

3I003 – Algorithmique  
Cours 4 : Parcours de graphes

Année 2018-2019

Responsables et chargés de cours

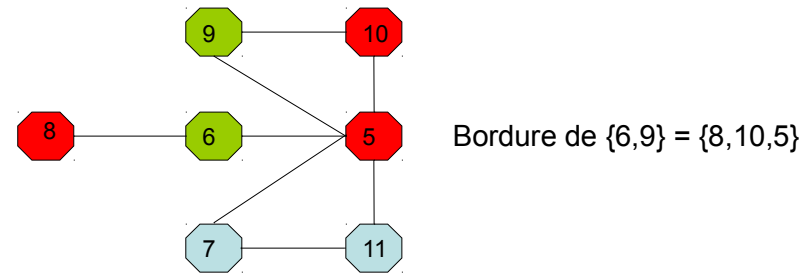
Fanny Pascual  
Olivier Spanjaard

# Parcours : définitions et notations

Soit  $G=(S,A)$  un graphe.

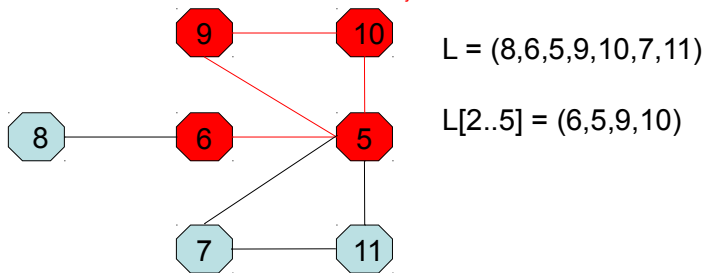
$B(T)$  : Bordure de  $T \subset S$

Sous-ensemble des sommets de  $S-T$  dont au moins un voisin est dans  $T$ .



Soit  $L=(s_1,s_2,\dots,s_p)$  une liste de sommets distincts :

$L[i..j]$  est la sous-liste  $(s_i,\dots,s_j)$



Parcours du graphe  $G=(S,A)$  :

liste  $L=(s_1,s_2,\dots,s_n)$  des  $n$  sommets de  $G$  telle que :

pour tout  $i=2..n$ , le sommet  $s_i$  appartient à la bordure de  $\{s_1,s_2,\dots,s_{i-1}\}$ , si cette bordure n'est pas vide.

Soit  $L=(s_1,s_2,\dots,s_n)$  un parcours de  $G$ .

Si  $B(\{s_1,s_2,\dots,s_{i-1}\})$  est vide, alors  $s_i$  est appelé point de régénération de  $L$ .

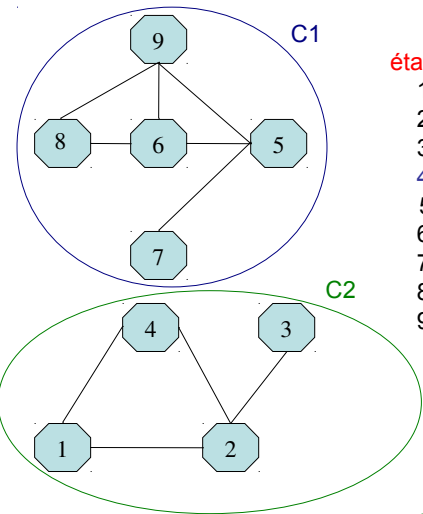
Par convention,  $s_1$  est un point de régénération.

L'étape  $i$  du parcours consiste à ajouter  $s_i$  (nouveau sommet visité) à la sous-liste  $L[1..i-1]$  des sommets déjà visités.

A la fin de l'étape  $i$  :

un sommet de  $L[1..i]$  est dit ouvert s'il possède au moins un voisin non visité ; dans le cas contraire, il est dit fermé.

Exemple (graphe non orienté) :



étape i	L[1..i]	B(L[1..i])
1	(1)	(2,4)
2	(1,2)	(3,4)
3	(1,2,4)	(3)
4	(1,2,4,3)	∅
5	(1,2,4,3,8)	(6,9)
6	(1,2,4,3,8,6)	(5,9)
7	(1,2,4,3,8,6,5)	(9,7)
8	(1,2,4,3,8,6,5,7)	(9)
9	(1,2,4,3,8,6,5,7,9)	∅
	C1	C2

A la fin de l'étape 3, les sommets visités 1 et 4 sont fermés, le sommet visité 2 est ouvert.

Propriétés des parcours : connexité

Soit  $L=(s_1,s_2,\dots,s_n)$  un parcours de G.  
Soit  $(i_1,\dots,i_r)$  les indices des points de régénération de L.

$G[i_1..i_2-1], G[i_2..i_3-1],\dots, G[i_r..n]$  sont les sous-graphes induits par les composantes connexes de G ;  
 $L[i_1..i_2-1], L[i_2..i_3-1],\dots, L[i_r..n]$  sont des parcours des sous-graphes induits par les composantes connexes de G.

Preuve: simple par récurrence sur le nombre r de points de régénération.

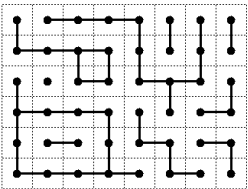
Remarque : un parcours de G permet donc de calculer les composantes connexes de G.

Application en infographie

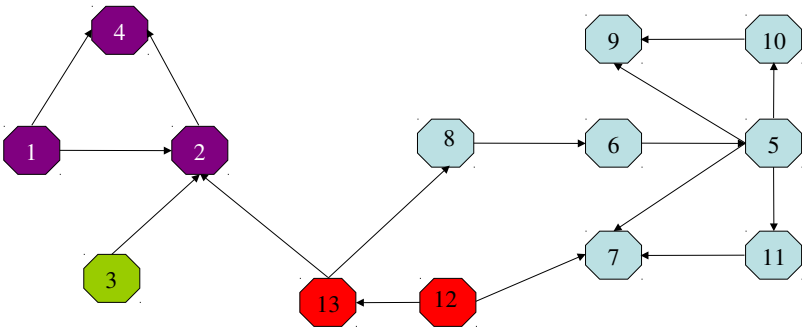
L'algorithme de remplissage par diffusion est un algorithme classique en infographie qui change la couleur d'un ensemble connexe de pixels de même couleur délimités par des contours.



A	C	C	C	C	A	C	B
A	A	A	A	C	A	C	B
B	C	A	A	C	C	C	A
B	B	B	B	A	C	A	A
B	C	C	B	A	A	C	C
B	B	B	B	B	A	A	C



Exemple (graphe orienté) :



Un parcours :  $L=(1,2,4,3,8,6,5,9,10,7,11,12,13)$

Points de régénération :  $\{1,3,8,12\}$

## Forêt couvrante associée à un parcours.

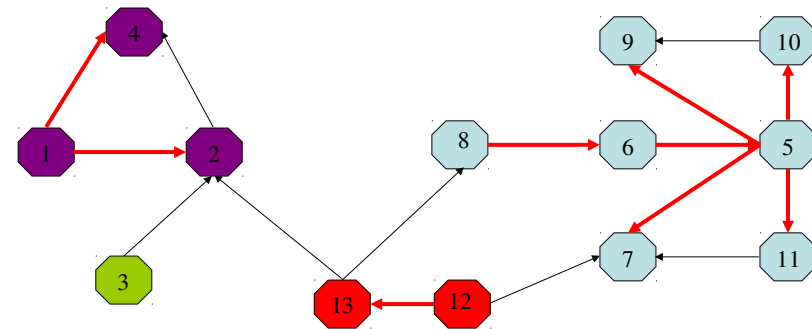
Soit  $L=(s_1,s_2,\dots,s_n)$  un parcours de  $G$ .

On choisit pour chaque sommet  $s_j$  (non point de régénération) un prédécesseur  $s_k$  avec  $1 \leq k \leq j-1$ .

On note alors  $père_L(s_j)$  le prédécesseur de  $s_j$  choisi.

Le graphe partiel de  $G$  formé des arêtes (arcs)  $(père_L(s_j),s_j)$  est une forêt couvrante de  $G$  notée  $F(L)$ .

$F(L)$  est formée de  $r$  arborescences  $A_1(L), A_2(L), \dots, A_r(L)$  où  $A_k(L)$  est une arborescence dont la racine est le  $k^{ième}$  point de régénération.



Un parcours :  $L= (1,2,4,3,8,6,5,9,10,7,11,12,13)$

Points de régénération :  $\{1,3,8,12\}$

Le graphe partiel des arcs rouges est une forêt  $F(L)$  couvrante de  $G$ .

## Parcours en largeur Parcours en profondeur

Soit  $L=(s_1,s_2,\dots,s_n)$  un parcours de  $G$ .

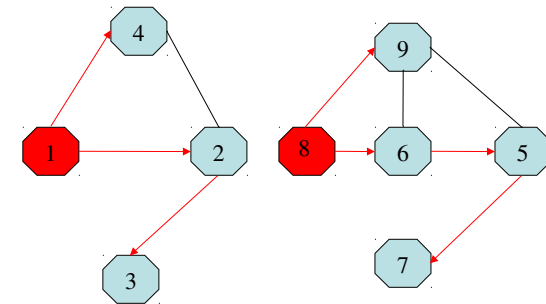
$L$  est un **parcours en largeur** de  $G$  si tout sommet visité  $s_i$ , qui n'est pas un point de régénération, est **voisin du premier sommet visité ouvert** de  $L[1..i-1]$ .

**Remarque :**  $père_L(s_i)$  est le premier sommet visité ouvert de  $L[1..i-1]$ .

$L$  est un **parcours en profondeur** de  $G$  si tout sommet visité  $s_i$ , qui n'est pas un point de régénération, est **voisin du dernier sommet visité ouvert** de  $L[1..i-1]$ .

**Remarque :**  $père_L(s_i)$  est le dernier sommet visité ouvert de  $L[1..i-1]$ .

Exemple de parcours en largeur

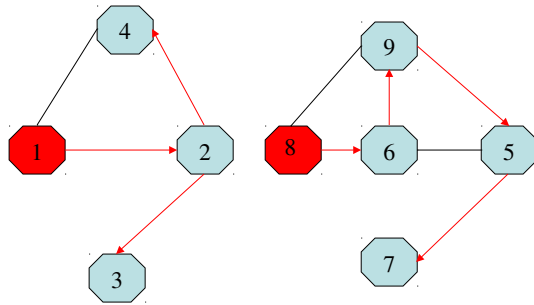


(1)  
(1,2)  
(1,2,4)  
(1,2,4,3)  
(1,2,4,3,8)  
(1,2,4,3,8,6)  
(1,2,4,3,8,6,9)  
(1,2,4,3,8,6,9,5)  
(1,2,4,3,8,6,9,5,7)

Après chaque étape :

- premier sommet visité ouvert **en rouge** ;
- autres sommets visités ouverts **en vert**.

Exemple de parcours  
en profondeur



(1)  
(1,2)  
(1,2,4)  
(1,2,4,3)  
(1,2,4,3,8)  
(1,2,4,3,8,6)  
(1,2,4,3,8,6,9)  
(1,2,4,3,8,6,9,5)  
(1,2,4,3,8,6,9,5,7)

Après chaque étape :

- dernier sommet visité ouvert **en rouge** ;
- autres sommets visités ouverts **en vert**.

## Récapitulatif (graphe orienté)

- La **bordure**  $B(T)$  d'un sous-ensemble  $T$  de sommets est constituée de l'ensemble des sommets de  $S-T$  dont **au moins un prédécesseur est dans  $T$** .
- **Parcours** : rangement  $L = (s_1, \dots, s_n)$  des sommets du graphe où tout sommet  $s_i \in B(\{s_1, \dots, s_{i-1}\})$  ou  $B(\{s_1, \dots, s_{i-1}\}) = \emptyset$
- Un sommet  $s_i$  tel que  $B(\{s_1, \dots, s_{i-1}\}) = \emptyset$  est un **point de régénération** (par convention,  $s_1$  est un point de régénération)
- On choisit **pour chaque sommet  $s_j$**  (non point de régénération) un **prédécesseur**  $s_k$  avec  $1 \leq k \leq j-1$ . On note alors  **$\text{père}_L(s_j)$**  le prédécesseur de  $s_j$  choisi.
- Le graphe partiel de  $G$  formé des arcs  **$(\text{père}_L(s_j), s_j)$**  est une **forêt couvrante** de  $G$  notée  **$F(L)$** .

## Récapitulatif (graphe orienté)

- Un sommet  $s_i$  est dit **ouvert** dans  $\{s_1, \dots, s_k\}$  si il comporte un successeur dans  $S - \{s_1, \dots, s_k\}$ .
- Parcours en **largeur** : tout sommet  $s_i$  (non point de régénération) est successeur du premier sommet ouvert dans  $\{s_1, \dots, s_{i-1}\}$ .
- Parcours en **profondeur** : tout sommet  $s_i$  (non point de régénération) est successeur du dernier sommet ouvert dans  $\{s_1, \dots, s_{i-1}\}$ .
- A un parcours en largeur ou en profondeur correspond **une unique forêt couvrante**.

## Algorithme de parcours en largeur (graphe orienté)

```

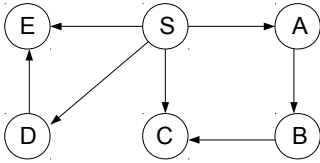
Procédure Largeur (G)
début
    pour chaque sommet  $s$  de  $G$  faire
        visité[s] := 0;
    compteur := 0;
    pour chaque sommet  $s$  de  $G$  faire
        si visité[s] == 0 alors Explorer (G, s);
fin

Procédure Explorer (G, s)
début
    F := FileVide(); compteur++; visité[s] := compteur; Enfiler (F, s);
    tant que F n'est pas vide faire
        x := Defiler (F);
        pour chaque  $y$  successeur de  $x$  faire
            si visité[y] == 0 alors
                compteur := compteur + 1; visité[y] := compteur;
                Enfiler (F, y);
fin
    
```

compteur est une  
variable globale.

A la fin de **Explorer (G, s)**, on a  **$\text{visité}[s'] > 0$  pour tout sommet  $s'$  accessible** à partir de  $s$ . L'algorithme numérote chaque sommet  $s$  par son rang dans le parcours en largeur, qui est indiqué dans  **$\text{visité}[s]$** .

## Exemple

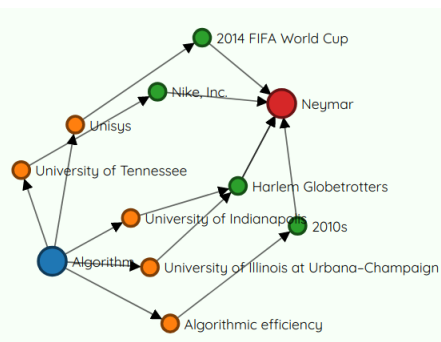
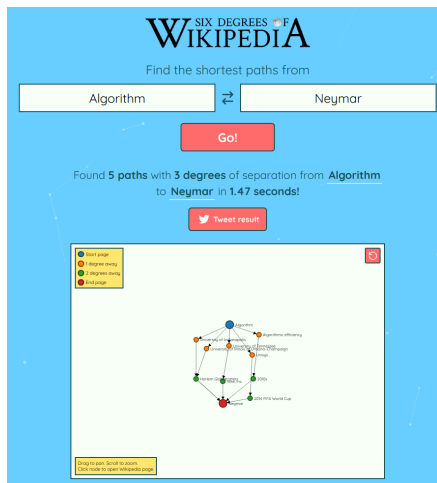


Ordre de visite des sommets	F après visite du sommet
	[S]
S	[A C D E]
A	[C D E B]
C	[D E B]
D	[E B]
E	[B]
B	[ ]

## Analyse de complexité

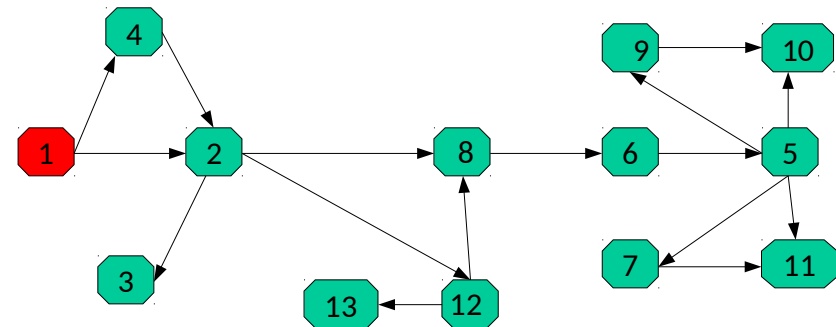
- On suppose une représentation du graphe G par des **listes de successeurs**
- Il y a au plus un appel **Explorer**(G, s) par sommet s, du fait du tableau **visité**
- Lors d'un appel **Explorer**(G, s), il y a les pas suivants :
  - Des opérations en temps constants (création de **F**, incrémentation de **compteur**, mise à jour de **visité[s]**, insertion dans **F**)
  - Une boucle **pour** qui scanne les arcs issus de s, imbriquée dans une boucle **tant que** sur les sommets s
    - La complexité cumulée du **pas 1**, en comptant tous les appels, est  $O(n)$  où n le nb de sommets de G, car il y a au plus n appels à la procédure **Explorer**
    - La complexité cumulée du **pas 2**, en comptant tous les appels, est  $O(n+m)$  où m le nb d'arcs de G, puisque chaque sommet de G est inséré une seule fois dans la file **F** (du fait de la condition **visité[t]==0**) et chaque arc de G est examiné une fois
- La complexité globale de l'algorithme est donc  $O(n+m)$
- Remarque** : la complexité cumulée en  $O(n+m)$  du pas 2 est liée à l'utilisation de listes de successeurs

## Six degrés de Wikipédia



Déterminer la distance minimale entre deux sommets peut se faire à l'aide d'un parcours en largeur.

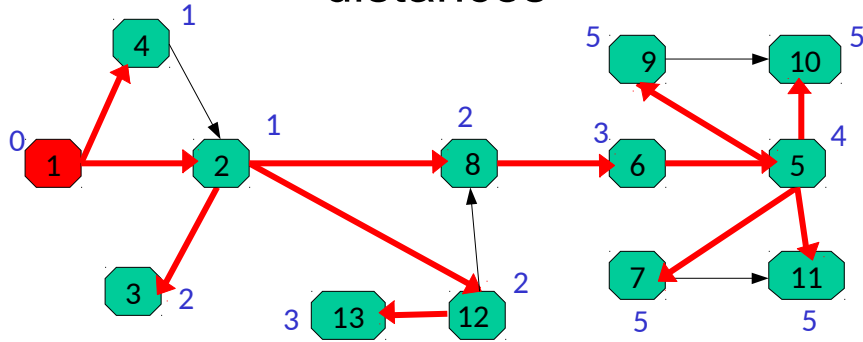
## Exemple de calcul des distances



$d[x]$  : longueur d'un chemin min en nombre d'arcs de 1 à x  
 Calcul de  $d[x]$  : lors de la visite du sommet x, on fait  

$$d[x] = d[\text{père}_L(x)] + 1.$$

## Exemple de calcul des distances



Parcours en largeur : (1,2,4,3,8,12,6,13,5,7,9,10,11)

$d[x]$  : longueur d'un chemin min en nombre d'arcs de 1 à  $x$

Calcul de  $d[x]$  : lors de la visite du sommet  $x$ , on fait

$$d[x] = d[\text{père}_L(x)] + 1.$$

## Pseudo-code et idée de la preuve

On suppose que le tableau  $d$  est initialisé à  $d[x] := -1$  pour tout sommet  $x$ . On cherche à déterminer les distances en nombre d'arcs depuis un sommet  $s$  racine du graphe orienté  $G$ .

```

Procédure Distance( $G, s$ )
début
     $F := \text{FileVide}()$ ;  $d[s] := 0$ ; Enfiler( $F, s$ );
    tant que  $F$  n'est pas vide faire
         $x := \text{Defiler}(F)$ ;
        pour chaque  $y$  successeur de  $x$  faire
            si  $d[y] == -1$  alors
                 $d[y] := d[x] + 1$ ; Enfiler( $F, y$ );
fin
    
```

**Idée de la preuve.** On peut prouver par récurrence la propriété suivante :

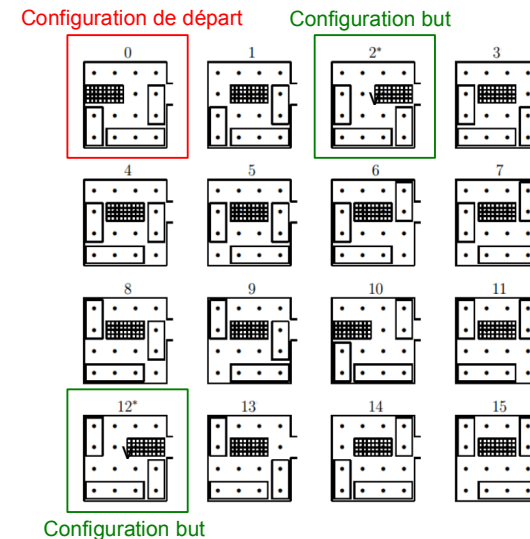
- Pour chaque  $\delta = 0, 1, \dots$ , il y a une étape de l'algorithme au terme de laquelle
- (1) tous les sommets à distance  $\leq \delta$  de  $s$  ont leur distance correctement fixée,
  - (2) tous les autres sommets ont leur distance fixée à  $-1$ ,
  - (3) la file contient exactement les sommets à distance  $\delta$ .

## Résolution de Rush hour<sup>®</sup>



**Objectif :** minimiser le nombre de déplacements de véhicules pour faire sortir la voiture rouge de l'embouteillage.

## Configurations possibles

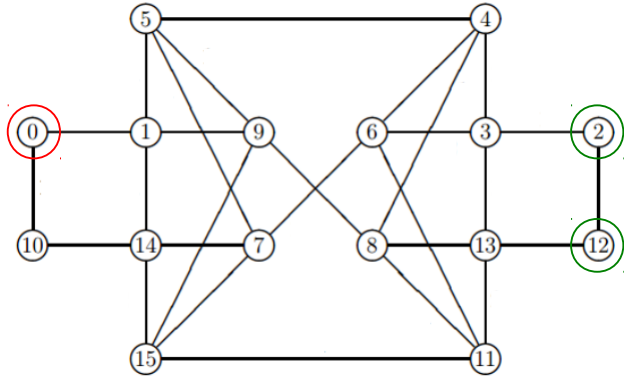


# Graphe des configurations

Graphe non-orienté  $G=(S,A)$  défini par :

$S = \{\text{configurations possibles}\}$

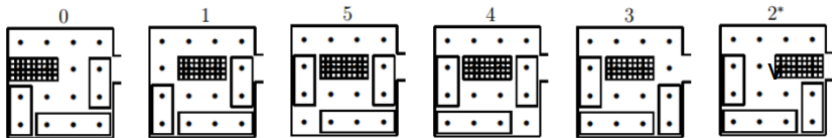
$A = \{\{i,j\}\} : \text{on peut passer de } i \text{ à } j \text{ ou de } j \text{ à } i \text{ en un déplacement de véhicule}\}$



On recherche une **plus courte chaîne** en nombre d'arêtes entre le **sommet 0** et le **sommet 2** ou le **sommet 12**.

## Une solution optimale

Le chemin de 0 à 2 dans la **forêt couvrante associée** au parcours en largeur est une solution optimale au problème.

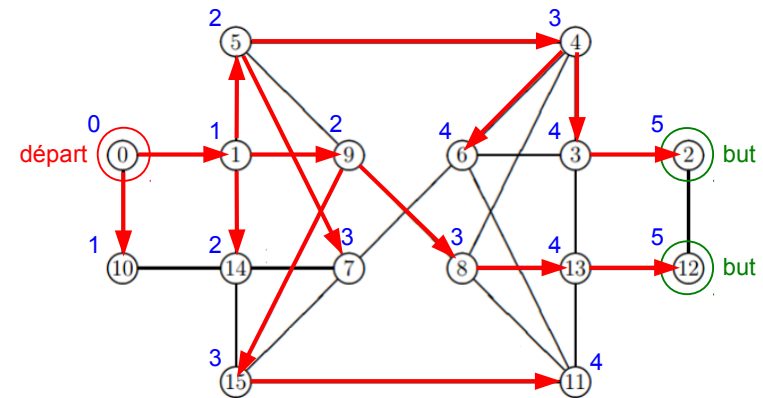


Les 40 configurations de départ proposées dans le jeu, **dont la grille est de taille fixée**, peuvent toutes être résolues par parcours en largeur.

**Remarque** : en pratique, le graphe des configurations est généré « à la volée », au fur et à mesure du parcours.

# Parcours en largeur

Parcours : (0, 1, 10, 5, 9, 14, 4, 7, 8, 15, 3, 6, 11, 13, 2, 12)



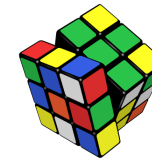
## Résolution de casse-têtes par parcours en largeur

**Autres exemples** de casse-têtes :

Taquin



Rubik's Cube



Le loup le chou et la chèvre



En notant **b** le **facteur de branchement** et **d** la **distance minimum** de la configuration de départ à une configuration but, et en se plaçant dans le cas où le graphe des configurations est un arbre, le nombre de configurations générées, et donc la **complexité**, s'écrit :

$$1 + b + b^2 + \dots + b^d = (b^{d+1} - 1) / (b - 1) \rightarrow O(b^d)$$

Autrement dit, approche praticable sur des **petits problèmes**.



# Algorithme de parcours en profondeur (graphe orienté)

```

Procédure Prof(G)
début
    pour chaque sommet s de G faire
        visité[s] := faux ;
    pour chaque sommet s de G faire
        si non visité[s] alors Explorer(s) ;
fin

Procédure Explorer(G,s)
début
    visité[s] := vrai ;
    prévisite(s) ;
    pour chaque t successeur de s faire {
        action éventuelle sur l'arc (s,t) ;
        si non visité[t] alors Explorer(G,t) ;
    }
    postvisite(s) ;
fin
    
```

A la fin de **Explorer**(G, s), on a **visité[v]=vrai pour tout sommet v accessible** à partir de s. Les procédures **prévisite** et **postvisite** sont optionnelles, et correspondent à des opérations qu'on réalise quand l'appel récursif sur un sommet débute, et quand il se termine.

## Prévisite et postvisite

Pour chaque sommet s, on va noter les moments de deux événements importants :

- Le moment du début de l'appel récursif **Explorer**(s)
- Le moment de la fin de l'appel récursif **Explorer**(s)

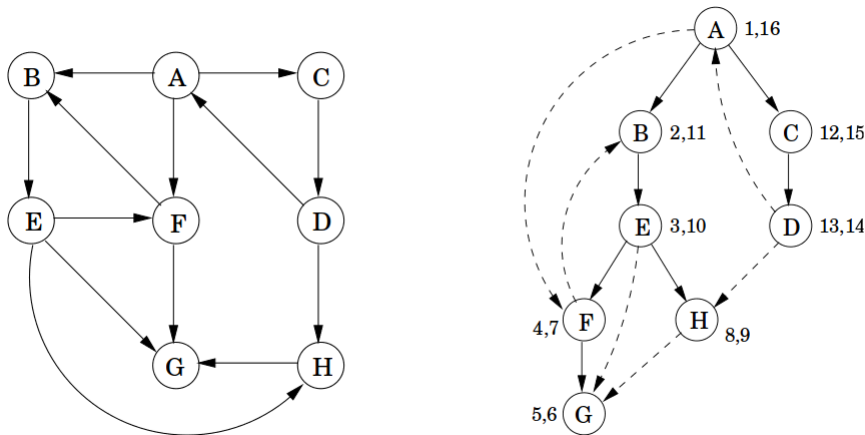
<b>Procédure</b> prévisite(s)	<b>Procédure</b> postvisite(s)
pre[s] := num	post[s] := num
num := num + 1	num := num + 1

(num est une variable globale initialisée à 1)

**Propriété.** A l'issue du parcours en profondeur, pour tout couple  $(s_1, s_2)$  de sommets :

- soit les deux intervalles  $[pre(s_1), post(s_1)]$  et  $[pre(s_2), post(s_2)]$  sont disjoints,
- soit l'un est contenu dans l'autre.

## Exemple



Les arcs en pointillés n'appartiennent pas à la forêt sous-jacente.

## Analyse de complexité

- On suppose une représentation du graphe G par des **listes de successeurs**
- Il y a un appel récursif **Explorer**(s) par sommet s, du fait du tableau **visité**
- Lors d'un appel récursif **Explorer**(s), il y a les pas suivants :
  1. Des opérations en temps constants + opérations de **pré/postvisite**
  2. Une boucle qui scanne les arcs issus de s
    - La complexité cumulée du **pas 1**, en comptant tous les appels récursifs, est  $O(n)$  où n le nb de sommets, puisque chaque sommet est visité une seule fois
    - La complexité cumulée du **pas 2**, en comptant tous les appels récursifs, est  $O(m)$  où m le nb d'arcs de G, puisque chaque arc du graphe G est examiné une fois
- La complexité globale de l'algorithme est donc  $O(n+m)$
- **Remarque** : la complexité cumulée en  $O(m)$  du pas 2 est liée à l'utilisation de listes de successeurs



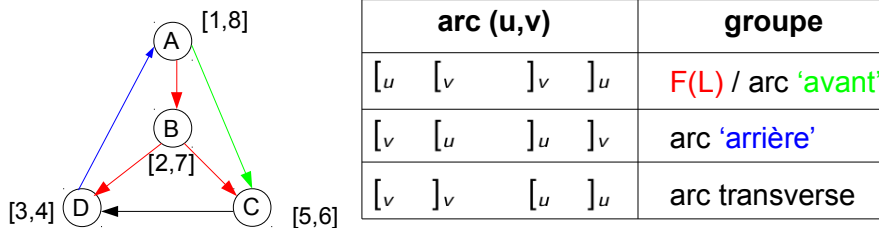
## Arcs associée à un parcours en profondeur

Soit  $L=(s_1,s_2,\dots,s_n)$  un **parcours en profondeur** de  $G$ .

Soit  $F(L)$  sa forêt sous-jacente.

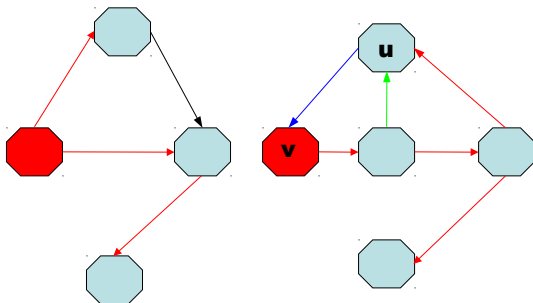
Les arcs de  $G$  sont classés en **4 groupes** :

- 1) les arcs de  **$F(L)$**  ;
- 2) les arcs '**avant**'  $(s_i,s_j)$  :  $s_j$  est un descendant de  $s_i$  dans  $F(L)$  ;
- 3) les arcs '**arrière**'  $(s_i,s_j)$  :  $s_i$  est un descendant de  $s_j$  dans  $F(L)$  ;
- 4) les arcs 'transverses'  $(s_i,s_j)$  : tous les autres arcs.



## Graphe sans circuit → pas d'arc arrière

Par contraposée : si il existe un **arc arrière**  **$(u,v)$** , on construit un circuit en concaténant  **$(u,v)$**  et le chemin de  $v$  à  $u$  dans  **$F(L)$**



## Existence d'un circuit

**Problème** :

Soit  $G=(S,A)$  un graphe orienté.

**Existe-t-il un circuit dans  $G$  ?**

**Existence de circuit** :

Soit  $L=(s_1,s_2,\dots,s_n)$  un **parcours en profondeur** de  $G$

et soit  $F(L)$  sa forêt sous-jacente.

$G$  est **sans circuit** si et seulement s'il **n'existe pas d'arc arrière pour  $L$** .

## Graphe avec circuit → arc arrière

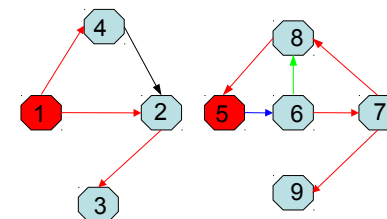
**Lemme des sommets accessibles**

Soit  $L=(s_1,s_2,\dots,s_n)$  un parcours en profondeur de  $G$ . Tous les sommets de  $\{s_{i+1},\dots,s_n\}$  accessibles à partir de  $s_i$  dans  $G$  seront ses descendants dans  $F(L)$ .

Soit  $C=(s_1,\dots,s_k)$ , un circuit dans  $G$ . Soit  $s_i$  le premier sommet de  $C$  visité dans le parcours en profondeur.

D'après le lemme des sommets accessibles, les autres sommets de  $C$  sont des descendants de  $s_i$  dans  $F(L)$ .

Donc  $(s_{i-1},s_i)$  est un arc arrière.

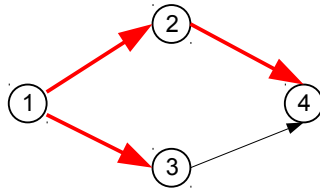


$C=(5,6,7,8)$   
6 visité en 1er  
 **$(5,6)$  arc arrière**

## Remarque importante

Le **lemme des sommets accessibles** n'est **pas** valide pour un **parcours en largeur**.

Exemple :



On considère le parcours en largeur  $L=(1,2,3,4)$ . La forêt couvrante associée  $F(L)$  est indiquée en rouge sur la figure.

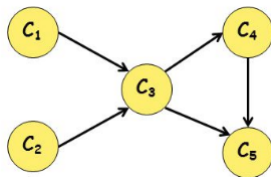
Le sommet 4 est accessible lorsque le sommet 3 est visité dans le parcours en largeur, et pourtant **le sommet 4 n'est pas descendant du sommet 3** dans  $F(L)$ .

## Ordonnancement

Ensemble de **5 UEs** :  $\{C_1, C_2, C_3, C_4, C_5\}$

- $C_1$  et  $C_2$  n'ont pas de prérequis
- Il vaut mieux avoir suivi  $C_1$  et  $C_2$  pour suivre  $C_3$
- Il vaut mieux avoir suivi  $C_3$  pour suivre  $C_4$
- Il vaut mieux avoir suivi  $C_3$  et  $C_4$  pour suivre  $C_5$

Le graphe représentant ces **contraintes de précédence** :



Un ordre réalisable dans lequel suivre les UEs :  $C_1, C_2, C_3, C_4, C_5$ .

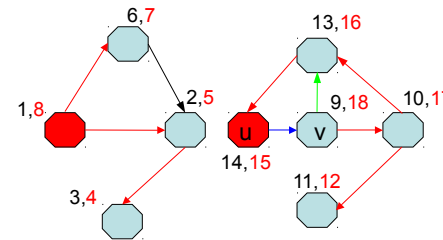
## Algorithme de détection de circuit

### Algorithme de détection de circuit

Pour détecter s'il existe un circuit dans un graphe il suffit donc de faire un parcours en profondeur de ce graphe et détecter si l'on trouve un arc arrière.

### Détection d'un arc arrière

A l'issue du parcours en profondeur, on parcourt les listes de successeurs (en  $O(n+m)$ ) pour détecter si il existe un arc  $(u,v)$  tel que  $\text{post}(u) < \text{post}(v)$ .



(les numérotations pré/postfixe sont indiquées sur le graphe)

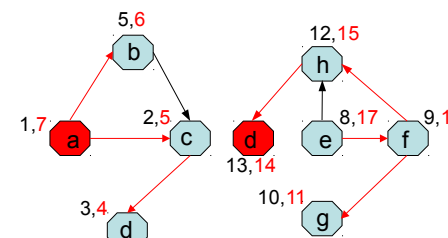
arc $(u,v)$	groupe
$[u \quad ]_v \quad ]_u$	$F(L)$ / arc 'avant'
$]_v \quad [u \quad ]_u \quad ]_v$	arc 'arrière'
$]_v \quad ]_v \quad [u \quad ]_u$	arc transverse

## Tri topologique

- **Rappel** : un **tri topologique** de  $G$  est un rangement des sommets de  $G$  tel que  $v$  figure après  $u$  pour tout arc  $(u,v)$  de  $G$ .
- Un tri topologique est obtenu en rangeant les sommets dans l'**ordre inverse** de la numérotation postfixe d'un parcours en profondeur. Ce rangement peut être obtenu en  $O(n+m)$  à l'aide d'une pile sur laquelle on empile chaque sommet  $s$  lorsque l'appel récursif **Explorer**( $G,s$ ) se termine.

### Propriété

Dans un graphe orienté sans circuit, pour tout arc  $(u,v)$ ,  $\text{post}(u) > \text{post}(v)$ .



(pour chaque sommet  $s$ ,  $\text{post}(s)$  est indiqué en rouge)

### tri topologique

e	f	h	d	g	a	b	c	d
17	16	15	14	11	7	6	5	4

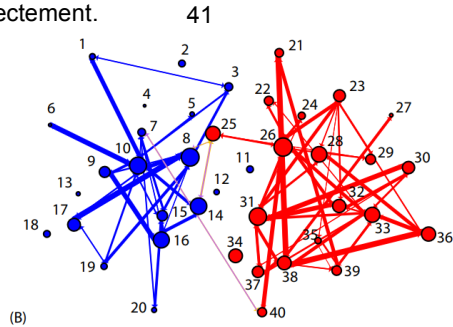
# Détection de communautés

L'identification de **composantes fortement connexes** dans un graphe orienté permet de détecter des communautés.

**Exemple** : réseaux sociaux

On considère le graphe dont les sommets sont les utilisateurs du réseau social, et on met un arc d'un utilisateur auteur d'une publication vers un utilisateur qui y a répondu

Une **grande composante fortement connexe** indique la présence d'une **communauté** où de nombreux utilisateurs interagissent, directement ou indirectement.



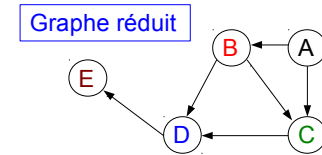
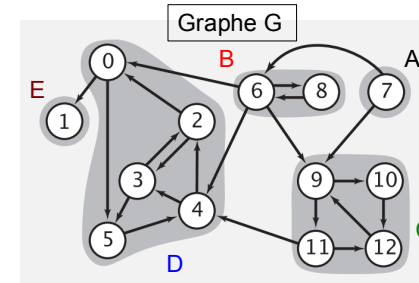
Analyse des interactions dans la blogosphère politique américaine lors de l'élection présidentielle de 2004 (Adamic et Glance, 2005).

# Détection des composantes fortement connexes

Algorithme pour déterminer les **composantes fortement connexes** (abrégées par CFC) par d'un graphe G.

**Composante fortement connexe terminale** : une CFC est terminale si le sommet correspondant dans le graphe réduit n'a pas de successeur.

**Idée** : faire un parcours en profondeur de G en faisant en sorte que chaque point de régénération appartienne à une composante fortement connexe terminale du sous-graphe des sommets non visités.

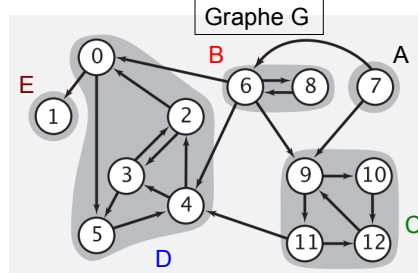
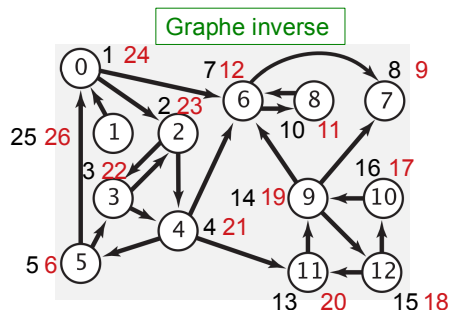


Un parcours en profondeur :  
(1, 0, 5, 4, 2, 3, 11, 12, 9, 10, 6, 8, 7)  
E D C B A

## Algorithme de Kosaraju-Sharir

L'algorithme comporte trois étapes pour un graphe G :

1. Déterminer le **graphe inverse** de G.
2. Faire un parcours en profondeur du graphe inverse en effectuant une **numérotation postfixe** des sommets.
3. Faire un parcours en profondeur de G en commençant par le sommet de **plus grand numéro postfixe** et en choisissant **comme point de régénération**, quand nécessaire, le sommet de plus grand numéro postfixe parmi les sommets non visités.



Parcours : (1, 0, 5, 4, 2, 3, 11, 12, 9, 10, 6, 8, 7)

## Analyse de complexité

**Etape 1** : La construction de la représentation du graphe inverse de G sous forme de listes de successeurs se fait en  $\Theta(n+m)$  si G est lui-même représenté sous forme de listes de successeurs.

`Ginverse:=GrapheVide()`

**Pour** tout sommet s de G **faire**

**Pour** tout successeur t de s **faire**

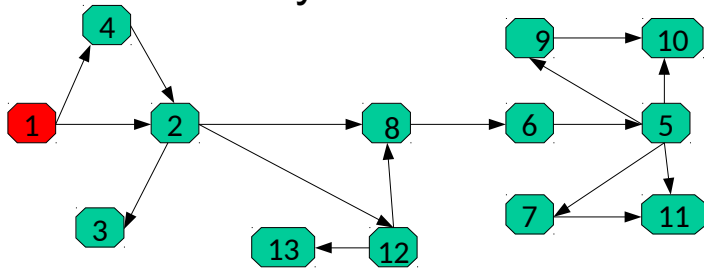
        insérer s dans la liste de succ de t dans Ginverse

**Etape 2** : Parcours en profondeur du graphe inverse de G en  $\Theta(n+m)$ .

**Etape 3** : Parcours en profondeur de G en  $\Theta(n+m)$ .

L'algorithme de Kosaraju-Sharir est donc de complexité  $\Theta(n+m)$ .

# Synthèse



- Parcours **générique** en  $\Theta(n+m)$   
L = (1, 2, 8, 4, 3, 12, 13, 6, 5, 9, 10, 7, 11)  
**Applications** : reconnaissance d'un graphe non-orienté biparti (voir TD), détection des composantes connexes
- Parcours **en largeur** en  $\Theta(n+m)$   
L = (1, 2, 4, 3, 8, 12, 6, 13, 5, 7, 9, 10, 11)  
**Applications** : plus court chemins en nombre d'arcs
- Parcours **en profondeur** en  $\Theta(n+m)$   
L = (1, 2, 3, 8, 6, 5, 7, 11, 9, 10, 12, 13, 4)  
**Applications** : détection de circuit, liste topologique, détection des composantes fortement connexes