Introduction aux graphes et aux matrices

Cours n°4

EPITA Cyber 2
2025-2026

Ce cours présente le parcours en profondeur dans les graphes et les problèmes de connexité qu'il permet de résoudre.

1 Introduction

Certains problèmes de la théorie des graphes se réduisent à des questions sur la connexité du graphe : Peut-on atteindre un sommet i donné à partir d'un sommet j ? Quel est l'ensemble de sommets accessibles depuis un sommet donné ?

Le parcours en profondeur et l'ensemble des algorithmes qui en découlent permettent de répondre efficacement aux questions de ce type.

2 Parcours en Profondeur

2.1 Parcours en profondeur d'un graphe non-orienté

Le principe du parcours en profondeur (ou DFS pour Depth-First Search en anglais) à partir d'un sommet quelconque i est d'explorer chaque chaine qui part de i sans jamais passer deux fois par le même nœud. Pour cela, il faut suivre une première chaine élémentaire jusqu'à arriver à un sommet sans voisin 'non exploré', on remonte alors au sommet précédant de la chaine (backtracking) et on explore un voisin 'non exploré', si il existe. On procède ainsi jusqu'à ce que toutes les chaines partant de i soient parcourues.

Autrement dit, lorsque l'on est en train d'explorer un nœud, deux possibilité s'offrent à nous pour continuer l'exploration du graphe :

— soit le nœud possède des voisins 'non exploré' et alors on va vers ces voisins ;

— soit tous les voisins du nœud sont 'explorés' et alors on revient au sommet précédent.

L'idée est d'étiqueter les nœuds pour savoir à chaque instant s'il a été exploré, et s'il reste des chaines a explorer à partir de ce nœud :

— Lorsque que l'on visite un nœud, on le met dans l'état **'exploré'**. Un nœud qui n'a jamais été visité est donc dans l'état **'non exploré'**.

— Lorsque tous les voisins d'un nœud exploré sont dans l'état 'exploré', on dit que ce nœud est **fermé**. Au contraire, si certains voisins de ce nœud n'ont pas encore été exploré, le nœud est dit **ouvert**.

Dans le parcours en profondeur, lorsqu'un sommet devient fermé, on remonte au sommet précédant dans la chaîne et on **explore un sommet voisin non exploré**, si il existe. Si

plusieurs sommets voisins sont non explorés, on choisira d'**explorer les sommets dans l'ordre numérique ou lexicographique**.

L'exploration qui a commencé en i est terminée lorsque le **sommet i est fermé**. On a alors exploré tous les **sommets accessibles par une chaîne** depuis le sommet i. Les sommets explorés et les arêtes empruntées lors de l'exploration à partir de i forment un **arbre**.

Si un parcours en profondeur à partir d'un nœud i s'arrête sans que l'on ait parcouru tous les nœuds du graphe, alors cela signifie que l'on a exploré toute la composante connexe contenant i mais qu'il existe d'autres composantes connexes. Ainsi, s'il reste des sommets non explorés, on continue le parcours en profondeur en commençant depuis un sommet non exploré. On trouvera alors une autre composante connexe. Et ainsi de suite jusqu'à parcourir toutes les **composantes connexe du graphe** (chaque arbre d'exploration correspondant à une composante connexe).

Exemple :

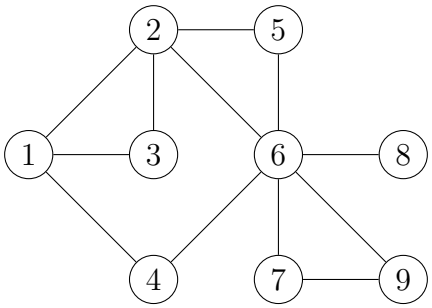

FIGURE 1 – Un graphe non orienté G_1 d'ordre 9.

Parcours en profondeur de G_1 :

nœud	non explorés	explorés	ouverts	fermés	arêtes
/	$\{1,2,3,4,5,6,7,8,9\}$	$\emptyset$	$\emptyset$	$\emptyset$	/
1	$\{2,3,4,5,6,7,8,9\}$	$\{1\}$	$\{1\}$	$\emptyset$	/
2	$\{3,4,5,6,7,8,9\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$	$(1,2)$
3	$\{4,5,6,7,8,9\}$	$\{1,2,3\}$	$\{1,2,3\}$	$\emptyset$	$(2,3)$
5	$\{4,6,7,8,9\}$	$\{1,2,3,5\}$	$\{1,2,5\}$	$\{3\}$	$(2,5)$
6	$\{4,7,8,9\}$	$\{1,2,3,5,6\}$	$\{1,2,5,6\}$	$\{3\}$	$(5,6)$
4	$\{7,8,9\}$	$\{1,2,3,4,5,6\}$	$\{1,2,5,6,4\}$	$\{3\}$	$(6,4)$
7	$\{8,9\}$	$\{1,2,3,4,5,6,7\}$	$\{1,2,5,6,7\}$	$\{3,4\}$	$(6,7)$
9	$\{8\}$	$\{1,2,3,4,5,6,7,9\}$	$\{1,2,5,6,7,9\}$	$\{3,4\}$	$(7,9)$
8	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	$\{1,2,5,6,8\}$	$\{3,4,9,7\}$	$(6,8)$
/	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	$\emptyset$	$\{3,4,9,7,8,6,5,2,1\}$	/

L'ordre de parcours des sommets pour G_1 est : 1-2-3-5-6-4-7-9-8.
La séquence d'arêtes empruntées est : (1,2), (2,3), (2,5), (5,6), (6,4), (6,7), (7,9), (6,8).

L'ensemble des arêtes empruntés forment l'**arbre couvrant** suivant :

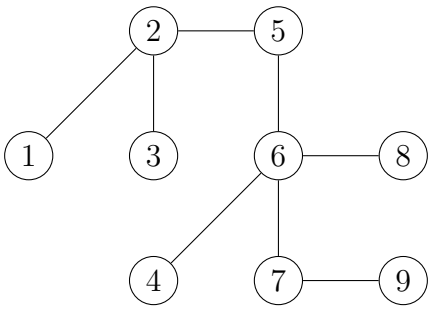

FIGURE 2 – L'arbre couvrant obtenu après le parcours en profondeur de G_1.

Implémentation du parcours en profondeur. Pour implémenter le parcours en profondeur, on va utiliser une **pile** pour se souvenir des sommets à visiter. Il y a alors deux stratégies possibles : soit la pile est implicite via une récursion, soit explicite avec une structure de pile. Voici un pseudo-code pour chacune des stratégies :

Parcours en profondeur (version récursive) :

```
\* pour effectuer un parcours en profondeur
   à partir du nœud i, appeler la fonction
   récursive Explorer sur l'entrée i */

Explorer(graphe G, nœud j):
     marquer j comme 'exploré'
     pour tous les voisins k de j:
          si k est 'non exploré':
               Explorer(G, k)
```

Parcours en profondeur (version itérative) :

```
Explorer(graphe G, nœud i):
     pile = [i]
     tant que la pile n'est pas vide:
          j = pile.dépiler()
          si j est 'non exploré':
               marquer j comme 'exploré'
               pour tous les voisins k de j:
                    pile.empiler(k)
```

Pour *marquer* un nœud comme étant 'exploré', on pourra utiliser un tableau `Etat` à une dimension de longueur n (où n est l'ordre du graphe). Chaque cellule du tableau contiendra un booléen (0 ou 1). On aura alors `Etat[i]` $= 0$ lorsque le nœud i est 'non exploré' et `Etat[i]` $= 1$ lorsqu'il a déjà été 'exploré'. Le tableau `Etat` doit donc être initialisé tout à 0 au début de l'algorithme.

Questions.

(1) Dans les deux pseudo-codes précédents, quand est-ce que le nœud j est ouvert ? Quand est-ce qu'il est fermé ?

(2) Dans l'algorithme itératif, dans quel ordre faut-il empiler les voisins de j pour explorer les nœuds dans le même ordre que dans l'algorithme récursif ?

(3) Que se passe-t-il lorsque le graphe que l'on parcours est un arbre ?

(4) Pour implémenter un parcours en profondeur, quelle représentation en machine du graphe est la plus appropriée ? Quelle est alors la complexité du parcours en profondeur ?

2.2 Parcours en profondeur d'un graphe orienté

Pour les graphes orientés, le principe du parcours en profondeur est très similaire aux graphes non orientés : au lieu d'explorer les voisins d'un nœud, on explorera ses successeurs.

Ainsi, pour le parcours en profondeur d'un graphe orienté à partir d'un nœud i, il faut suivre un premier chemin élémentaire jusqu'à arriver à un sommet sans successeur 'non exploré', on remonte alors au sommet précédant du chemin et on explore un sommet successeur 'non exploré', si il existe.

L'exploration qui a commencé en i est terminée lorsque le **sommet i est fermé**. On a alors exploré tous les **sommets accessibles par un chemin** depuis le sommet i. Les sommets explorés et les arcs empruntés forment une **arborescence de racine** i.

Si il reste des sommets non explorés, on continue le parcours en commençant une exploration depuis un sommet non exploré.

Exemple :

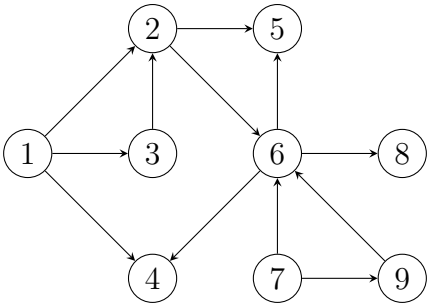

FIGURE 3 – Un graphe orienté G_2 d'ordre 9.

nœud	non explorés	explorés	ouverts	fermés	arcs
/	$\{1,2,3,4,5,6,7,8,9\}$	$\emptyset$	$\emptyset$	$\emptyset$	/
1	$\{2,3,4,5,6,7,8,9\}$	$\{1\}$	$\{1\}$	$\emptyset$	/
2	$\{3,4,5,6,7,8,9\}$	$\{1,2\}$	$\{1,2\}$	$\emptyset$	$(1,2)$
5	$\{3,4,6,7,8,9\}$	$\{1,2,5\}$	$\{1,2,5\}$	$\emptyset$	$(2,5)$
6	$\{3,4,7,8,9\}$	$\{1,2,5,6\}$	$\{1,2,6\}$	$\{5\}$	$(2,6)$
4	$\{3,7,8,9\}$	$\{1,2,4,5,6\}$	$\{1,2,6,4\}$	$\{5\}$	$(6,4)$
8	$\{3,7,9\}$	$\{1,2,4,5,6,8\}$	$\{1,2,6,8\}$	$\{5,4\}$	$(6,8)$
3	$\{7,9\}$	$\{1,2,3,4,5,6,8\}$	$\{1,3\}$	$\{5,4,8,6,2\}$	$(1,3)$
/	$\{7,9\}$	$\{1,2,3,4,5,6,8\}$	$\emptyset$	$\{5,4,8,6,2,3,1\}$	/
7	$\{9\}$	$\{1,2,3,4,5,6,7,8\}$	$\{7\}$	$\{5,4,8,6,2,3,1\}$	/
9	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	$\{7,9\}$	$\{5,4,8,6,2,3,1\}$	$(7,9)$
/	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	$\emptyset$	$\{5,4,8,6,2,3,1,9,7\}$	/

L'ordre de parcours des sommets pour G_2 est : 1-2-5-6-4-8-3-7-9.
La séquence d'arcs empruntés est : (1,2), (2,5), (2,6), (6,4), (6,8), (1,3), (7,9).

On obtient deux arborescences lors du parcours de G_2 :

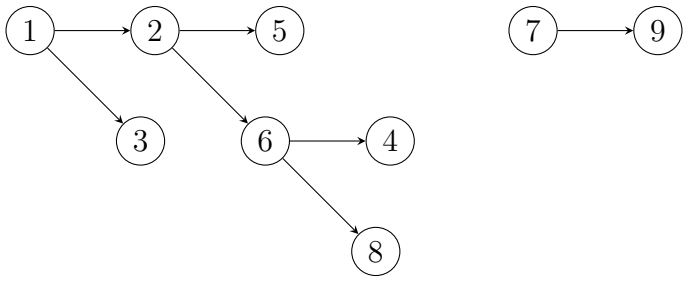

FIGURE 4 – Arborescences obtenues par parcours en profondeur de G_2.

Questions. Que se passe-t-il lorsque l'on parcours une arborescence de racine i ?

Vocabulaire supplémentaire. On distingue quatre types d'arcs découlant d'un parcours en profondeur :

- **arc d'arbre** : un arc emprunté lors du parcours en profondeur du graphe,
- **arc avant** : un arc (i,j) est un arc avant si (i,j) n'est pas un arc d'arbre et si j est un descendant de i dans une des arborescences obtenues par le parcours,
- **arc arrière** : un arc (i,j) est un arc arrière si (i,j) n'est pas un arc d'arbre et si j est un ancêtre de i dans une des arborescences obtenues par le parcours,
- **arc croisé** : un arc (i,j) est un arc croisé s'il n'y pas de relation entre i et j dans les arborescences (ils peuvent faire partie de la même arborescence).

Dans le parcours en profondeur du graphe G_2, on obtient :

- arcs d'arbre : (1,2), (1,3), (2,5), (2,6), (6,4), (6,8), (7,9),
- arcs avants : (1,4),
- arcs arrières : aucun,
- arcs croisés : (7,6), (9,6), (3,2), (6,5).

Implémentation du parcours en profondeur pour les graphes orientés. Pour implémenter le parcours en profondeur dans les graphes orientés, on peut reprendre les implémentations (récursives et itératives) vues précédemment pour les graphes non orientés en remplaçant "voisins de j" par "successeurs de j" :

Parcours en profondeur d'un graphe orienté (version récursive) :

```
\* pour effectuer un parcours en profondeur
   à partir du nœud i, appeler la fonction
   récursive Explorer sur l'entrée i */

Explorer(graphe G, nœud j):
     marquer j comme 'exploré'
     pour tous les successeurs k de j:
          si k est 'non exploré':
               Explorer(G, k)
```

Parcours en profondeur d'un graphe orienté (version itérative) :

```
Explorer(graphe G, nœud i):
     pile = [i]
     tant que la pile n'est pas vide:
          j = pile.dépiler()
          si j est 'non exploré':
               marquer j comme 'exploré'
               pour tous les successeurs k de j:
                    pile.empiler(k)
```

Comme pour les graphes non orientés, pour *marquer* un nœud comme étant 'exploré', on pourra utiliser un tableau `Etat` à une dimension de longueur n (où n est l'ordre du graphe).

3 Problème de Connexité

3.1 Déterminer les composantes connexes

Le parcours en profondeur d'un graphe non orienté à partir d'un sommet i produit un arbre contenant tous les sommets accessibles depuis i par une chaîne. Les sommets de cet arbre constituent donc une composante connexe du graphe initial.

Pour déterminer les composantes connexes d'un graphe non orienté, il suffit d'appliquer un **parcours en profondeur qui produira un arbre par composante connexe**.

Pour déterminer les composantes connexes d'un graphe orienté, on transforme le graphe en graphe non orienté, puis on applique un parcours en profondeur.

3.2 Déterminer les composantes fortement connexes

Le parcours en profondeur d'un graphe orienté à partir d'un sommet i produit une arborescence de racine i qui contient tous les sommets accessibles depuis i par un chemin. Il **ne s'agit pas d'une composante fortement connexe** du graphe initial

Pour déterminer les composantes fortement connexes nous utiliserons l'algorithme Kosaraju-Sharir et la notion de numérotations préfixes et suffixes.

3.2.1 Numérotation préfixe et suffixe

Les numérotations préfixes et suffixes des sommets sont déterminées lors du parcours en profondeur d'un graphe. La **numérotation préfixe** indique l'ordre dans lequel les sommets ont été **ouverts**, la **numération suffixe** indique l'ordre dans lequel ils ont été **fermés**.

Exemple :

noeud	numéro préfixe	numéro suffixe
1	1	7
2	2	5
3	7	6
4	5	2
5	3	1
6	4	4
7	8	9
8	6	3
9	9	8

FIGURE 5 – Numérotations préfixe et suffixe obtenues par parcours de G_2 à partir de 1.

3.2.2 Algorithme de Kosaraju-Sharir

L'algorithme de Kosaraju-Sharir (1978) détermine les composantes fortement connexes d'un graphe à l'aide de deux parcours en profondeur et de la numération suffixe.
L'algorithme de Kosaraju-Sharir suit les étapes ci-dessous :

1. Faire un parcours en profondeur de G, pour obtenir la numération suffixe,
2. Inverser les arêtes du graphe. On notera le graphe inverse $G^\top$,
3. Faire un parcours en profondeur de $G^\top$ pour lequel le sommet choisit pour chaque racine d'arborescence est le sommet 'non exploré' avec le plus grand numéro suffixe,
4. Les arborescences obtenues lors du deuxième parcours constituent les composantes fortement connexes du graphe initial G.

Exemple.
Reprenons le graphe G_2. Son graphe inversé est $G_2^\top$:

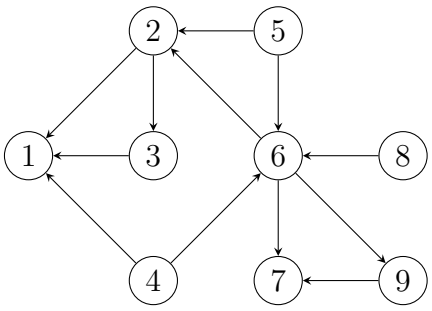

FIGURE 6 – Le graphe inversé $G_2^\top$.

Un parcours en profondeur de $G_2^\top$ en partant toujours du nœuds 'non exploré' qui a le plus grand numéro suffixe nous donne :

nœud	non explorés	explorés	ouverts	fermés	arcs
/	$\{1,2,3,4,5,6,7,8,9\}$	$\emptyset$	$\emptyset$	$\emptyset$	/
7	$\{1,2,3,4,5,6,8,9\}$	$\{7\}$	$\{7\}$	$\emptyset$	/
/	$\{1,2,3,4,5,6,8,9\}$	$\{7\}$	$\emptyset$	$\{7\}$	/
9	$\{1,2,3,4,5,6,8\}$	$\{7,9\}$	$\{9\}$	$\{7\}$	/
/	$\{1,2,3,4,5,6,8\}$	$\{7,9\}$	$\emptyset$	$\{7,9\}$	/
1	$\{2,3,4,5,6,8\}$	$\{1,7,9\}$	$\{1\}$	$\{7,9\}$	/
/	$\{2,3,4,5,6,8\}$	$\{1,7,9\}$	$\emptyset$	$\{7,9,1\}$	/
3	$\{2,4,5,6,8\}$	$\{1,3,7,9\}$	$\{3\}$	$\{7,9,1\}$	/
/	$\{2,4,5,6,8\}$	$\{1,3,7,9\}$	$\emptyset$	$\{7,9,1,3\}$	/
2	$\{4,5,6,8\}$	$\{1,2,3,7,9\}$	$\{2\}$	$\{7,9,1,3\}$	/
/	$\{4,5,6,8\}$	$\{1,2,3,7,9\}$	$\emptyset$	$\{7,9,1,3,2\}$	/
6	$\{4,5,8\}$	$\{1,2,3,6,7,9\}$	$\{6\}$	$\{7,9,1,3,2\}$	/
/	$\{4,5,8\}$	$\{1,2,3,6,7,9\}$	$\emptyset$	$\{7,9,1,3,2,6\}$	/
8	$\{4,5\}$	$\{1,2,3,6,7,8,9\}$	$\{8\}$	$\{7,9,1,3,2,6\}$	/
/	$\{4,5\}$	$\{1,2,3,6,7,8,9\}$	$\emptyset$	$\{7,9,1,3,2,6,8\}$	/
4	$\{5\}$	$\{1,2,3,4,6,7,8,9\}$	$\{4\}$	$\{7,9,1,3,2,6,8\}$	/
/	$\{5\}$	$\{1,2,3,4,6,7,8,9\}$	$\emptyset$	$\{7,9,1,3,2,6,8,4\}$	/
5	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	$\{5\}$	$\{1,2,3,4,6,7,8,9\}$	/
/	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	$\emptyset$	$\{1,2,3,4,5,6,7,8,9\}$	/

L'ordre de parcours des sommets pour $G_2^\top$ est : 7-9-1-3-2-6-8-4-5.
Ici, l'exemple est un peu particulier car aucun arc n'a été parcouru. Ce qui signifie que chaque nœud est une composante fortement connexe à lui seul. On obtient donc 9 arborescences lors du parcours de $G_2^\top$ (chaque nœud isolé est une arborescence) :

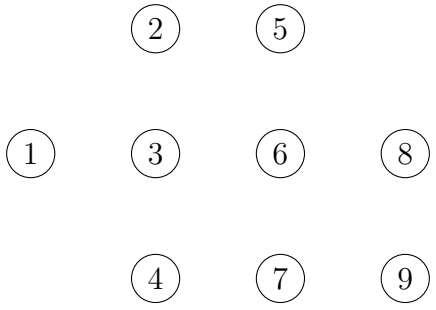

FIGURE 7 – Arborescences obtenues par parcours en profondeur de $G_2^\top$.