

Chapitre 3 : Exploration d'un graphe

Algorithmique de graphes

Sup Galilée-INFO2

Sylvie Borne

2012-2013

Plan

- 1 Exploration d'un graphe / Parcours
- 2 Parcours en largeur (BFS)
 - Partition des sommets en couches
 - Principe de l'algorithme
 - Implémentation
 - Complexité
 - Application : tester si un graphe est biparti
- 3 Parcours en profondeur (DFS)
 - Prolongement d'une chaîne élémentaire
 - Principe de l'algorithme
 - Implémentation
 - Complexité
- 4 Parcours et connexité
- 5 Parcours et graphes orientés

Exploration de graphes

En utilisant un graphe comme modèle, on a souvent besoin d'un examen exhaustif des sommets. On peut concevoir cet examen comme une promenade le long des arcs/arêtes au cours de laquelle on visite les sommets.

Les algorithmes de parcours de graphes servent de base à bon nombre d'algorithmes. Ils n'ont pas une finalité intrinsèque. Le plus souvent, un parcours de graphe est un outil pour étudier une propriété globale du graphe :

- le graphe est-il connexe ?
- le graphe est-il biparti ?
- le graphe orienté est-il fortement connexe ?
- quels sont les sommets d'articulation ?

Exploration de graphes

Problème : On appelle exploration / parcours d'un graphe, tout procédé déterministe qui permet de choisir, à partir des sommets visités, le sommet suivant à visiter.

Le problème consiste à déterminer un ordre sur les visites des sommets.

Remarque :

L'ordre dans l'examen des sommets induit :

- une numérotation des sommets visités
- le choix d'une arête pour atteindre un nouveau sommet à partir des sommets déjà visités.



Exploration de graphes

Attention : La notion d'exploration / parcours peut être utilisée dans les graphes orientés comme non-orientés. Dans la suite, nous supposerons que le graphe est non-orienté. L'adaptation au cas des graphes orientés s'effectue sans aucune difficulté.



Exploration de graphes

Définition : *racine*

Le sommet de départ, fixé à l'avance, dont on souhaite visiter tous les descendants est appelé racine de l'exploration.

Définition : *parcours*

Un parcours de racine r est une suite L de sommets telle que

- 1 r est le premier sommet de L ,
- 2 chaque sommet apparaît une fois et une seule dans L ,
- 3 tout sommet sauf la racine est adjacent à un sommet placé avant lui dans la liste.

Deux types d'exploration :

- parcours en largeur
- parcours en profondeur



Définition : *distance*

Soient $x, y \in V$, on pose
 $d(x, y)$ = longueur d'un plus court chemin (si $G = (V, A)$ un graphe orienté) (ou plus courte chaîne, si $G = (V, E)$ un graphe non-orienté) entre x et y (en nombre d'arêtes).

$d(x, y)$ est appelé distance entre x et y .

Remarque :

S'il n'existe pas de chaîne (chemin) entre x et y , $d(x, y) = +\text{inf}$.

Définition : *partition en couches*

On appelle partition en couches associée à un sommet source r (appelé racine), la suite d'ensembles définie itérativement comme suit :

$$C_0 = \{r\}$$

$$C_1 = \{x \in V \mid d(r, x) = 1\}$$

$$C_2 = \{x \in V \mid d(r, x) = 2\}$$

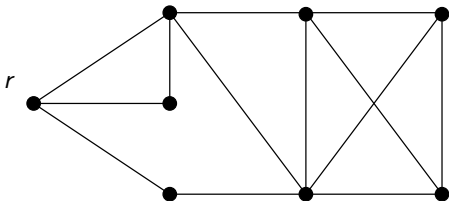
...

$$C_i = \{x \in V \mid d(r, x) = i\}$$



Partition des sommets en couches

Exemple :



Remarque :

Il n'existe pas d'arêtes entre deux couches C_i et C_j si $|i - j| \geq 2$.

Remarque :

Cas orienté

On définit les couches de la même manière, en considérant les descendants.

C_1 = ensemble des sommets descendants (successeurs) de r .

→ Ici, il peut y avoir des arcs entre C_i et C_j avec $i \geq j + 2$.

Ceux-ci vont nécessairement de C_i vers C_j .



Principe de l'algorithme

On visite les sommets en construisant les couches dans l'ordre.

Idée : $C_{i+1} \subseteq V(C_i)$ où $V(C_i)$ désigne l'ensemble des voisins des sommets appartenant à C_i .

C_{i+1} s'obtient à partir de C_i en déployant, pour chaque sommet de C_i , et ce dans l'ordre, la liste des voisins non encore marqués.

Algorithme :

On part de r et on visite successivement les sommets.

- on visite d'abord les voisins (successeurs) de r .
Ces sommets constituent C_1 .
- C_{i+1} est obtenu en visitant tous les sommets successeurs des sommets de C_i , non encore visités.



Définition : *sommet marqué*

Un sommet est dit marqué s'il a été placé dans une couche C_i .

Définition : *sommet exploré*

Un sommet marqué est dit exploré lorsque l'on aura marqué tous ses voisins.

Propriété :

Lors d'un parcours en largeur, on applique la règle "premier marqué-premier exploré".

i.e. **Pour construire les couches, on explore les sommets en respectant l'ordre dans lequel ils ont été marqués.**



Implémentation

La gestion des sommets est réalisable au moyen d'une structure de données appelée file.

Définition : *file*

Une file (queue en anglais) est une structure de données basée sur le principe du "Premier entré, premier sorti" (FIFO en anglais), ce qui veut dire que les premiers éléments ajoutés à la file seront les premiers à être récupérés.

Implémentation :

On utilise une file de taille $n + 1$ où $n = |V|$.

On associe à chaque sommet un numéro.

La racine aura le numéro 1.

Les sommets marqués à partir d'une couche C_i auront le numéro $i + 1$.

On utilise deux curseurs : tête et queue.



Implémentation

→ Procédure LARGEUR (G : graphe, r : sommet)

pour $i = 1$ à n **faire**

 Marque[i] $\leftarrow 0$

fin pour

tête $\leftarrow 1$

queue $\leftarrow 1$

File[tête] $\leftarrow r$

Marque[r] $\leftarrow 1$



Implémentation

tantque tête \leq queue **faire**

$x \leftarrow \text{File}[\text{tête}]$

pour y parcourant la liste des successeurs de x **faire**

si Marque[y] = 0 **alors**

Marque[y] = Marque[x] + 1

queue \leftarrow queue + 1

File[queue] \leftarrow y

finsi

fin pour

tête \leftarrow tête + 1

fin tantque



Implémentation

Remarque :

L'ordre de visite des sommets dépend de l'ordre des listes d'adjacence car on doit parcourir ces listes.

Remarque :

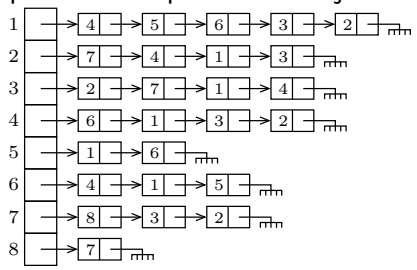
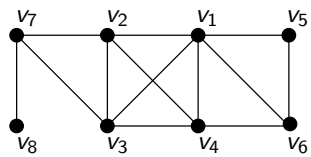
L'algorithme détermine, pour chaque sommet x visité, une chaîne élémentaire unique de r à x .



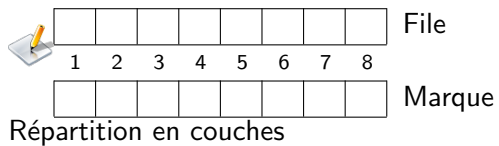
Implémentation

Exemple :


Soit un graphe G et une représentation par listes d'adjacence.




Parcours en largeur à partir de la racine 7.



Complexité

- Espace : 
représentation du graphe : $\mathcal{O}(n + m)$
structure file : $\mathcal{O}(n)$
tableau marque : $\mathcal{O}(n)$
 \Rightarrow complexité dans l'espace : $\mathcal{O}(n + m)$

- Temps : 
marquage : n opérations
exploration :
chaque sommet x nécessite d_x opérations
donc en tout $\sum_{x \in V} d_x = 2|E| = 2m$
 \Rightarrow complexité en temps : $\mathcal{O}(n + m)$



Tester si un graphe est biparti

Définition : *stable*

On appelle stable, un ensemble de sommets induisant un sous-graphe sans arête.

Définition : *biparti*

On appelle graphe biparti, un graphe $G = (V, E)$ pour lequel V peut être partitionné en deux stables A et B .

Propriété :

Un graphe G est biparti si et seulement si G est 2_chromatique.

Tester si un graphe est biparti

Algorithme :

- Faire un parcours en largeur de G .
- Colorer les niveaux pairs en rouge, les niveaux impairs en vert.
- Si aucun arc/arête entre les sommets d'un même niveau

Alors

le graphe est biparti

Sinon

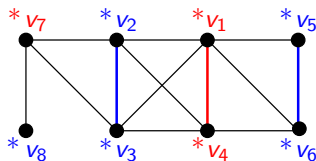
le graphe n'est pas biparti.

Exemple :



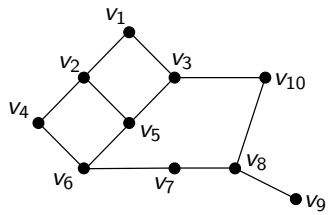
Le graphe de l'exemple précédent n'est pas biparti.
Problème avec les arêtes

V_2V_3 , V_1V_4 , V_5V_6 .



Tester si un graphe est biparti

Exemple :



Parcours en largeur à partir de la racine v_1 et en suivant l'ordre lexicographique.

1	2	3	4	5	6	7	8	9	10

File

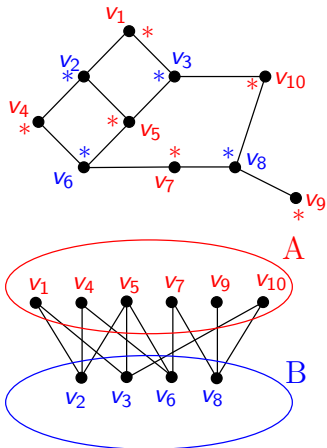
--	--	--	--	--	--	--	--	--	--

Marque



Tester si un graphe est biparti

Exemple :



Parcours en largeur à partir de la racine v_1 et en suivant l'ordre lexicographique.

File :

1	2	3	4	5	10	6	8	7	9
---	---	---	---	---	----	---	---	---	---

Marque :

1	2	3	4	5	6	7	8	9	10
1	2	2	3	3	4	5	4	5	3
*	*	*	*	*	*	*	*	*	*

Parcours en profondeur (DFS)

Définition : *prolongement d'une chaîne élémentaire*

Soit Γ une chaîne élémentaire entre deux sommets x_0 et x_p . On appelle prolongement de $\Gamma = (e_1, e_2, \dots, e_p)$, une chaîne élémentaire Γ' de la forme

$$\Gamma' = (e_1, e_2, \dots, e_p, e_{p+1})$$

où e_{p+1} est une nouvelle arête ajoutée à la chaîne.

Si e_{p+1} est entre x_p et x_{p+1} , on dit que la chaîne est prolongée à x_{p+1} .

Remarque :

Il est clair que Γ n'admet pas de prolongement si et seulement si tous les voisins de x_p se trouvent dans Γ .



Principe de l'algorithme DFS

On part du sommet r .

Lorsque l'on arrive à un sommet x , on ne l'explore pas, on visite un de ses voisins non encore visité, on cherche donc à prolonger le chemin de r à x .

Propriété :

Lors d'un parcours en profondeur, on applique la règle "Dernier marqué, premier exploré".

i.e. **On explore les sommets dans l'ordre inverse de celui utilisé pour les marquer.**

Remarque :

On pourra utiliser "fermé" pour "exploré".

Principe de l'algorithme DFS

La gestion des sommets est réalisable au moyen d'une structure de données appelée pile.

Définition : *pile*

Une pile (stack en anglais) est une structure de données basée sur le principe du dernier arrivé, premier sorti (LIFO en anglais), ce qui veut dire que les derniers éléments ajoutés à la pile seront les premiers à être récupérés.



Implémentation

Implémentation :

→ Procédure PROFONDEUR (G : graphe, r : sommet)



```
pour  $i = 1$  à  $n$  faire  
    Marque[ $i$ ] ← faux  
fin pour  
 $h \leftarrow 1$  (hauteur de pile)  
Pile[ $h$ ] ←  $r$   
tantque  $h > 0$  faire  
     $x \leftarrow$  Pile[ $h$ ]  
    si PS[ $x$ ]  $\neq 0$  alors  
         $y \leftarrow$  LS[PS[ $x$ ]]  
        si Marque[ $y$ ] = faux alors
```

```
        Marque[ $y$ ] = vrai  
         $h \leftarrow h + 1$   
        Pile[ $h$ ] ←  $y$   
    finsi  
    Mettre à jour PS[ $x$ ]  
sinon  
     $h \leftarrow h - 1$   
finsi  
fin tantque
```



Exemple

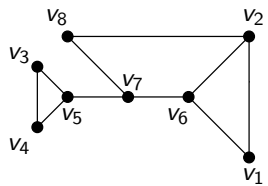
Exemple :



Soit un graphe G et une représentation par table des successeurs.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LS	2	6	6	1	8	4	5	3	5	3	4	7	2	7	1	6	5	8	2	7

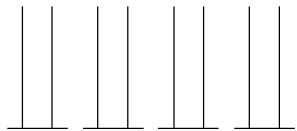
1 2 3 4 5 6 7 8



PS :

1	3	6	8	10	13	16	19
---	---	---	---	----	----	----	----

Parcours en profondeur à partir de la racine 2.



Pile

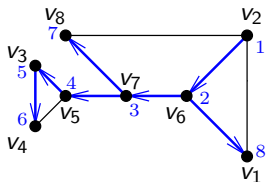
Exemple

Exemple :



Soit un graphe G et une représentation par table des successeurs.

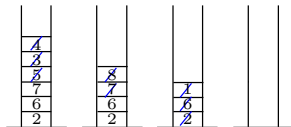
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
LS	2	6	6	1	8	4	5	3	5	3	4	7	2	7	1	6	5	8	2	7
										1	2	3	4	5	6	7	8			



PS :

1	3	6	8	10	13	16	19
---	---	---	---	----	----	----	----

Parcours en profondeur à partir de la racine 2.



Pile

Remarque :


L'ordre de parcours des sommets dans un parcours en profondeur dépend de la représentation du graphe par les tableaux *PS* et *LS*.


Remarque :

L'algorithme précédent peut être écrit sous forme récursive.



Complexité

- Espace : 
représentation du graphe : $\mathcal{O}(n + m)$
structure pile : $\mathcal{O}(n)$
tableau marque : $\mathcal{O}(n)$
 \Rightarrow complexité dans l'espace : $\mathcal{O}(n + m)$

- Temps : 
marquage : n opérations
exploration :
chaque sommet x nécessite d_x opérations
donc en tout $\sum_{x \in V} d_x = 2|E| = 2m$
 \Rightarrow complexité en temps : $\mathcal{O}(n + m)$



Parcours et connexité

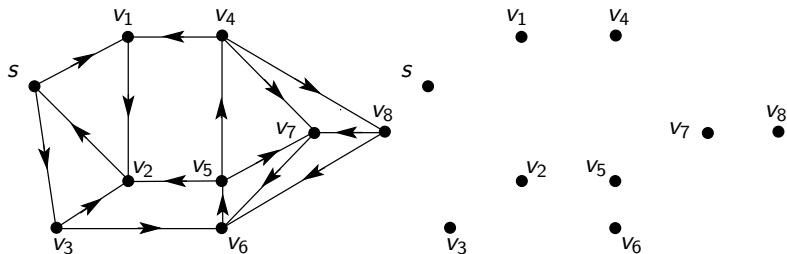
Un parcours du graphe détermine si le graphe est connexe.
Les algorithmes supposent que le graphe soit connexe.
Si ce n'est pas le cas, il faut considérer
une racine pour chaque composante connexe.



Exemple :



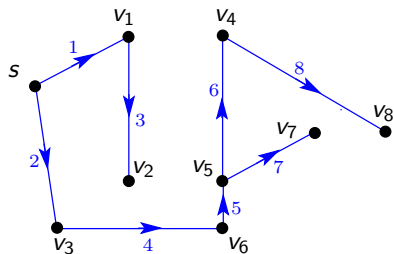
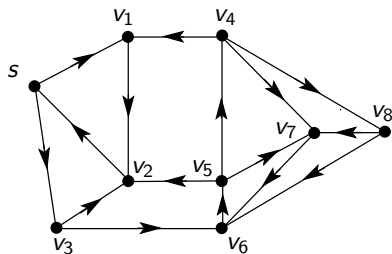
Parcours en **largeur** à partir de s



Exemple :



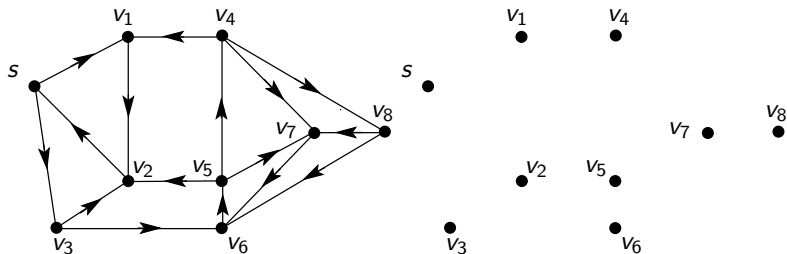
Parcours en **largeur** à partir de s



Exemple :



Parcours en **profondeur** à partir de s



Parcours et graphes orientés

Exemple :



Parcours en **profondeur** à partir de s

