs de liaison
arcs de retour
arcs d'avant
arcs transverses

Parcours en profondeur – graphes orientés



arcs de liaison
arcs de retour
arcs d'avant
arcs transverses

Parcours en profondeur : Applications

On peut utiliser un parcours en profondeur dans un graphe orienté pour détecter la présence de circuits :

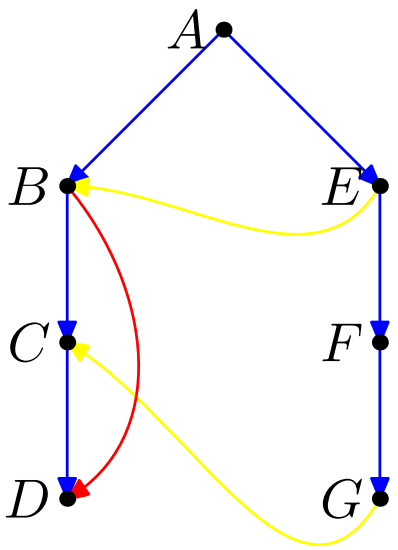
- Dans un graphe orienté, chaque arc de retour témoigne d'un circuit
- Pour chaque circuit, il y a au moins un arc de retour

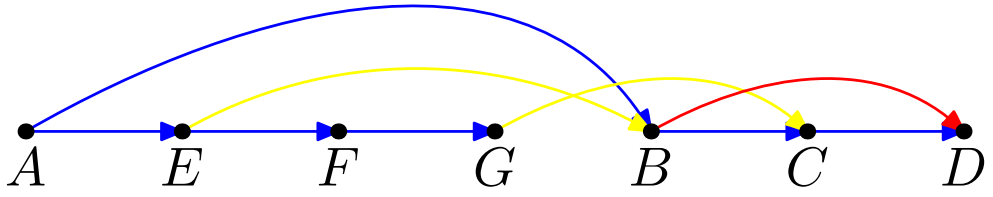
Parcours en profondeur : Applications

Dans un graphe orienté sans circuit, un parcours en profondeur permet d'établir un *tri topologique* – trouver un ordre linéaire des sommets tel que tout arc suive la même direction (de gauche à droite) :

Parcours en profondeur : Applications

Dans un graphe orienté sans circuit, un parcours en profondeur permet d'établir un *tri topologique* – trouver un ordre linéaire des sommets tel que tout arc suive la même direction (de gauche à droite) : il suffit d'ordonner les sommets par l'ordre décroissant de la fin de visite.





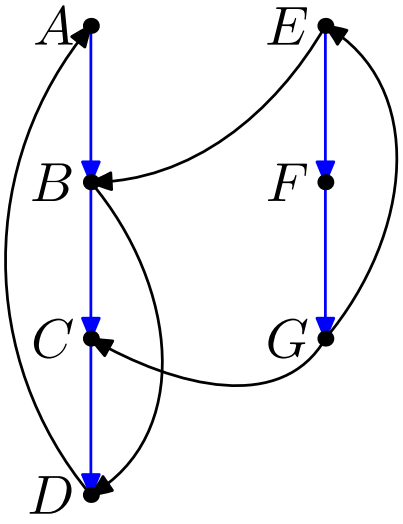
Définitions

Un graphe orienté est dit *fortement connexe* si ses sommets sont tous accessibles entre eux, i.e., pour toute paire de sommets u et v il existe un chemin de u à v et aussi un chemin de v à u.

Définitions

Un graphe orienté est dit *fortement connexe* si ses sommets sont tous accessibles entre eux, i.e., pour toute paire de sommets u et v il existe un chemin de u à v et aussi un chemin de v à u.

Le graphe ci-dessous est-il fortement connexe ?



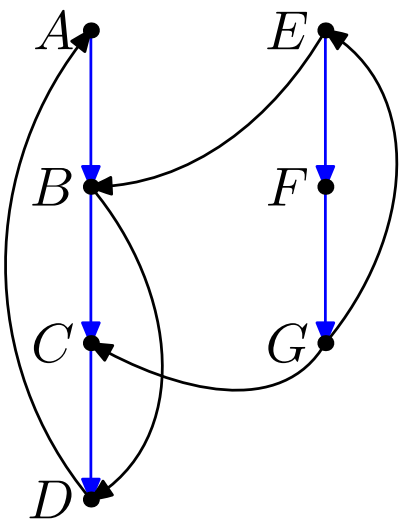
Définitions

Dans un graphe orienté, une *composante fortement connexe* est un sous-ensemble X de sommets de G induisant un graphe fortement connexe, X étant maximal par rapport à la connexité forte.

Définitions

Dans un graphe orienté, une *composante fortement connexe* est un sous-ensemble X de sommets de G induisant un graphe fortement connexe, X étant maximal par rapport à la connexité forte.

Quelles sont les composantes fortement connexes ?



Quelques observations

Vrai ou faux ?

- Soit uv un arc d'une composante fortement connexe. Alors il existe un chemin de v à u.

Quelques observations

Vrai ou faux ?

- Soit uv un arc d'une composante fortement connexe. Alors il existe un chemin de v à u.
- Soit uv un arc d'une composante fortement connexe. Alors uv est contenu dans un circuit.

Quelques observations

Vrai ou faux ?

- Soit uv un arc d'une composante fortement connexe. Alors il existe un chemin de v à u.
- Soit uv un arc d'une composante fortement connexe. Alors uv est contenu dans un circuit.
- Soit u et v deux sommets d'un graphe orienté. Si u et v appartiennent au même circuit, alors u et v sont dans la même composante fortement connexe.

Quelques observations

Vrai ou faux ?

- Soit uv un arc d'une composante fortement connexe. Alors il existe un chemin de v à u.
- Soit uv un arc d'une composante fortement connexe. Alors uv est contenu dans un circuit.
- Soit u et v deux sommets d'un graphe orienté. Si u et v appartiennent au même circuit, alors u et v sont dans la même composante fortement connexe.
- Soit u et v deux sommets d'un graphe orienté. Si u et v sont dans la même composante fortement connexe, alors u et v appartiennent au même circuit.

Quelques observations

Vrai ou faux ?

- Soit uv un arc d'une composante fortement connexe. Alors il existe un chemin de v à u.
- Soit uv un arc d'une composante fortement connexe. Alors uv est contenu dans un circuit.
- Soit u et v deux sommets d'un graphe orienté. Si u et v appartiennent au même circuit, alors u et v sont dans la même composante fortement connexe.
- Soit u et v deux sommets d'un graphe orienté. Si u et v sont dans la même composante fortement connexe, alors u et v appartiennent au même circuit. FAUX !

Calcul de composantes fortement connexes

Lemme

Soit G un graphe orienté, soit C un circuit dans G. Soit G/C le graphe obtenu en réduisant C en un seul sommet, disons c. Alors G/C a autant de composantes connexes que G, en plus,

- *Si une composante X ne couvre pas C, alors X est une composante de G/C ;*
- *Si X est une composante couvrant C, alors $X \setminus V(C) \cup \{c\}$ est une composante de G/C.*

Note. En réduisant un circuit C, on peut supprimer les boucles et pour tout ensemble d'arcs parallèles il suffit d'en garder un.

Calcul de composantes fortement connexes

On applique le lemme tant que le graphe contient encore des circuits.

Calcul de composantes fortement connexes

On applique le lemme tant que le graphe contient encore des circuits.

Le graphe final est un graphe sans circuit, dont tout sommet représente une composante fortement connexe du graphe de départ.

Calcul de composantes fortement connexes

On applique le lemme tant que le graphe contient encore des circuits.

Le graphe final est un graphe sans circuit, dont tout sommet représente une composante fortement connexe du graphe de départ.

On l'appelle le *graphe des composantes fortement connexes* $CFC(G)$.

Tester la connexité forte avec Parcours en profondeur

Calculer la composante fortement connexe d'un sommet u :

Idée : Pour un sommet u, déterminer

- l'ensemble $Desc(u)$ des sommets accessibles depuis u
- l'ensemble $Anc(u)$ des sommets u est accessible depuis

avec deux parcours en profondeur, un sur G, et l'autre sur G^{-1}.

La CFC(u) est alors $Desc(u) \cap Anc(u)$.

Parcours en profondeur et CFC

Soit G un graphe orienté, soit X une CFC de G. Soit x le premier sommet de X découvert par un parcours de G en profondeur. Alors x est l'ancêtre commun de tous les sommets de X. Autrement dit, tous les sommets de X sont découverts et visités avant la fin de la visite de x. Par ailleurs, si

$$d(x) = \min\{d(u) : u \in X\},$$

alors

$$f(x) = \max\{f(u) : u \in X\}.$$

Parcours en profondeur et CFC

Soit X_1 et X_2 deux CFC d'un graphe orienté G. Notons x_1 (resp. x_2) le premier sommet de X_1 (X_2) découvert par un parcours de G en profondeur.

- Si $X_1 \rightarrow X_2$ dans $CFC(G)$, alors x_1 est un ancêtre de x_2, $I(x_1) \supset I(x_2)$.
- Si $X_1 \leftarrow X_2$ ou X_1 et X_2 ne sont pas du tout reliés par un arc dans $CFC(G)$, alors $I(x_1) \cap I(x_2) = \emptyset$.

Algorithme de Kosaraju

Soit G un graphe orienté.

1. Exécuter un parcours en profondeur de G.
2. Exécuter un parcours en profondeur sur le graphe transposé G^{-1} en explorant les sommets dans l'ordre décroissant de la fin de visite du premier parcours.

Les arborescences produites par le second parcours sont les CFC de G.

Algorithme de Kosaraju – exemple

$A : B, G$
$B : E$
$C :$
$D : F$
$E : D, G$
$F : E$
$G : B, C$

Algorithme de Kosaraju – complexité

Soit G un graphe orienté.

1. Exécuter un parcours en profondeur de G – $O(m+n)$.

(*) Calculer le graphe transposé G^{-1} – $O(m+n)$ si graphe représenté par listes de successeurs.

2. Exécuter un parcours en profondeur de G^{-1} – $O(m+n)$.

La complexité de l'algorithme de Kosaraju est donc $O(m+n)$, la même que celle d'un parcours en profondeur.

Algorithme de Tarjan

permet de calculer les CFC en un seul parcours en profondeur.

Pendant le parcours en profondeur de G, on utilise un tableau supplémentaire $t[\,]$ pour calculer, pour chaque sommet de G, le sommet le plus haut (l'ancêtre le plus ancien) vers lequel il est possible de remonter grâce à des arcs de retour.

Algorithme de Tarjan

permet de calculer les CFC en un seul parcours en profondeur.

Pendant le parcours en profondeur de G, on utilise un tableau supplémentaire $t\,[\,]$ pour calculer, pour chaque sommet de G, le sommet le plus haut (l'ancêtre le plus ancien) vers lequel il est possible de remonter grâce à des arcs de retour.

Si jamais pour un sommet u à la fin de sa visite on a $t[u] = u$, on sais que u est la racine d'une composante fortement connexe.

Algorithme de Tarjan – implémentation

Dans $t[\,]$, au lieu de stocker
le sommet le plus haut (l'ancêtre le plus ancien) vers lequel il est possible de remonter grâce à des arcs de retour,
on va stocker
le début de visite du sommet le plus haut (l'ancêtre le plus ancien) vers lequel il est possible de remonter grâce à des arcs de retour.

Algorithme de Tarjan – implémentation

Dans $t[\,]$, au lieu de stocker
le sommet le plus haut (l'ancêtre le plus ancien) vers lequel il est possible de remonter grâce à des arcs de retour,
on va stocker
le début de visite du sommet le plus haut (l'ancêtre le plus ancien) vers lequel il est possible de remonter grâce à des arcs de retour.

Pour calculer $t[u]$, au début de visite de u, on l'initialise à $d[u]$.
On le met à jour à la fin de visite de chacun de ses fils, et aussi lors d'une découverte d'un arc de retour.

Algorithme de Tarjan – implémentation

Algorithme 1 : DFS(G)

Données : graphe G

```
1  début
2      pour chaque s sommet de G
         faire
3          couleur[s] ← BLANC ;
4          π[s] ← NULL ;
5      fin
6      temps ← 0 ;
7      pour chaque s sommet de G
         faire
8          si couleur[s] = BLANC
             alors
9              Visiter(s) ;
10         fin
11     fin
12 fin
```

Algorithme 2 : Visiter(u)

```
13 début
14     couleur[u] ← GRIS ;
15     d[u] ← temps ← temps + 1 ;
16     pour chaque v voisin de u
         faire
17         si couleur[v] = BLANC
             alors
18             π[v] ← u ;
19             Visiter(v) ;
20         fin
21     fin
22     couleur[u] ← NOIR ;
23     f[u] ← temps ← temps + 1 ;
24 fin
```

Algorithme de Tarjan – implémentation

Algorithme 1 : DFS(G)

Données : graphe G

```
1  début
2      pour chaque s sommet de G
        faire
3          couleur[s] ← BLANC ;
4          π[s] ← NULL ;
5      fin
6      temps ← 0 ;
7      pour chaque s sommet de G
        faire
8          si couleur[s] = BLANC
            alors
9              Visiter(s) ;
10         fin
11     fin
12 fin
```

Algorithme 2 : Visiter(u)

```
13 début
14     couleur[u] ← GRIS ;
15     d[u] ← temps ← temps + 1 ;
16     t[u] ← d[u] ;
17     pour chaque v voisin de u faire
18         si couleur[v] = BLANC alors
19             π[v] ← u ;
20             Visiter(v) ;
21             t[u] ← min(t[u], t[v]);
22         fin
23         sinon
24             si couleur[v] = GRIS alors
25                 t[u] ← min(t[u], d[v]);
26             fin
27         fin
28     fin
29     couleur[u] ← NOIR ;
30     f[u] ← temps ← temps + 1 ;
31 fin
```

Algorithme de Tarjan – implémentation

Algorithme 1 : DFS(G)

Données : graphe G

```
1  début
2      pour chaque s sommet de G
        faire
3          couleur[s] ← BLANC ;
4          π[s] ← NULL ;
5      fin
6      temps ← 0 ;
7      CFC ← [ ] ;
8      L ← [ ] ;
9      pour chaque s sommet de G
        faire
10         si couleur[s] = BLANC
            alors
11             Visiter(s) ;
12         fin
13     fin
14 fin
```

Algorithme 2 : Visiter(u)

```
15 début
16     couleur[u] ← GRIS ;
17     d[u] ← temps ← temps + 1 ;
18     t[u] ← d[u] ;
19     L.append(u) ;
20     pour chaque v voisin de u faire
21         si couleur[v] = BLANC alors
22             π[v] ← u ;
23             Visiter(v) ;
24             t[u] ← min(t[u], t[v]);
25         fin
26         sinon
27             si v in L alors
28                 t[u] ← min(t[u], d[v]);
29             fin
30         fin
31     fin
32     couleur[u] ← NOIR ;
33     f[u] ← temps ← temps + 1 ;
34     si t[u] = d[u] alors
35         CFC.append(L[u : ]);
36         L ← L[ : u];
37     fin
38 fin
```

Algorithme de Tarjan – exemple

u	: $Succ(u)$	$d[u]$	$f[u]$	$t[u]$
A	: B			
B	: D			
C	: A			
D	: C, E, F, H			
E	: G			
F	: E			
G	: F			
H	: I			
I	: J			
J	: H, K			
K	:			

Algorithme de Tarjan – exemple

u	: $Succ(u)$	$d[u]$	$f[u]$	$t[u]$
A	: B	1	21	1
B	: D	2	20	1
C	: A	4	5	1
D	: C, E, F, H	3	20	1
E	: G	6	11	6
F	: E	8	9	6
G	: F	7	10	6
H	: I	12	19	12
I	: J	13	18	12
J	: H, K	14	17	12
K	:	15	16	15

CFC :
$[E, G, F]$
$[K]$
$[H, I, J]$
$[A, B, D, C]$