LICENCE D'INFORMATIQUE

Sorbonne Université

3I003 – Algorithmique

Cours 5 : Rappels sur les parcours en général, et applications des parcours en profondeur

Année 2023-2024

Responsables et chargés de cours

Fanny Pascual

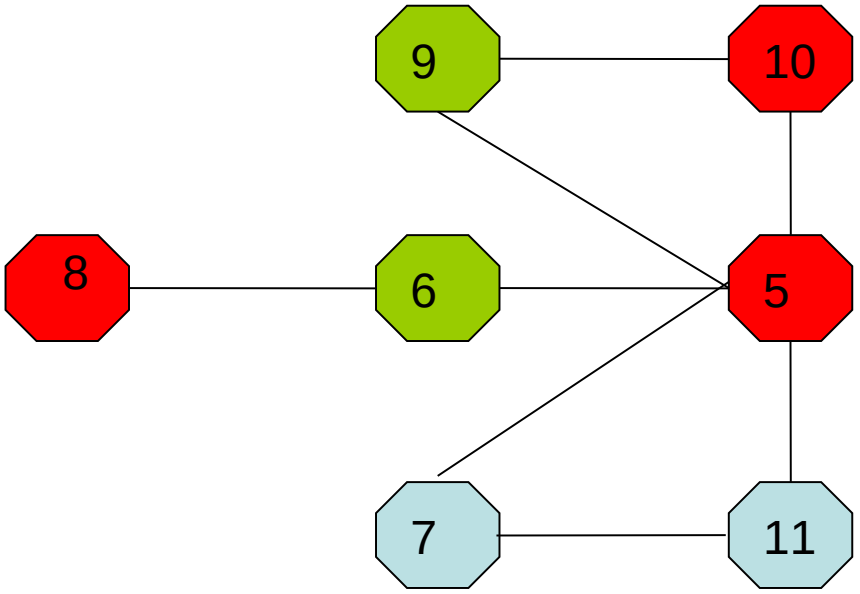
Olivier Spanjaard

Parcours : définitions et notations

Soit G=(S,A) un graphe.

B(T) : Bordure de T⊆S

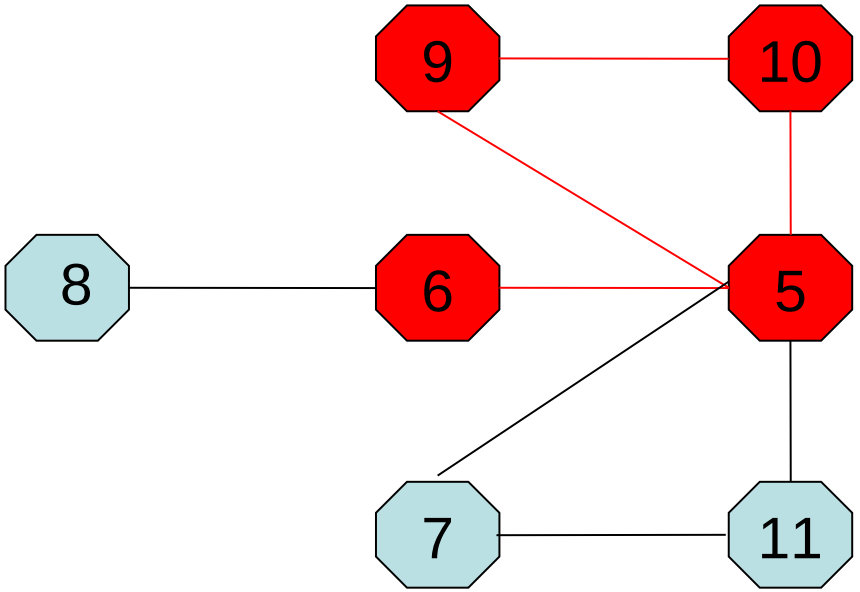
Sous-ensemble des sommets de S-T dont au moins un voisin est dans T.



Bordure de {6,9} = {8,10,5}

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

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



L = (8,6,5,9,10,7,11)

L[2..5] = (6,5,9,10)

Parcours du graphe G=(S,A) :
liste $L=(s_1,s_2,\ldots,s_n)$ des n sommets de G telle que :
pour tout i=2..n, le sommet s_i appartient à la bordure de $\{s_1,s_2,\ldots,s_{i-1}\}$, si cette bordure n'est pas vide.

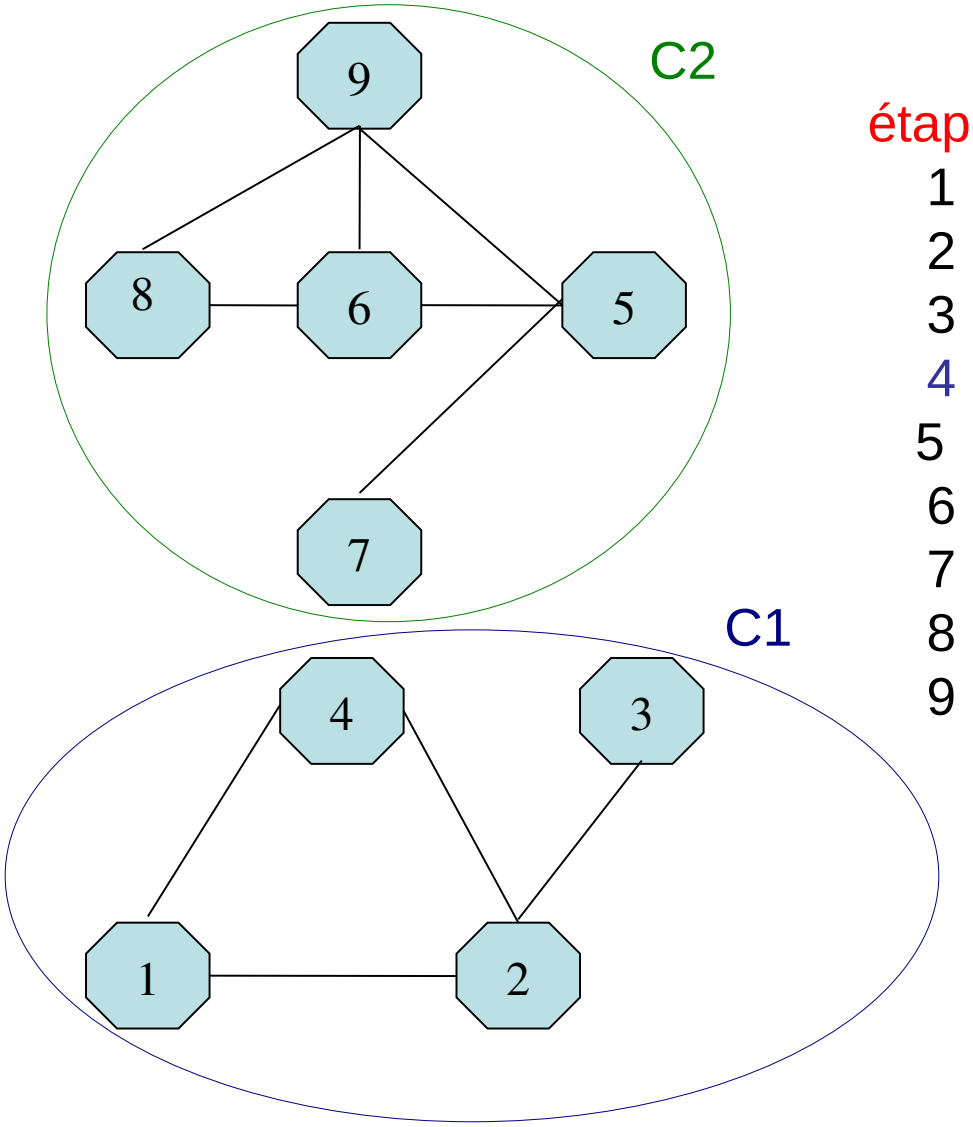
Soit $L=(s_1,s_2,\ldots,s_n)$ un parcours de G.

Si $B(\{s_1,s_2,\ldots,s_{i-1}\})$ 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 $L=(s_1,s_2,\ldots,s_n)$ un parcours de G.

Soit $(i_1,\ldots,i_r)$ les indices des points de régénération de L.

$G[i_1..i_2\text{-}1]$, $G[i_2..i_3\text{-}1]$,…, $G[i_r..n]$ sont les sous-graphes induits par les composantes connexes de G ;

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

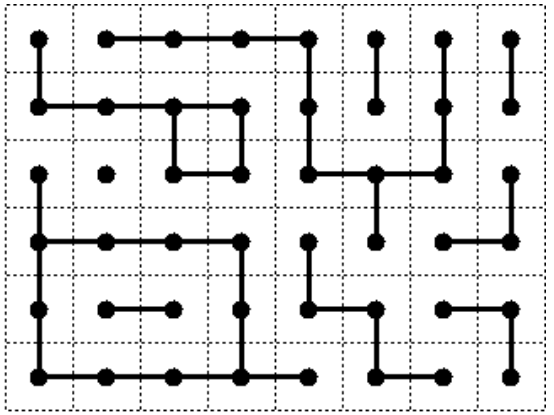
Preuve: simple par récurrence sur le nombre r de points de régénération.

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

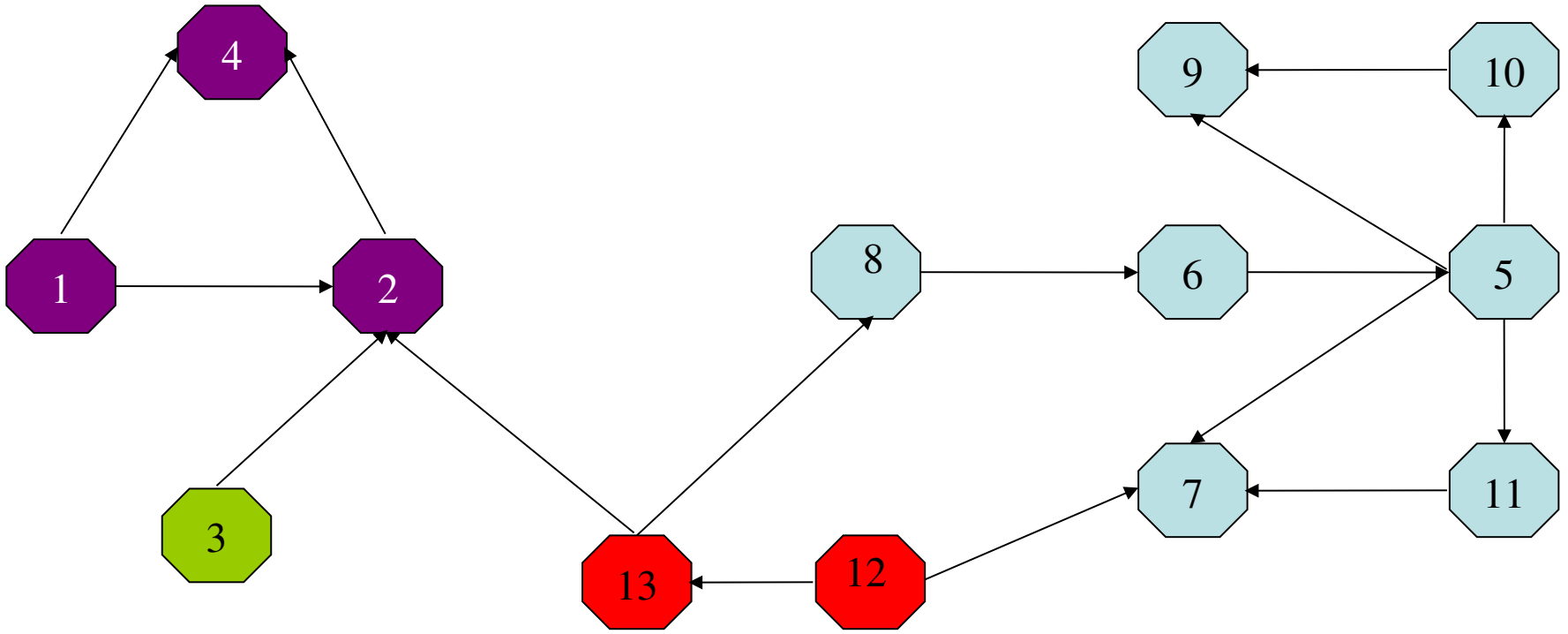
Application en infographie

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}

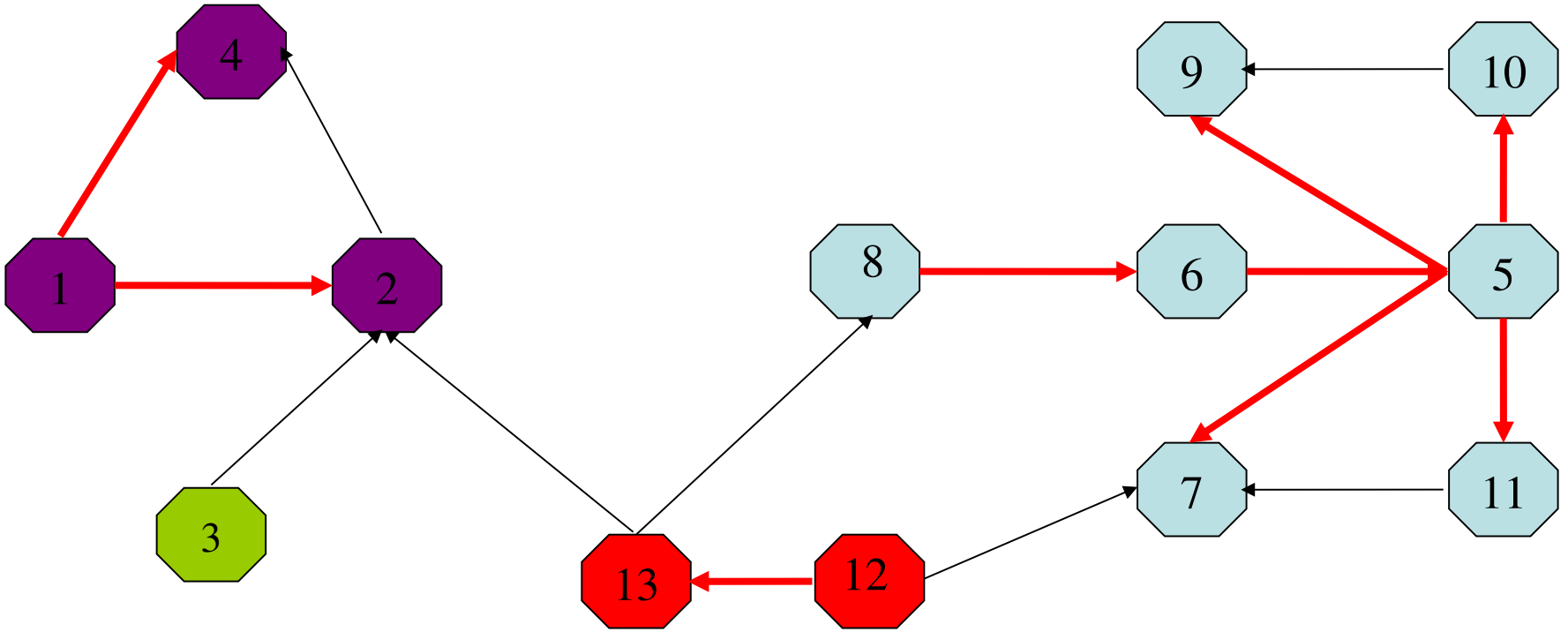
Forêt couvrante associée à un parcours.

Soit $L=(s_1,s_2,\ldots,s_n)$ 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 $A_1(L), A_2(L),\ldots,A_r(L)$
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.

Quiz : Composantes connexes

Au terme d'un parcours d'un graphe non-orienté G, le nombre d'arêtes de la forêt couvrante associée est k. Quel est le nombre de composantes connexes de G ?

A) k
B) k+1
C) n-k-1
D) n-k

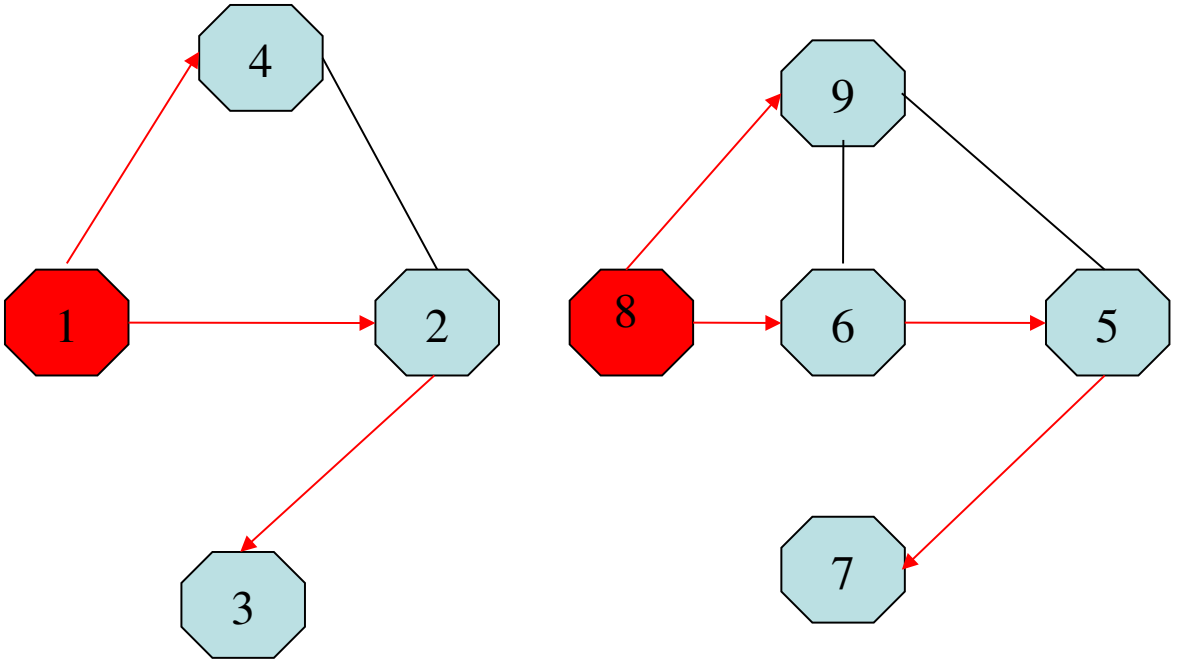
Parcours en largeur
Parcours en profondeur

Soit $L=(s_1,s_2,\ldots,s_n)$ 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



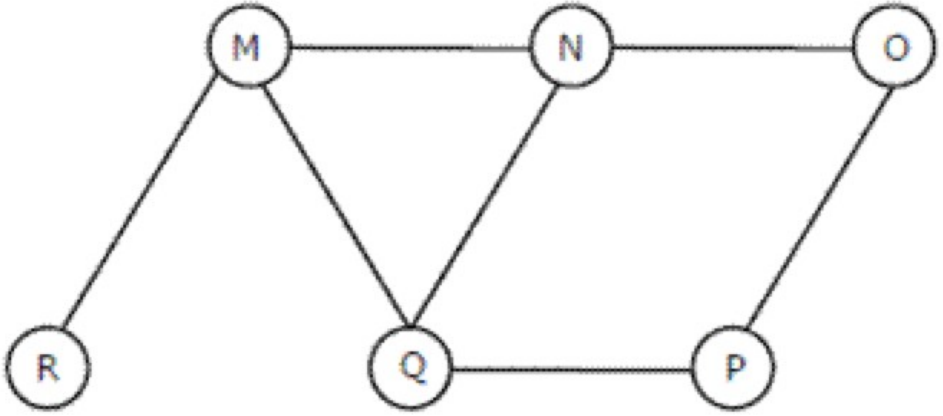
(1)
(1,2)
(1,2,4)
(1,2,4,3)
(1,2,4,3,8)
(1,2,4,3,8,6)
(1,2,4,3,8,6,9)
(1,2,4,3,8,6,9,5)
(1,2,4,3,8,6,9,5,7)

Après chaque étape :
- premier sommet visité ouvert en rouge ;
- autres sommets visités ouverts en vert.

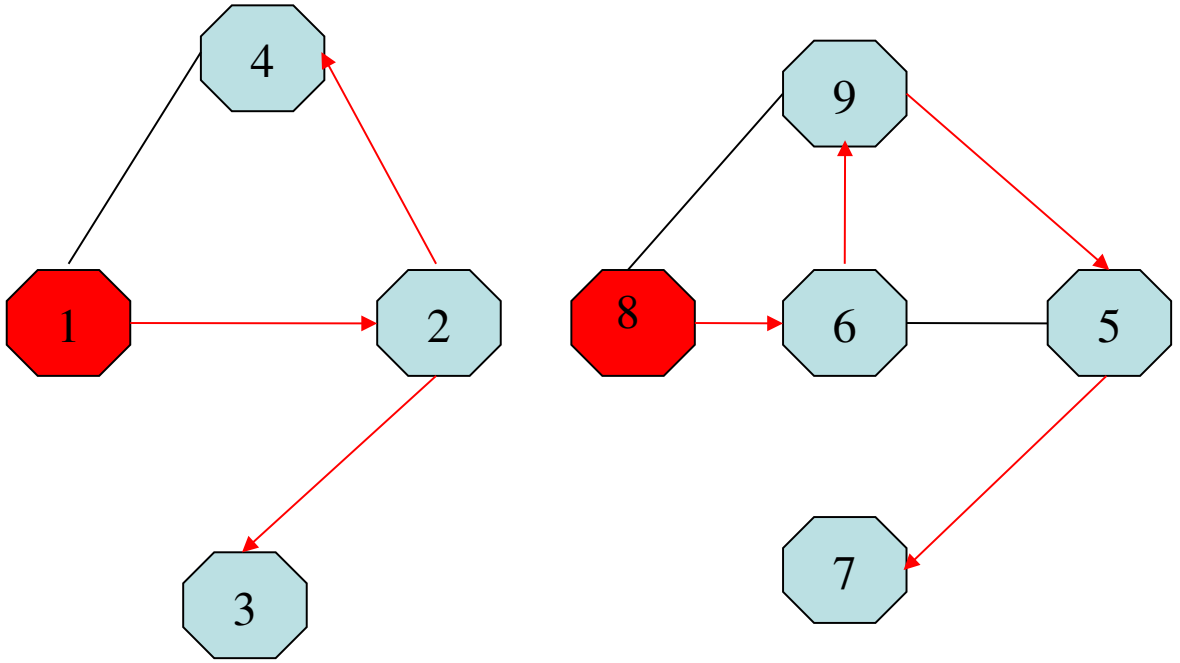
Quiz : Parcours en largeur

Laquelle des listes de sommets est un parcours en largeur possible du graphe ?

A) (M,N,O,P,Q,R)
B) (N,Q,M,P,O,R)
C) (Q,M,N,P,R,O)
D) (Q,M,N,P,O,R)



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)

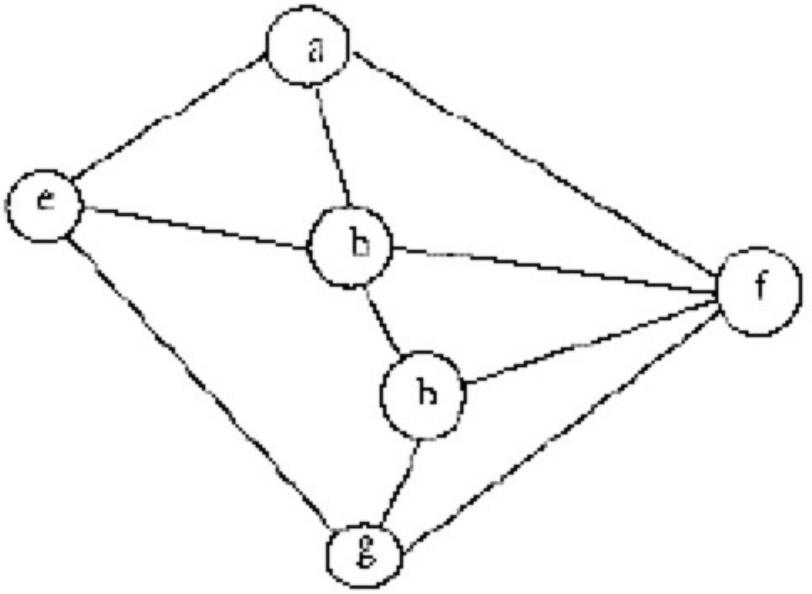
Après chaque étape :
- dernier sommet visité ouvert en rouge ;
- autres sommets visités ouverts en vert.

Quiz : Parcours en profondeur

Parmi les listes suivantes, lesquelles sont des parcours en profondeur du graphe ?

1. (a,b,e,g,h,f) 2. (a,b,f,e,h,g) 3. (a,b,f,h,g,e) 4. (a,f,g,h,b,e)

A) 1, 2 et 3 seulement
B) 1 et 4 seulement
C) 2, 3 et 4 seulement
D) 1, 3 et 4 seulement



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 = $(s_1,...,s_n)$ des sommets du graphe où tout sommet $s_i \in B(\{s_1,...,s_{i-1}\})$ ou $B(\{s_1,...,s_{i-1}\}) = \emptyset$
- Un sommet s_i tel que $B(\{s_1,...,s_{i-1}\}) = \emptyset$ 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 $père_L(s_j)$ le prédécesseur de s_j choisi.
- Le graphe partiel de G formé des arcs $(père_L(s_j),s_j)$ est une forêt couvrante de G notée F(L).

Récapitulatif
(graphe orienté)

- Un sommet s_i est dit ouvert dans $\{s_1,\ldots,s_k\}$ si il comporte un successeur dans $S - \{s_1,\ldots,s_k\}$.

- Parcours en largeur : tout sommet s_i (non point de régénération) est successeur du premier sommet ouvert dans $\{s_1,\ldots,s_{i-1}\}$.

- Parcours en profondeur : tout sommet s_i (non point de régénération) est successeur du dernier sommet ouvert dans $\{s_1,\ldots,s_{i-1}\}$.

- A un parcours en largeur ou en profondeur correspond une unique forêt couvrante.

Quiz : Nombre d'itérations

Quelle est la valeur de la variable *compteur* au terme de l'algorithme ci-dessous, pour un graphe orienté *G=(S,A)* à *n* sommets et *m* arcs représenté sous forme de listes de successeurs ?

compteur ← 0
pour tout sommet x dans S faire
    pour tout successeur y de x faire
        compteur ← compteur+1

A) n
B) m
C) n+m
D) nm

Quiz : Complexité

Quelle est la complexité de l'algorithme ci-dessous, pour un graphe orienté *G=(S,A)* orienté à *n* sommets et *m* arcs représenté sous forme de listes de successeurs ?

```
compteur ← 0
pour tout sommet x dans S faire
        pour tout successeur y de x faire
                compteur ← compteur+1
```

A) O(n)
B) O(m)
C) O(n+m)
D) O(nm)

Algorithme de parcours en profondeur

(graphe orienté)

```
Procédure Prof(G)
début
      pour chaque sommet de 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 par un chemin de sommets non visités.

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(G,s)
- Le moment de la fin de l'appel récursif Explorer(G,s)

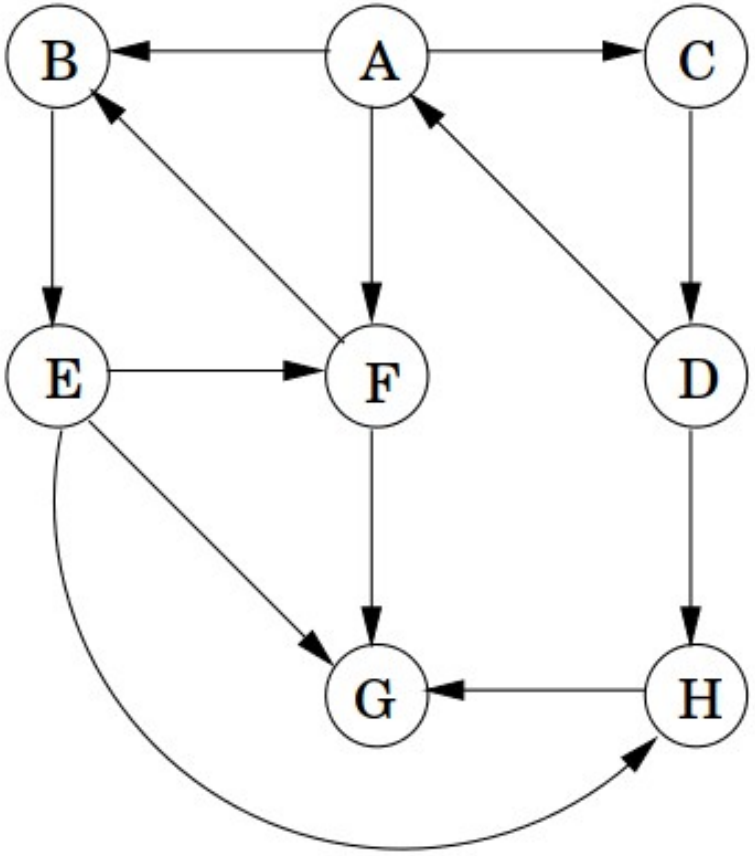
```
Procédure prévisite(s)
pre[s] := num
num := num + 1
```

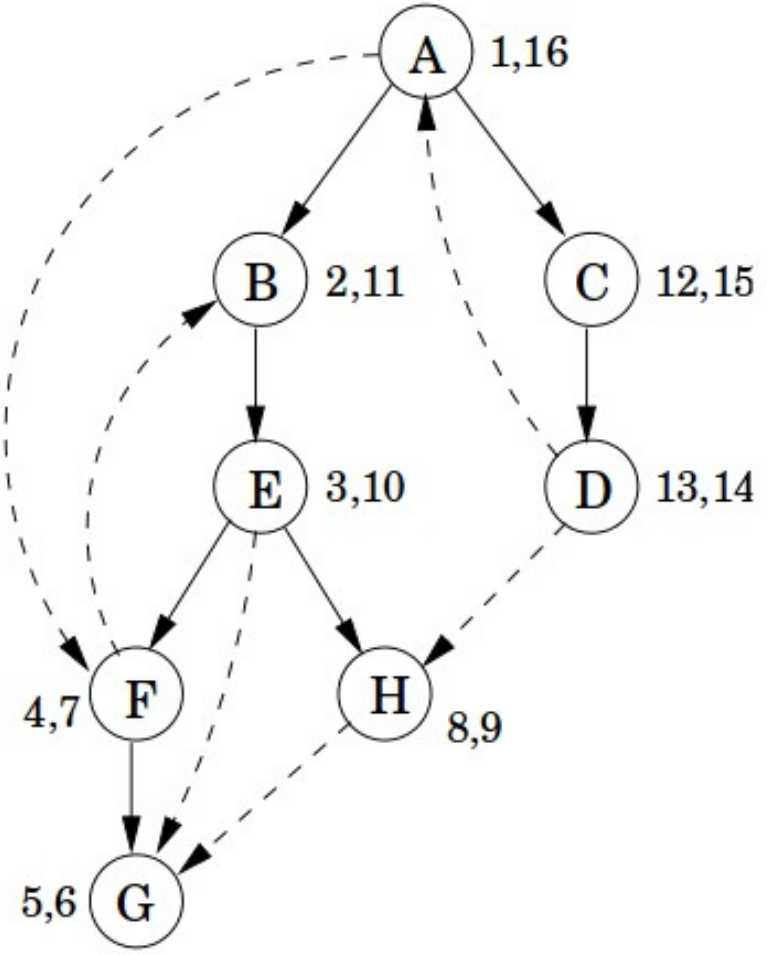
```
Procédure postvisite(s)
post[s] := num
num := num + 1
```

(**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`**(G,*s*) par sommet s, du fait du tableau **`visité`**
- Lors d'un appel récursif **`Explorer`**(G,*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

Quiz : Parcours d’un arbre

Quelle est la complexité d'un algorithme de parcours dans un arbre (choisir la réponse la plus précise possible), représenté sous forme de listes d'adjacence ?

A) $O(n^2)$
B) $O(n)$
C) $O(m)$
D) $O(n+m)$

Quiz : Parcours d’une clique

Quelle est la complexité d'un algorithme de parcours dans un graphe complet (choisir la réponse la plus précise possible), représenté sous forme de listes d'adjacence ?

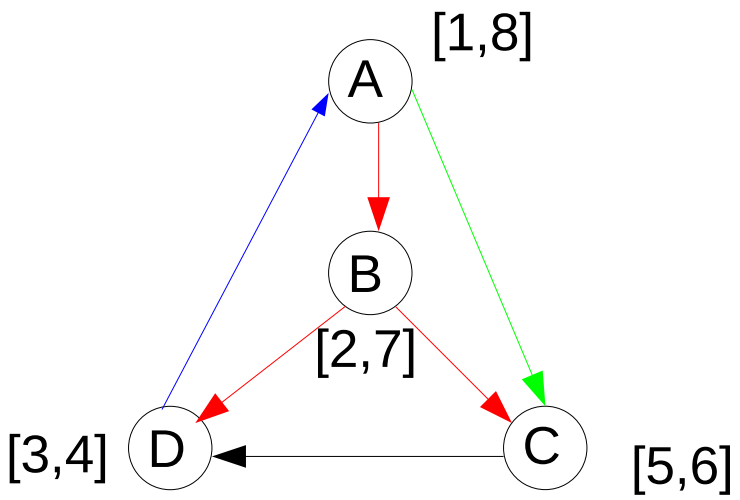
A) $O(n)$
B) $O(m)$
C) $O(n^2)$
D) $O(m^2)$

Arcs associée à un parcours en profondeur

Soit $L=(s_1,s_2,\ldots,s_n)$ 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.



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

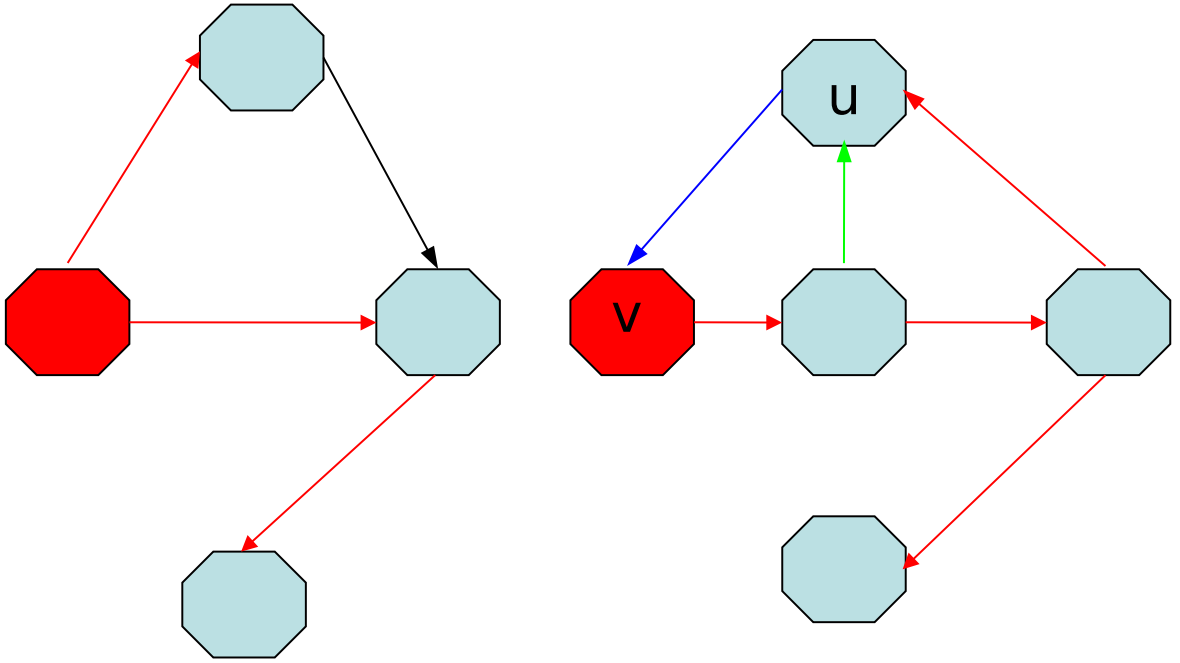
Existence d'un circuit

Problème :
Soit G= (S,A) un graphe orienté.
Existe t-il un circuit dans G ?

Existence de circuit :
Soit L=(s_1,s_2,…,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 sans circuit → pas d'arc arrière

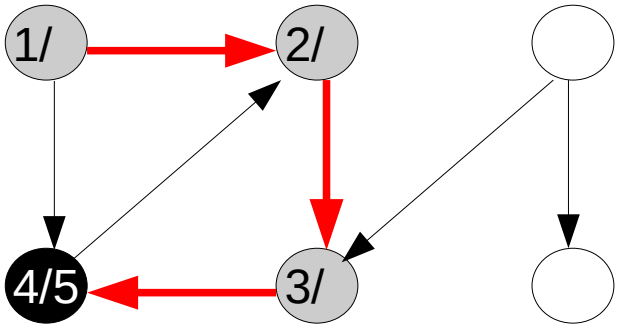
Par contraposée : si il existe un arc arrière (u,v), on construit un circuit en concaténant (u,v) et le chemin de v à u dans F(L)



Théorème du chemin blanc

Lors du parcours d'un graphe G, supposons que l'on colore :
- en **blanc** les sommets non-visités,
- en **gris** les sommets en cours de visite (Explorer(G,s) débuté mais pas terminé),
- en **noir** les sommets visités (appel Explorer (G,s) terminé).

Théorème du chemin blanc
Dans une forêt F de parcours en profondeur d'un graphe G, un sommet t est descendant d'un sommet s ***si et seulement si***, au moment pre[s] où le parcours découvre s, il existe un chemin dans G de s à t composé uniquement de sommets blancs (s exclus).



Preuve

$\Rightarrow$ Si t est un descendant de s dans F, alors tous les arcs (u,v) le long du chemin de s à t dans F vérifient pre[u]<pre[v]. Au moment où le parcours découvre s, les appels sur les sommets le long de ce chemin n'ont donc pas débuté, autrement dit ils sont blancs.

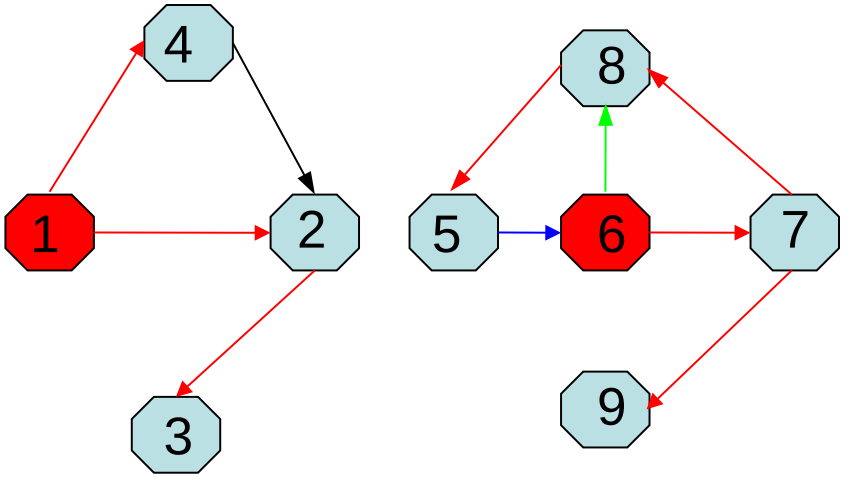
$\Leftarrow$ Par l'absurde, considérons le premier sommet w le long du chemin blanc C de s à t qui n'est pas un descendant de s dans F. Notons v son prédécesseur dans C (possiblement v=s).

On sait que pre[s]<pre[w] car w est blanc lorsque le parcours découvre s. De plus, (v,w) ne peut être ni dans F, ni avant, ni arrière. C'est donc un arc transverse : pre[w]<post[w]<pre[v]<post[v]. On a post[v]$\leq$post[s] car v descendant de s dans F. De pre[s]<pre[w]<post[w]<pre[v]<post[v]$\leq$post[s] on déduit que [pre[w],post[w]] est inclus dans [pres[s],post[s]].

Autrement dit, w est un descendant de s dans F. Contradiction.

Graphe avec circuit → arc arrière

Soit $C=(s_1,...,s_k)$, un circuit dans G. Soit s_i le premier sommet de C visité dans le parcours en profondeur.
D'après le théorème du chemin blanc, les autres sommets de C sont des descendants de s_i dans F(L).
Donc (s_{i-1},s_i) est un arc arrière.

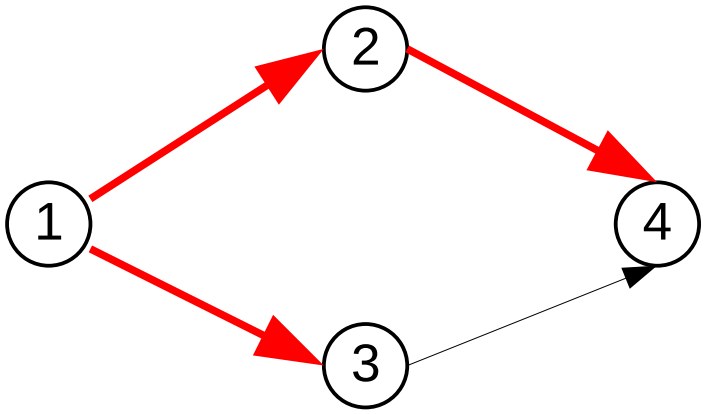


C=(5,6,7,8)
6 visité en 1er
(5,6) arc arrière

Remarque importante

Montrons que le théorème du chemin blanc 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.

Il existe un chemin blanc de 3 à 4 au moment pre[3] où le parcours découvre 3, et pourtant le sommet 4 n'est pas descendant du sommet 3 dans F(L).

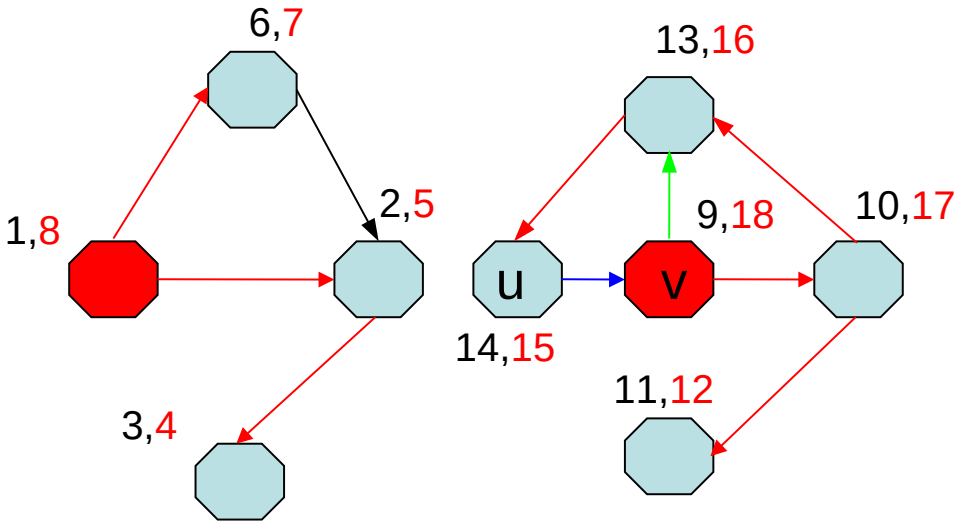
Algorithme de détection de circuit

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

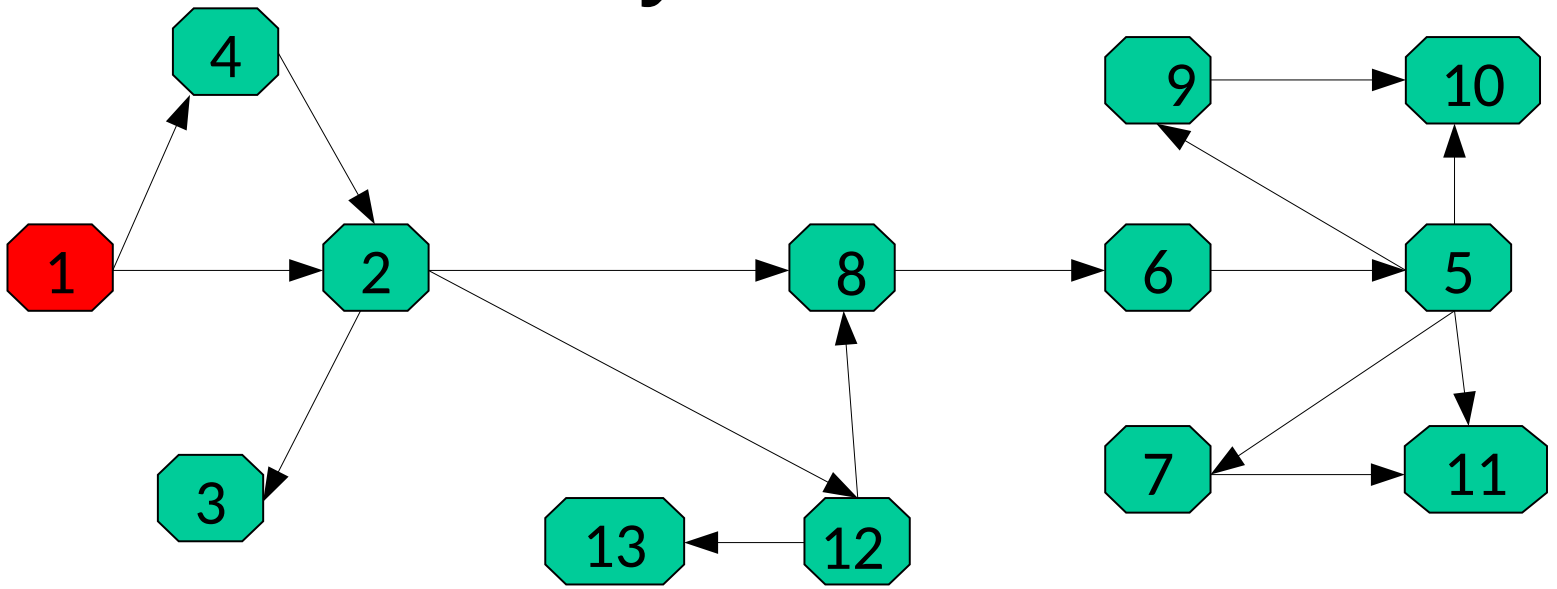
A l'issue du parcours en profondeur, on parcourt les listes de successeurs (en O(n+m)) pour détecter si il existe un arc (u,v) tel que post(u)<post(v).



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

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

Synthèse



- Parcours générique en Θ(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 Θ(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 Θ(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 (voir TD et séance de révision)

Quiz : Existence d’un chemin

Etant donnés deux sommets *s* et *t* dans un graphe orienté *G*, quel algorithme de parcours est-il possible d'utiliser pour déterminer si il existe un chemin de *s* à *t* dans *G* ?

A) Parcours générique
B) Parcours en largeur
C) Parcours en profondeur
D) Les trois