

# INFORMATIQUE

2024 - 2025

## Algorithmes sur les Graphes et Combinatoire

Laurent PHILIPPE

SUP<sup>F</sup><sub>C</sub>

---

---

# Algorithmique sur les Graphes et Combinatoire

MASTERS INFORMATIQUE DVL / I2A / ISL  
1ÈRE ANNÉE

---

---

## Chapitre I - Graphes et Algorithmes : Exercices



Centre de télé enseignement  
Filière Informatique  
Domaine Universitaire de la Bouloie  
25030 Besançon Cedex (France)

# Table des matières

1	Modélisation sous forme de graphes	2
1	Choix d'un itinéraire	2
2	Loup, Chèvre et Chou	2
3	Organisation d'une session d'examen	2
4	Planification de travaux	3
2	Propriétés des graphes et arbres	9
5	Propriété des graphes bipartis	9
3	Manipulation des structures de graphes	11
6	Matrice d'adjacence vers <i>FS/APS</i>	11
7	<i>FS/APS</i> vers matrice d'adjacence	11
8	construction de $G^{-1}$	11
9	Calcul de la fermeture transitive d'un graphe	12
4	Parcours en largeur et parcours en profondeur	19
10	Parcours en largeur	19
11	Parcours en profondeur	19
5	Coloration de graphes	26
12	Planification de travaux	26
13	Algorithme de coloration	26

# 1 Modélisation sous forme de graphes

Le but de ces exercices est de proposer une modélisation sous la forme d'un graphe du problème posé avant d'en faire la résolution.

## Exercice 1 : Choix d'un itinéraire

Les durées des trajets entre deux villes sont indiquées comme suit :

Bordeaux ↔ Nantes	4h
Bordeaux ↔ Marseille	9h
Bordeaux ↔ Lyon	12h
Nantes ↔ Paris - Montparnasse	2h
Nantes ↔ Lyon	7h
Paris - Montparnasse ↔ Paris - Lyon	1h (en bus)
Paris - Lyon ↔ Grenoble	4h30
Marseille ↔ Lyon	2h30
Marseille ↔ Grenoble	4h30
Lyon ↔ Grenoble	1h30

**Question 1.1** : *Modéliser les distances entre ces villes sous la forme d'un graphe*

**Question 1.2** : *Comment faire pour aller le plus rapidement possible de Bordeaux à Grenoble ? de Marseille à Nantes ? de Lyon à Nantes ?*

## Exercice 2 : Loup, Chèvre et Chou

Dans une énigme bien connue, un passeur se trouve sur le bord d'un fleuve et doit faire passer de l'autre côté de la rive un loup, une chèvre et un chou mais il ne peut transporter qu'un seul client à la fois et ne peut laisser seuls ensemble ni le loup avec la chèvre, ni la chèvre avec le chou puisqu'ils mangerait l'autre.

**Question 2.1** : *Donner les différents états possibles pour le système.*

Dans l'état initial, le batelier, le loup, la chèvre et le chou sont sur la rive de départ.

**Question 2.2** : *Existe-t-il plusieurs solutions pour le passeur ?*

## Exercice 3 : Organisation d'une session d'examen

Les étudiants A, B, C, D, E, F, G et H doivent passer des examens, chacun dans des disciplines différentes données dans la table 1. Chaque examen occupe une demi-journée.

Matière	Informatique	Gestion	Droit	Maths	Économie
Étudiants	A, B	C, D	C, E, F, G	A, E, F, H	B, F, G, H

TABLE 1 – Matières à passer pour chaque étudiant

**Question 3.1** : *Compte-tenu de ces contraintes, on cherche à organiser la session d'examens de manière à ce que l'ensemble des examens soit terminé le plus tôt possible.*

#### **Exercice 4 : Planification de travaux**

Pour rénover une maison, il est prévu de refaire l'installation électrique (3 jours), de réaménager (5 jours), de carreler la salle de bains (2 jours), de faire le parquet dans le séjour (6 jours), de repeindre les chambres (3 jours). La peinture et le carrelage ne devant être faits qu'après l'installation électrique.

**Question 4.1** : *Si le propriétaire décide de tout faire lui-même, dans quel ordre doit il procéder et quelle est la durée minimale des travaux ?*

**Question 4.2** : *Si la rénovation est faite par une entreprise et que chacune des tâches est accomplie par un employé différent, quelle est la durée minimale des travaux ?*

## Correction Exercice 1 : Choix d'un itinéraire

**Solution question 1.1 :** *Modéliser les distances entre ces villes sous la forme d'un graphe*

La figure 1 modélise les distances entre les villes sous la forme d'un graphe dont les arêtes sont valuées par la distance.

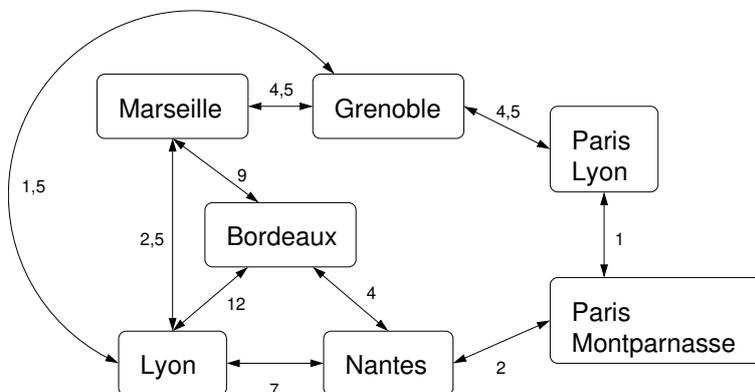


FIGURE 1 – Graphe des villes avec les distances entre elles

**Solution question 1.2 :** *Comment faire pour aller le plus rapidement possible de Bordeaux à Grenoble ? de Marseille à Nantes ? de Lyon à Nantes ?*

Pour aller le plus rapidement d'une ville à une autre, il s'agit de trouver le plus court chemin dans ce graphe pour un départ et une destination en explorant les chemins possibles :

Bordeaux → Grenoble :

- Bordeaux - Marseille - Grenoble : 13,5
- **Bordeaux - Nantes - Paris Montp. - Paris Lyon - Grenoble : 11,5**
- Bordeaux - Lyon - Grenoble : 13,5
- Bordeaux - Marseille - Lyon - Grenoble : 13
- Bordeaux - Nantes - Lyon - Grenoble : 12,5

Marseille → Nantes :

- **Marseille - Lyon - Nantes : 9,5**

Lyon → Nantes :

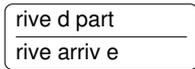
- **Lyon - Nantes : 7**

## Correction Exercice 2 : Loup, Chèvre et Chou

**Solution question 2.1 :** *Donner les différents états possibles pour le système.*

Il est possible d'identifier l'ensemble des états accessibles (ou états possibles) depuis l'état initial. La solution passe par un ensemble d'états intermédiaires avant d'atteindre l'état final. La liste des états est donnée par la figure 2.

L : loup C : chèvre X : chou B : batelier



Etats accessibles

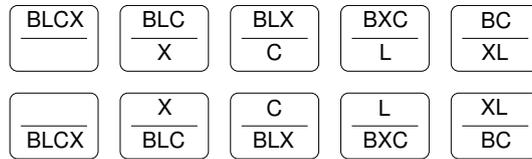
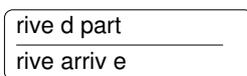


FIGURE 2 – liste des états accessibles depuis la position initiale

**Solution question 2.2 :** *Existe-t-il plusieurs solutions pour le passeur ?*

On réalise le graphe suivant : les états possibles sont les sommets du graphe et les arêtes sont les transitions possibles d'un état possible à un autre état possible. Dans ce cas la solution est un des chemins qui permet de passer de l'état initial à l'état final. La figure 3 illustre ce graphe et donc les solutions possibles pour le batelier s'il veut faire passer la rivière au loup, à la chèvre et au chou. Sur ce graphe deux chemins différents sont clairement visibles : il existe donc bien plusieurs solutions pour le batelier : une fois la chèvre sur l'autre rive il peut transporter soit le chou soit le loup, qu'il laissera sur la rive en ramenant la chèvre.

L : loup C : chèvre X : chou B : batelier



Graphe des tats accessibles

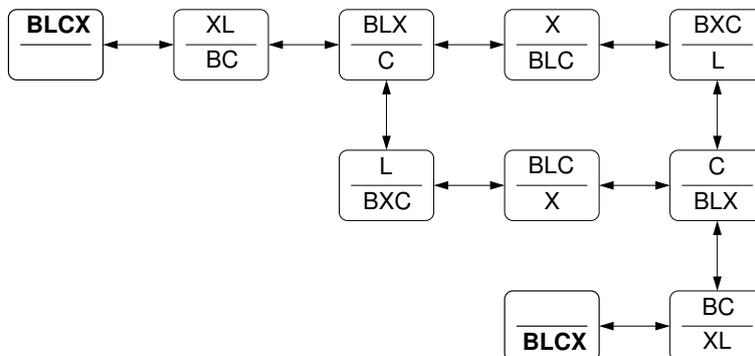


FIGURE 3 – liste des états accessibles depuis la position initiale

**Correction Exercice 3 : Organisation d'une session d'examen**

**Solution question 3.1 :** *Session d'examen la plus courte possible*

	A	B	C	D	E	F	G	H
Informatique	X	X						
Gestion			X	X				
Droit			X		X	X	X	
Maths	X				X	X		X
Économie		X				X	X	X

TABLE 2 – Table des matières à passer par étudiant

Pour commencer nous construisons la table des matières à passer pour chacun des étudiants. Dans cette table nous pouvons remarquer tout d’abord que certaines matières n’ont pas de dépendance entre elles :

- Droit et Informatique ;
- Gestion et Maths ;
- Gestion et Économie ;
- Gestion et Informatique.

Nous pouvons voir ce problème sous un autre angle ; Il s’agit en fait d’un problème de coloration de graphe non orienté. Les sommets de ce graphe sont les matières que les étudiants doivent éventuellement repasser et les arêtes sont les relations entre les matières qui ont en commun des étudiants. Plus formellement on a le graphe  $G(X, E)$  avec :

$$X = \{\text{matières de la session d'examen}\}$$

$$E = \{(x, y) \text{ avec } x \in X \text{ et } y \in X \mid \text{certains étudiants repassent } x \text{ et } y\}$$

Le problème de coloration est qu’on ne peut pas donner la même couleur à deux sommets voisins. Il s’agit de faire une coloration valide, mais en plus de trouver le nombre minimum de couleur pour cela. En associant une couleur à une demi-journée d’examen nous pouvons voir que le résoudre le problème de la coloration nous permet bien de trouver le nombre minimal de demi-journées nécessaires et donc d’avoir la session d’examen la plus courte possible.

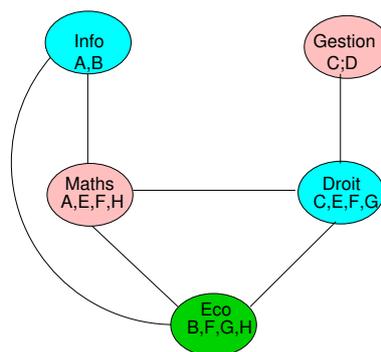


FIGURE 4 – Graphe modélisant l’organisation des examens

La figure 4 illustre une coloration possible du graphe  $G$ . Arbitrairement on peut en effet choisir la coloration : (Bleu ; Informatique et Droit), (Rose ; Gestion et Maths) et (Vert ; Économie).

En fait il y a 3! possibilités de coloration, toutes ne sont pas valides, et il suffit ici d'en choisir une. La solution donnée précédemment est optimale. La durée minimum de la session d'examen est de 3 demi-journées.

### Correction Exercice 4 : Planification de travaux

Pour répondre à cet exercice, on construit le graphe avec les contraintes de précédence : les tâches sont les sommets du graphe et les arcs sont les contraintes de précédence. Les valeurs des tâches sont associées aux sommets, mais pourrait aussi l'être aux arcs. Il est possible d'ajouter une tâche de début et une tâche de fin pour rendre le graphe connexe, comme sur la figure 5.

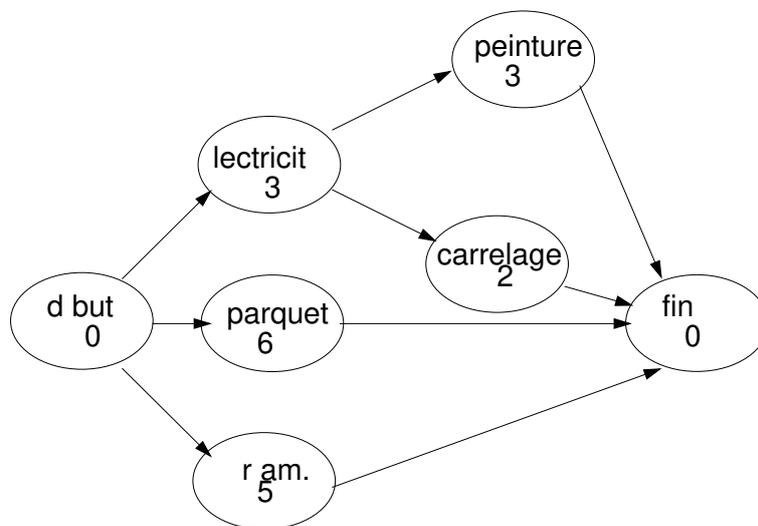


FIGURE 5 – Graphe des tâches à planifier

**Solution question 4.1 :** *Si le propriétaire décide de tout faire lui-même, dans quel ordre doit il procéder et quelle est la durée minimale des travaux ?*

Si le propriétaire décide de tout faire lui-même, la seule contrainte à respecter est de faire l'électricité avant la peinture et le carrelage. Comme il ne peut faire qu'une seule chose à la fois la durée minimale des travaux est la somme des tâches, comme montré à la figure 6 soit 19 jours.



FIGURE 6 – Graphe des tâches exécutées seuls par le propriétaire

**Solution question 4.2 :** *Si la rénovation est faite par une entreprise et que chacune des*

*tâches est accomplie par un employé différent, quelle est la durée minimale des travaux ?*

Il est possible, sur le graphe donné à la figure 5, de calculer la durée minimale du chantier en calculant la longueur de tous les chemins possibles et en prenant le plus grand. Ici nous voyons qu'ils y a deux chemins de longueur 5 et deux chemins de longueur 6. Le chantier durera donc au minimum 6 jours.

La suite des exercices de cette partie concerne la conception d'algorithmes sur les graphes. Pour comprendre les algorithmes qui sont donnés ou valider ceux que vous avez conçus, il est recommandé de les dérouler sur des exemples. Cela peut se faire à la main, sur une feuille de papier, ou en programmant puis en testant sur un exemple. Pour le déroulement manuel, vous pouvez vous inspirer des figures du cours illustrant le parcours en largeur (figure 1.10) ou la recherche d'un plus court chemin (figure 1.13). Si vous souhaitez programmer, vous pouvez utiliser n'importe quel langage, avec les structures de données décrites dans le cours. À noter qu'il existe des bibliothèques implémentant les structures et fonctions de base sur les graphes. Je vous recommande la bibliothèque JGraphT<sup>1</sup> qui implémente les graphes sous la forme d'objets et propose la plupart des fonctions de manipulation des sommets et des arêtes. Quatre exemples simples - création d'un graphe, recherche en profondeur, import/export d'un graphe au format DOT et affichage - sont donnés en accompagnement pour vous faciliter le démarrage.

## 2 Propriétés des graphes et arbres

### **Exercice 5 : Propriété des graphes bipartis**

Un graphe est biparti si et seulement si il n'admet pas de cycle de longueur impaire.

**Question 5.1 :** *Montrez par récurrence sur le nombre de sommets  $n$  du graphe biparti qu'un graphe biparti n'admet pas de cycle de longueur impaire.*

**Question 5.2 :** *Montrez qu'un graphe qui n'admet que des cycles de longueur paire est un graphe biparti.*

---

1. <https://jgrapht.org/>

## Correction Exercice 5 : Propriété des graphes bipartis

**Solution question 5.1 :** *Montrez par récurrence sur le nombre de sommets  $n$  du graphe biparti qu'un graphe biparti n'admet pas de cycle de longueur impaire.*

Soit un graphe biparti  $G = (V_1, V_2, E)$  contenant un cycle  $(x_1, \dots, x_n, x_1)$ . Nous pouvons supposer que  $x_1 \in V_1$ , sans perte de généralité (cette hypothèse ne réduit pas la portée de notre preuve puisque nous pouvons raisonner de manière symétrique si  $x_1 \in V_2$ ). Alors pour tout  $k$ ,  $x_{2k} \in V_2$  et  $x_{2k+1} \in V_1$  (les sommets impairs du cycle sont dans  $V_1$  et les sommets pairs sont dans  $V_2$ ) puisque :

1. chaque arête de  $E$  a une extrémité dans chaque ensemble d'après la propriété biparti,
2.  $x_1 \in V_1$ ,
3. par récurrence sur  $k$ , si la propriété est vraie pour  $k-1$ , alors le sommet  $x_{2k-1} \in V_1$ .  
Donc l'arête qui relie  $x_{2k-1}$  à  $x_{2k}$  va de  $V_1$  à  $V_2$  et  $x_{2k} \in V_2$ . L'arête qui relie  $x_{2k}$  à  $x_{2k+1}$  va de  $V_2$  à  $V_1$  et  $x_{2k+1} \in V_1$ .

Pour que le cycle  $(x_1, \dots, x_n, x_1)$  soit de longueur impaire, il faut qu'il parcourt un nombre impair d'arêtes et le nombre  $n$ , l'indice de  $x_n$ , est donc impair. De ce fait  $x_n$  devrait être dans  $V_1$  et l'arête  $(x_n, x_1)$  est entre deux sommets de  $V_1$ , ce qui est contraire avec l'hypothèse de graphe biparti (un graphe biparti ne contient pas d'arêtes entre deux sommets d'une même partie). Il n'existe donc pas de graphe biparti acceptant des cycles de longueur impaire.

**Solution question 5.2 :** *Montrez qu'un graphe qui n'admet que des cycles de longueur paire est un graphe biparti.*

Soit  $x_0$  un sommet de  $G$  et soit  $A$  l'ensemble des sommets à distance paire du sommet  $x_0$  (la longueur du plus court chemin entre  $x_0$  et chacun des éléments de  $A$  est paire). Soit l'ensemble de sommets  $B$  tel que  $B = G - A$ . La décomposition en  $A$  et  $B$  est alors une bipartition du graphe  $G$  ssi il n'y a aucune arête entre deux sommets de  $A$  ou deux sommets de  $B$ . Supposons qu'il existe une arête  $(x_a, x_{a'})$  dans  $A$ , alors par hypothèse il existe un chemin de longueur minimale  $c$  de longueur paire de  $x_0$  à  $x_a$  et un chemin de longueur minimale  $c'$  de longueur paire de  $x_0$  à  $x_{a'}$ . Soit  $x_w$  le sommet commun aux deux chemins  $c$  et  $c'$  le plus distant de  $x_0$ . Nous pouvons former le cycle passant par  $x_w$ ,  $x_a$  et  $x_{a'}$  puisqu'il existe un chemin de  $x_w$  à  $x_a$ , une arête de  $x_a$  à  $x_{a'}$  et un chemin de  $x_w$  à  $x_{a'}$ . Or ce cycle est de longueur impaire car les distances  $(x_w, x_a)$  et  $(x_w, x_{a'})$  ont la même parité soit paire, soit impaire, et leur somme donne forcément un nombre pair. Nous arrivons donc à une contradiction et il ne peut y avoir d'arêtes entre deux sommets de  $A$ . En raisonnant de manière symétrique nous prouvons qu'il ne peut y avoir non plus d'arête entre deux sommets de  $B$  et les ensembles  $A$  et  $B$  constituent bien une partition du graphe  $G$  qui est, de ce fait, biparti.

### 3 Manipulation des structures de graphes

Le but de ces exercices est de se familiariser avec la manipulation des graphes et leurs structures de données, passer d'une structure à une autre.

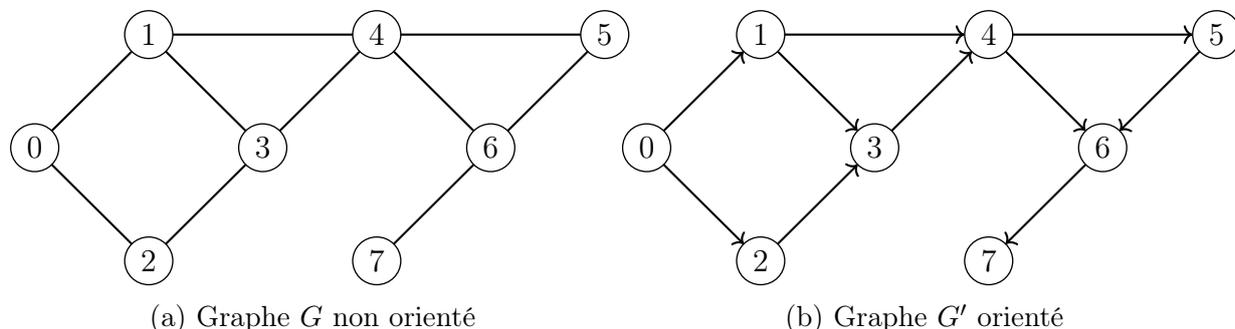


FIGURE 7 – Graphes non orienté et orienté

#### **Exercice 6 : Matrice d'adjacence vers $FS/APS$**

Soit un graphe orienté  $G$  défini par sa matrice d'adjacence  $A$ .

**Question 6.1 :** *Écrire un algorithme qui permet de construire la file des successeurs  $FS$  et le tableau des adresses des premiers successeurs  $APS$  du graphe  $G$ .*

**Question 6.2 :** *Comment modifier l'algorithme pour qu'il fonctionne pour un graphe non orienté ?*

**Question 6.3 :** *Construire les matrices d'adjacence et dérouler l'algorithme sur les graphes  $G$  et  $G'$  donnés à la figure 7.*

#### **Exercice 7 : $FS/APS$ vers matrice d'adjacence**

Soit un graphe orienté  $G$  défini par sa file des successeurs  $FS$  et son tableau des adresses des premiers successeurs  $APS$ .

**Question 7.1 :** *Écrire un algorithme construisant la matrice d'adjacence  $A$  du graphe  $G$ .*

**Question 7.2 :** *Comment modifier l'algorithme pour qu'il fonctionne pour un graphe non orienté ?*

**Question 7.3 :** *Construire les structures  $APS$  et  $FS$  et dérouler l'algorithme sur les graphes  $G$  et  $G'$  donnés à la figure 7a.*

#### **Exercice 8 : construction de $G^{-1}$**

À partir d'un graphe orienté  $G$ , on construit le graphe inverse  $G^{-1}$  en inversant le sens de tous les arcs orientés de  $G$ . Nous disposons des tables  $FS$  et  $APS$  du graphe  $G$ .

**Question 8.1 :** *Écrire un algorithme, linéaire en nombre de sommets, qui calcule le tableau des prédécesseurs  $FP$  (qui est aussi l'ensemble des suivants dans  $G^{-1}$ ) et le tableau*

*des adresses des premiers prédécesseurs APP (qui est le tableau des adresses des premiers suivants dans  $G^{-1}$ ).*

Il est conseillé de procéder en déterminant d'abord le nombre de prédécesseurs de chaque sommet en faisant un parcours linéaire de  $FS$ , puis en remplissant  $APP$  et  $FP$ .

### **Exercice 9 : Calcul de la fermeture transitive d'un graphe**

Soit un graphe orienté  $G$  défini par sa matrice d'adjacence  $A$ .

**Question 9.1 :** *Écrire un algorithme permettant de calculer la fermeture transitive du graphe  $G$ .*

**Question 9.2 :** *Comment modifier l'algorithme pour qu'il fonctionne pour un graphe non orienté ?*

## Correction Exercice 6 :

**Solution question 6.1 :** *Écrire un algorithme qui permet de construire la file des successeurs  $FS$  et le tableau des adresses des premiers successeurs  $APS$  du graphe  $G$ .*

On rappelle que les structures  $APS$  et  $FS$  telles que définies dans le cours supposent que les sommets sont numérotés de 0 à  $n - 1$  et la valeur -1 sert de séparateur dans  $FS$ . La numéro de ligne  $i$  dans la matrice  $A$  correspond donc au sommet  $i$ .

L'algorithme 1 permet de créer les structures  $APS$  et  $FS$  d'un graphe  $G$  à partir de la matrice d'adjacence  $A$ . Le principe est de parcourir la matrice ligne par ligne (lignes 2 à 9 de l'algorithme), sur chaque ligne (lignes 4 à 7 de l'algorithme) les successeurs du sommet sont les sommets adjacents, donc pour lesquels l'élément de la matrice vaut 1 (ligne 5). Ces successeurs sont ajoutés à la structure  $FS$  (ligne 6) et l'indice de parcours  $indiceCourant$  augmenté. À la fin de la ligne du sommet courant (ligne 9), un séparateur est ajouté dans  $FS$  puisqu'on passe au sommet suivant.

---

### Algorithme 1 : Matrice d'adjacence $\rightarrow FS/APS$

---

**Données :**  $n, m$  : nombres de sommets et d'arêtes

$A$  : matrice d'adjacence

$indiceCourant$  : indice de parcours de  $FS$  initialement à 0

**Résultat :**  $FS[0..n + m - 1]$  : tableau des successeurs des sommets

$APS[0..n - 1]$  : tableau des adresses des premiers successeurs

```
1 début
2   pour  $i$  de 0 à  $n - 1$  faire
3      $APS[i] \leftarrow indiceCourant$ ; /* l'adresse du premier successeur du
4       sommet  $i$  est la position courante du remplissage de  $FS$  */
5     pour  $j$  de 0 à  $n - 1$  faire
6       si  $A[i, j] = 1$  alors
7          $FS[indiceCourant] \leftarrow j$ 
8          $indiceCourant \leftarrow indiceCourant + 1$ 
9      $FS[indiceCourant] \leftarrow -1$  /* ajout du séparateur */
10     $indiceCourant \leftarrow indiceCourant + 1$ 
```

---

**Solution question 6.2 :** *Comment modifier l'algorithme pour qu'il fonctionne pour un graphe non orienté ?*

Cet algorithme fonctionne que le graphe soit orienté ou non. La seule chose qui change est la taille du tableau  $FS$  qui passe de  $n + m$  à  $n + 2m$  puisque les arêtes, contrairement aux arcs, sont comptées deux fois : une fois chez chacun des sommets incidents.

**Solution question 6.3 :** *Construire les matrices d'adjacence et dérouler l'algorithme sur les graphes  $G$  et  $G'$  donnés à la figure 7.*

La matrice  $A$  donne la matrice d'adjacence du graphe  $G$  et la matrice  $A'$  donne la matrice d'adjacence du graphe  $G'$ .

$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \quad A' = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

### Correction Exercice 7 :

**Solution question 7.1 :** *Écrire un algorithme construisant la matrice d'adjacence  $A$  du graphe  $G$ .*

Comme précédemment les sommets sont numérotés de 0 à  $n - 1$  et correspondent aux indices de la matrice.

Une solution est donnée par l'algorithme 2. Celui-ci parcourt le tableau  $FS$  pour remplir les lignes de la matrice (lignes 2 à 6). Lorsque la valeur du tableau est à 0 (ligne 3), il s'agit d'un séparateur entre sommets, on passe à la ligne de la matrice suivante (ligne 4)), sinon les numéros des sommets successeurs ( $FS[parcours]$ ) donnent les éléments de la matrice à mettre à 1 (ligne 6).

---

#### **Algorithme 2 :** $FS/APS \rightarrow$ matrice d'adjacence

---

**Données :**  $n, m$  : nombres de sommets et d'arêtes  
 $FS[0..n + m - 1]$  : tableau des successeurs  
 $succ$  : indice de parcours de  $FS$ , initialement à 0  
 $l$  : numéro de ligne dans la matrice  $A$ , initialement à 0

**Résultat :**  $A[0..n - 1, 0..n - 1]$  : la matrice d'adjacence

```

1 début
2   pour  $succ$  de 0 à  $n + m - 1$  faire
3     si  $FS[succ] = -1$  alors
4       |  $l \leftarrow l + 1$  /* passage au sommet suivant          */
5       sinon
6       |  $A[l, FS[succ]] \leftarrow 1$ 

```

---

**Solution question 7.2 :** *Comment modifier l'algorithme pour qu'il fonctionne pour un graphe non orienté ?*

Le fait que le graphe soit orienté ou non ne modifie en rien la manière de fonctionner de l'algorithme. La seule chose qui doit être changée dans l'algorithme est la taille du tableau  $FS$  qui passe de  $n + m$  à  $n + 2m$  puisque chaque arête  $y$  est comptée deux fois.

**Solution question 7.3 :** Construire les tableaux  $APS$  et  $FS$  et dérouler l'algorithme sur les graphes  $G$  et  $G'$  donnés à la figure 7.

Les tables 3 et 4 donnent les tableaux  $APS$  et  $FS$  du graphe  $G$ .

0	4	7	10	14	19	22	26
---	---	---	----	----	----	----	----

TABLE 3 – Table  $APS$  du graphe  $G$

1	2	-1	0	3	4	-1	0	3	-1	1	2	4	-1	1	3	...
...	5	6	-1	4	6	-1	4	5	7	-1	6	-1				

TABLE 4 – Table  $FS$  du graphe  $G$

### Correction Exercice 8 :

**Solution question 8.1 :** *Écrire un algorithme linéaire en nombre de sommets qui calcule le tableau des prédécesseurs  $FP$  et le tableau des adresses des premiers prédécesseurs  $APP$ .*

L'algorithme 3 donne une solution à la construction des tables  $APP$  et  $FP$ . Comme cela a été suggéré dans l'énoncé, l'algorithme commence par calculer le nombre de prédécesseurs pour chaque sommet (lignes 2 et 3) qu'il mémorise dans le tableau  $NP$ . Il construit ensuite une première version du tableau  $APP$  sur la base de ce nombre de prédécesseurs (lignes 4 à 6) : pour le sommet  $i$  l'adresse du premier prédécesseur sera l'adresse du premier prédécesseur du sommet précédent ( $APP[i - 1]$ ), plus le nombre de prédécesseurs de ce sommet, ( $nS[i - 1]$ ) qui vient d'être calculé, plus 1 pour le séparateur. L'algorithme remplit ensuite le tableau  $FP$  (lignes 6 à 11) en parcourant le tableau  $FS$  : si la valeur courante est nulle, alors il change de sommet (ligne 8), sinon il ajoute le sommet courant au tableau  $FP$  à la première place déterminée par le tableau  $APP$  (ligne 10) et incrémente la valeur du tableau  $APP$  pour mémoriser le prochain prédécesseur à la case suivante. Noter ici que l'indice du tableau  $APP$  est à chaque fois diminué de 1 pour tenir compte du décalage entre les numéros de sommets et les indices des tables. Pour finir l'algorithme reconstruit le tableau  $APP$  (lignes 12 à 15) qui a été modifié.

---

**Algorithme 3** : construction de  $FP/APP$  ou calcul de  $G^{-1}$ 

---

**Données** :  $n, m$  : nombres de sommets et d'arêtes  
 $FS[0..n + m - 1]$  : tableau des successeurs  
 $APS[0..n - 1]$  : tableau des adresses des premiers successeurs  
 $NP[0..n - 1]$  : nb prédécesseurs par sommet, **initialisé à 0**  
 $som$  : numéro du sommet courant, **initialisé à 0**

**Résultat** :  $FP[0..n + m - 1]$  : tableau des prédécesseurs  
 $APP[0..n - 1]$  : tableau des adresses des premiers prédécesseurs

```
1 début
2   pour  $i$  de 0 à  $n + m - 1$  faire
3     si  $FS[i] \neq -1$  alors  $NP[FS[i]] ++$ 
4     /* remplissage de  $APP$  */
5      $APP[0] \leftarrow 0$ 
6     pour  $i$  de 1 à  $n - 1$  faire  $APP[i] \leftarrow APP[i - 1] + NP[i - 1] + 1$ 
7     /* remplissage de  $FP$  */
8     pour  $i$  de 0 à  $n + m - 1$  faire  $FP[i] \leftarrow -1$ 
9     pour  $i$  de 0 à  $n + m - 1$  faire
10      si  $FS[i] = -1$  alors  $som \leftarrow som + 1$ 
11      sinon
12         $FP[APP[FS[i]]] \leftarrow som$ 
13         $APP[FS[i]] ++$ 
14      /* reconstruction de  $APP$  */
15       $APP[0] \leftarrow 0$  /* Premier sommet */
16       $som \leftarrow 1$ 
17      pour  $i$  de 0 à  $n + m - 1$  faire
18        si  $FP[i] = -1$  alors
19           $APP[som] \leftarrow i + 1$ 
20           $som \leftarrow som + 1$ 
```

---

**Correction Exercice 9** : Calcul de la fermeture transitive d'un graphe

**Solution question 9.1** : Écrire un algorithme permettant de calculer la fermeture transitive d'un graphe.

Comme nous l'avons vu dans le cours, la fermeture transitive d'un graphe est la contraction de toutes les relations du type  $a\mathcal{R}b$  et  $b\mathcal{R}c$  en  $a\mathcal{R}c$ . C'est-à-dire qu'un arc, ou une arête, est ajouté entre deux sommets  $a$  et  $c$  si il existe un chemin entre eux. Comme l'opération est répétée tant qu'il est possible d'ajouter des arcs, ou des arêtes, il y aura un arc ou une arête entre toute paire de sommets reliés par un chemin. Le calcul de la fermeture transitive  $G^*$  d'un graphe  $G$  permet donc de savoir s'il existe un chemin d'un sommet  $x$  vers un sommet  $y$  dans  $G$  en examinant si le sommet  $x$  a comme voisin  $y$  dans  $G^*$ .

Si  $A$  est la matrice d'adjacence du graphe  $G$ ,  $A^*$  est la matrice d'adjacence du graphe  $G^*$ , la fermeture transitive de  $G$  se calcule mathématiquement comme suit :

$$A^* = A + A^2 + A^3 + \dots + A^{n-1}$$

où  $n$  est le nombre de sommets du graphe et  $A^k$  est le produit booléen de la matrice  $A$   $k$  fois par elle-même<sup>2</sup>. Dans ce cas  $A^*[x, y] = 1$  signifie qu'il existe un chemin qui relie  $x$  à  $y$  et  $A^*[x, y] = 0$  sinon.

La complexité associée à ce calcul est  $O(n^4)$  ce qui est lourd mais il est heureusement possible de concevoir des algorithmes qui ne font pas le calcul des puissances successives et ont des complexités moins élevées. Il existe plusieurs approches pour réaliser ce calcul, nous donnons ici l'approche par compression de chemin et celle de Roy-Warshall.

L'approche par compression de chemin, donnée par l'algorithme 4, suit simplement de les arcs  $a \rightarrow b$  et  $b \rightarrow c$  pour ajouter  $c$  dans le voisinage de  $a$  autant que cela est nécessaire. Il s'agit quelque part de faire de la compression de chemin le long d'un parcours en largeur à partir de tous les sommets de  $G$ . Puisque le graphe peut avoir un diamètre maximal de  $n - 1$ , il est nécessaire de faire cette compression  $n - 1$  fois. Ici, suivant le sens de découverte des sommets, certains chemins peuvent avoir déjà été compressés.

---

**Algorithme 4 :** Algorithme de fermeture transitive type *compression de chemin*

---

**Données :**  $n$  : nombres de sommets

$A$  : matrice d'adjacence

**Résultat :**  $A^*$  : matrice de la fermeture transitive

```

1 début
2    $A^* \leftarrow A$ 
3   pour  $p$  de 1 à  $n - 1$  faire
4     pour  $i$  de 0 à  $n - 1$  faire
5       pour  $j$  de 0 à  $n - 1$  faire
6         si  $A^*[i, j] = 1$  alors
7           pour  $k$  de 0 à  $n - 1$  faire
8             si  $A^*[j, k] = 1$  alors  $A^*[i, k] \leftarrow 1$ 

```

---

Cet algorithme a une complexité de  $O(n^4)$ , mais il est plus simple que l'algorithme d'addition des puissances de matrices. Ce dernier ne présente donc un intérêt que si nous disposons des puissances de la matrice  $A$ , par exemple pour mémoriser les chemins d'une certaine longueur.

L'approche proposée par Roy et Warshall consiste elle à parcourir chaque sommet et d'ajouter des liens entre tous ses prédécesseurs et tous ses successeurs. Il s'agit également d'une compression de chemin, même s'il est moins facile de s'en persuader. Il faut imaginer que chaque examen d'un sommet conduit à réduire la distance de un entre les sommets, quelque soit l'ordre d'examen des sommets. Ainsi un graphe linéaire conduit à rendre à la

---

2. Puisque la valeur 1 dans la matrice d'adjacence traduit un lien entre les sommets considérés, les puissances d'une matrice d'adjacence ne sont donc composées que de valeurs 0 ou 1 qui traduisent l'existence d'un chemin dont la longueur est égale à la puissance.

fin du processus, tous les sommets à distance 1 les uns des autres. L'algorithme 5 donne cette solution avec la matrice d'adjacence  $A$ . Même si sa similitude avec l'algorithme 4 est notable, la logique de fonctionnement reste différente et la complexité finale est inférieure, en  $O(n^3)$ .

---

**Algorithme 5 :** Algorithme de fermeture transitive de *Roy-Warshall*

---

**Données :**  $n$  : nombres de sommets

$A$  : matrice d'adjacence

**Résultat :**  $A^*$  : matrice de la fermeture transitive

```

1 début
2    $A^* \leftarrow A$ 
3   pour  $i$  de 0 à  $n - 1$  faire
4     pour  $j$  de 0 à  $n - 1$  faire
5       si  $A[j, i] = 1$  alors
6         pour  $k$  de 0 à  $n - 1$  faire
7           si  $A[i, k] = 1$  alors  $A^*[j, k] \leftarrow 1$ 

```

---

Pour finir, il est également possible d'utiliser des algorithmes de calcul de plus courts chemins, tels que celui de Floyd-Warshall donné dans le cours, pour trouver la fermeture transitive du graphe. Lorsque l'algorithme donne une distance entre deux sommets cela signifie qu'un chemin existe entre les deux. Au besoin, ces algorithmes peuvent être adaptés pour de rendre qu'une matrice d'adjacence classique. À noter que la complexité de ces algorithmes est comparable à celle de l'algorithme 5.

**Solution question 9.2 :** *Comment modifier l'algorithme pour qu'il fonctionne pour un graphe non orienté ?*

Puisque l'algorithme utilise la matrice d'adjacence il est utilisable quelque soit le type de graphe : orienté ou non. À noter que la fermeture transitive d'un graphe orienté conduit à la formation de cliques dans les composantes connexes.

## 4 Parcours en largeur et parcours en profondeur

L'objectif des exercices qui suivent est de maîtriser manipulation des parcours de graphe en largeur et en profondeur.

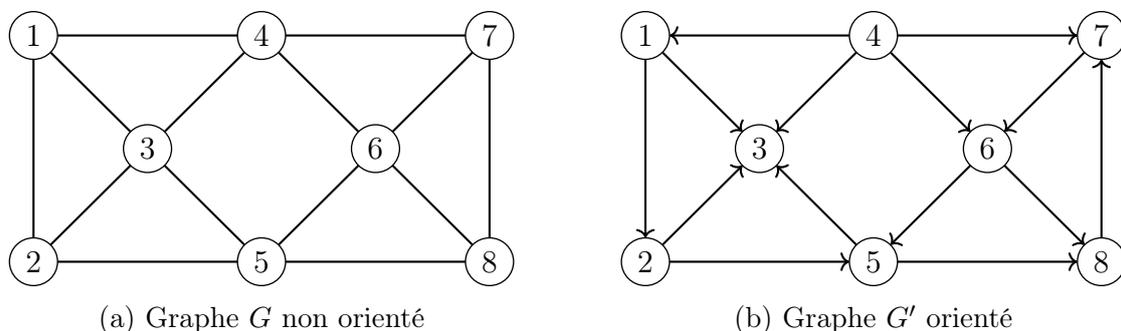


FIGURE 8 – Graphes non orienté et orienté

### Exercice 10 : Parcours en largeur

Dans cet exercice nous nous intéressons aux parcours largeur d'un graphe.

**Question 10.1 :** Donner un parcours en largeur de  $G$  (figure 8a) et son arborescence associée.

**Question 10.2 :** Les permutations  $L_1 = (3, 1, 2, 5, 6, 4, 7, 8)$  et  $L_2 = (1, 2, 3, 4, 5, 6, 7, 8)$  sont-elles des parcours en largeur de  $G$  ? Si oui, donner une arborescence associée. Est-elle unique ?

**Question 10.3 :** Donner un parcours en largeur pour  $G'$  (figure 8b) ainsi que l'arborescence associée.

**Question 10.4 :** Donner un algorithme récursif générique<sup>3</sup> de parcours en largeur d'un graphe  $G = (V, E)$  à partir du sommet  $s$ .

**Question 10.5 :** Dérouler l'algorithme sur le graphe de la figure 8a et vérifiez les résultats de la question 2

### Exercice 11 : Parcours en profondeur

**Question 11.1 :** Donner un parcours en profondeur de  $G$  (figure 8a) et son arborescence associée.

**Question 11.2 :**  $L_1 = (5, 3, 1, 2, 4, 6, 8, 7)$  et  $L_2 = (5, 8, 7, 3, 4, 1, 2, 6)$  sont-ils des parcours en profondeur ? Si oui, donner une arborescence associée. Est-elle unique ?

**Question 11.3 :** Donner un parcours en profondeur pour  $G'$  (figure 8b) ainsi que l'arborescence associée.

**Question 11.4 :** En confrontant les définitions des parcours en largeur et en profondeur,

3. Nous rappelons qu'un algorithme générique ne fait pas de supposition quant à la structure de données de mémorisation du graphe. Il se base alors sur des notations ensemblistes pour  $G = (V, E)$ .

*dire pourquoi la structure de données contenant les sommets visités encore ouvert du parcours en profondeur est cette fois ci une pile.*

**Question 11.5 :** *Donner l'état de la pile et de la liste des sommets au cours du parcours donné en 1.*

**Question 11.6 :** *Donner un algorithme récursif de parcours en profondeur d'un graphe  $G = (V, E)$  à partir du sommet  $s$ .*

**Question 11.7 :** *Dérouler l'algorithme sur le graphe de la figure 8a et vérifiez les résultats de la question 2*

## Correction Exercice 10 : Parcours en largeur

**Solution question 10.1 :** Donner un parcours en largeur de  $G$  (figure 8a) et son arborescence associée.

$L = (8, 5, 6, 7, 2, 3, 4, 1)$  est un parcours en largeur. L'arborescence associée est donnée à la figure 9;

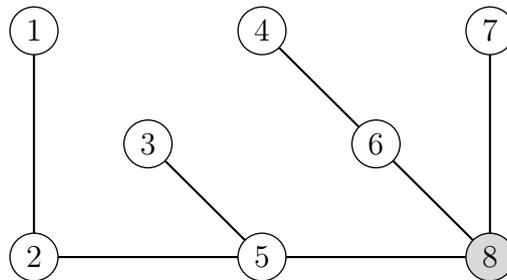


FIGURE 9 – Arborescence associée au parcours en largeur  $L = (8, 5, 6, 7, 2, 3, 4, 1)$

**Solution question 10.2 :** Les permutations  $L_1 = (3, 1, 2, 5, 6, 4, 7, 8)$  et  $L_2 = (1, 2, 3, 4, 5, 6, 7, 8)$  sont-elles des parcours en largeur? Si oui, donner une arborescence associée. Est-elle unique?

$L_1 = (3, 1, 2, 5, 6, 4, 7, 8)$  n'est pas un parcours en largeur car le sommet 6 n'est pas adjacent au premier sommet ouvert pour  $L[3, 1, 2, 5]$  dans le graphe 8a.

$L_2$  est un parcours en largeur. La figure 10 montre l'arborescence associée. Cette arborescence est unique car les arêtes sont choisies en fonction du parcours, en fermant les sommets dans l'ordre du parcours qui, lui, est unique. Les arêtes traduisent la relation unique père-fils établie par le parcours.

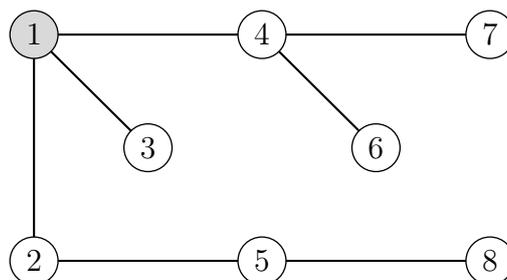


FIGURE 10 – Arborescence associée au parcours en largeur  $L_2 = (1, 2, 3, 4, 5, 6, 7, 8)$

**Solution question 10.3 :** Donner un parcours en largeur pour  $G'$  (figure 8b) ainsi que l'arborescence associée.

$L = (4, 3, 6, 7, 1, 8, 5, 2)$  est un parcours en largeur du graphe 8b. La figure 11 donne l'arborescence associée à  $L$ .

**Solution question 10.4 :** Donner un algorithme récursif générique de parcours en largeur d'un graphe  $G = (V, E)$  à partir du sommet  $s$ .

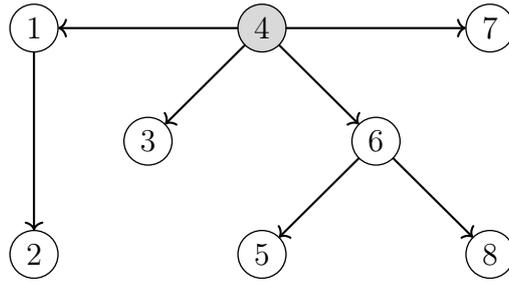


FIGURE 11 – Arborescence associée au parcours en largeur  $L = (4, 3, 6, 7, 1, 8, 5, 2)$

L’algorithme 6 réalise un parcours en largeur en appelant récursivement la fonction *parcoursLargeurRec*.

---

**Algorithme 6** : parcours en largeur récursif d’un graphe  $G(V, E)$  à partir du sommet  $s$

---

**Données** :  $f$  : file des sommets en cours **init** à vide  
 $couleur[u]$  :  $u \in \{BLANC, GRIS\}$ , **init** à  $BLANC$ ,  $\forall u \neq s \in V, GRIS$  pour  $s$

```

1 Fonction parcoursLargeurRec( $G, s$ ) :
2 début
3   pour chaque sommet  $v$  voisin de  $s$  faire
4     si  $couleur[v] = BLANC$  alors
5        $couleur[v] \leftarrow GRIS$ 
6        $f.add(v)$ 
7   si  $\neg f.empty()$  alors parcoursLargeurRec( $G, f.poll()$ )
8   retourner

```

---

Les lignes 3 à 5 permettent d’initialiser la file si nous sommes au premier appel de l’algorithme. Pour tous les appels successifs la couleur associée au sommet aura été mise à  $GRIS$  avant que le sommet ne soit mis dans la file. Ensuite, si la file n’est pas vide, l’algorithme en prend le premier élément et met tous les voisins de ce sommet dans la file. Ainsi les sommets sont bien entrés dans la file dans l’ordre de leur niveau par rapport au sommet d’origine.

Dans le cas du parcours en largeur nous pouvons constater que la formulation récursive n’est pas plus concise que la formulation itérative, l’utilisation d’un objet file étant toujours indispensable au bon déroulement.

### Correction Exercice 11 : Parcours en profondeur

**Solution question 11.1** : Donner un parcours en profondeur de  $G$  (figure 8a) et son arborescence associée.

$L = (6, 5, 3, 4, 7, 8, 1, 2)$  est un parcours en profondeur pour le graphe 8a. L’arborescence

associée est donnée à la figure 12 ;

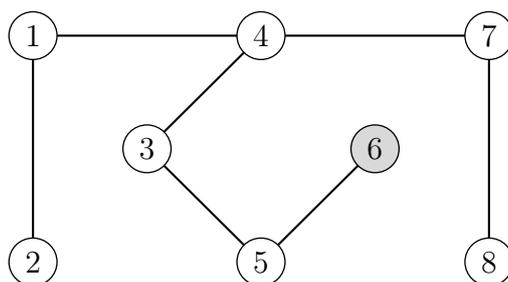


FIGURE 12 – Arborecence associée au parcours en profondeur  $L = (6, 5, 3, 4, 7, 8, 1, 2)$

**Solution question 11.2 :**  $L_1 = (5, 3, 1, 2, 4, 6, 8, 7)$  et  $L_2 = (5, 8, 7, 3, 4, 1, 2, 6)$  sont-ils des parcours en profondeur ? Si oui, donner une arborescence associée. Est-elle unique ?

$L_1 = (5, 3, 1, 2, 4, 6, 8, 7)$  est un parcours en profondeur. L'arborescence donnée à la figure 13 est unique car elle traduit la relation unique père-fils établie par le parcours.

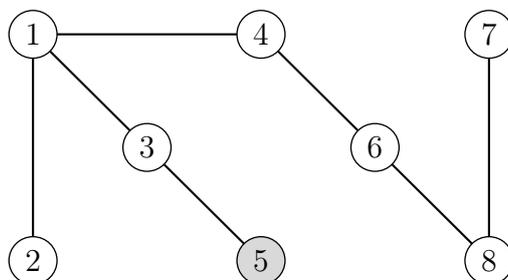


FIGURE 13 – Arborecence associée au parcours en profondeur  $L_1 = (5, 3, 1, 2, 4, 6, 8, 7)$

$L_2 = (5, 8, 7, 3, 4, 1, 2, 6)$  n'est pas un parcours en profondeur de 8a car 3 n'est pas dans le voisinage de 7.

**Solution question 11.3 :** Donner un parcours en profondeur pour  $G'$  (figure 8b) ainsi que l'arborescence associée.

$L = (4, 1, 3, 2, 5, 8, 7, 6)$  est un parcours en profondeur de  $G'$  (figure 7b). L'arborescence associée est donnée à la figure 14 ;

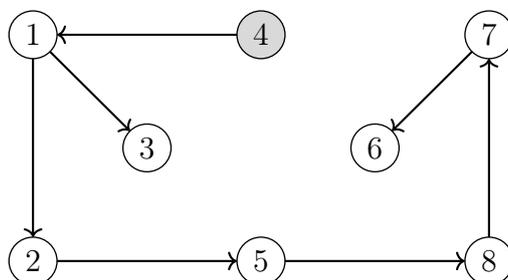


FIGURE 14 – Arborecence associée au parcours en profondeur  $L_1 = (4, 1, 3, 2, 5, 8, 7, 6)$

**Solution question 11.4 :** En confrontant les définitions des parcours en largeur et en

profondeur, dire pourquoi la structure de données contenant les sommets visités encore ouvert du parcours en profondeur est cette fois ci une pile.

Dans une pile, l'élément retiré est le dernier élément arrivé sur la pile. Ceci correspond donc bien à la définition des ajouts des voisins des sommets déjà visités. En effet, le prochain sommet du parcours est un voisin du dernier sommet ouvert du parcours en cours, donc de la pile.

**Solution question 11.5 :** Donner l'état de la pile et de la liste des sommets au cours du parcours donné en 1.

L'état de la pile et de la liste des sommets, pour le parcours  $L = (6, 5, 3, 4, 7, 8, 1, 2)$ , est le suivant :

P	L
6	6
6,5	6,5
6,5,3	6,5,3
6,5,3,4	6,5,3,4
6,5,3,4,7	6,5,3,4,7
6,5,3,4,7,8	6,5,3,4,7,8
6,5,3,4,7	6,5,3,4,7,8
6,5,3,4	6,5,3,4,7,8
6,5,3,4,1	6,5,3,4,7,8,1
6,5,3,4,1,2	6,5,3,4,7,8,1,2
6,5,3,4,1	6,5,3,4,7,8,1,2
6,5,3,4	6,5,3,4,7,8,1,2
6,5,3	6,5,3,4,7,8,1,2
6,5	6,5,3,4,7,8,1,2
6	6,5,3,4,7,8,1,2
-	6,5,3,4,7,8,1,2

**Solution question 11.6 :** Donner un algorithme récursif de parcours en profondeur d'un graphe  $G$  à partir du sommet  $s$ .

L'algorithme récursif du parcours en profondeur est très concis. En effet les appels récursifs permettent de se passer de la pile puisque celle-ci est implicitement mise en place lors des appels imbriqués. Contrairement au parcours en largeur la récursivité simplifie grandement l'algorithme

---

**Algorithme 7 :** Algorithme récursif de parcours en profondeur d'un graphe  $G(V, E)$

---

**Données :**  $couleur[u] : u \in \{BLANC, GRIS\}$ , initialisée à  $BLANC, \forall u \in V$

---

1 **Fonction** *parcoursProfondeur*( $G, u$ )

2 **début**

3      $couleur[u] \leftarrow GRIS$

4     **pour chaque** sommet  $v$  voisin de  $u$  **faire**

5         **si**  $couleur[v] = BLANC$  **alors** *parcoursProfondeur*( $G, v$ )

---

**NB** : Attention la maîtrise des algorithmes de parcours est très importante car ceux-ci sont très souvent à la base d'algorithmes sur les graphes.

## 5 Coloration de graphes

### Exercice 12 : Planification de travaux

Soient  $T = \{t_1, \dots, t_7\}$  un ensemble de 7 travaux, et  $M = \{m_1, \dots, m_7\}$  un ensemble de 7 machines. Chaque travail  $t_i$  utilise 3 machines et prend une journée à être réalisé. Pour ne pas avoir de travail inachevé en fin de journée, on décide que deux travaux  $t_i$  et  $t_j$  ne peuvent être exécutés dans une même journée que s'ils utilisent deux machines différentes. L'utilisation des machines par les travaux est donnée dans la table 5.

Machine / Travail	$t_1$	$t_2$	$t_3$	$t_4$	$t_5$	$t_6$	$t_7$
$m_1$	X	X					
$m_2$		X	X	X			
$m_3$	X		X				
$m_4$		X		X		X	
$m_5$	X		X		X		X
$m_6$					X	X	X
$m_7$				X	X	X	X

TABLE 5 – Utilisation des machines par les travaux

Nous cherchons à planifier les travaux  $T = \{t_1, \dots, t_7\}$  sur le moins de jours possibles.

**Question 12.1 :** Représenter les contraintes du problème à l'aide d'un graphe.

**Question 12.2 :** Proposer une solution pour la réalisation de ces travaux.

### Exercice 13 : Algorithme de coloration

La coloration des graphes est un domaine d'applications à part entière. A titre d'exemples on peut citer les cartes géographiques planaires, la visualisation, les tracés des circuits imprimés sur plusieurs faces, etc. Il s'agit de colorier les sommets d'un graphe avec un nombre minimum de couleurs. Le problème général est NP-complet, d'où l'idée d'utiliser des heuristiques. Plusieurs solutions sont envisagées, en voici deux :

1. traiter les sommets dans l'ordre des numéros apparaissant dans la structure de graphe,
2. traiter les sommets dans l'ordre décroissant de leurs degrés.

Dans les deux cas, on attribue à chaque sommet la plus petite couleur non utilisée par un de ses voisins.

**Question 13.1 :** Déterminer les deux colorations possibles du graphe figure 15 selon l'ordre des sommets et les degrés.

**Question 13.2 :** Écrire un algorithme qui permet de colorier les sommets d'un graphe.

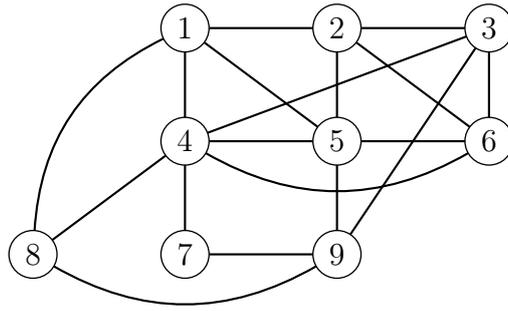


FIGURE 15 – Graphe à colorier

**Correction Exercice 12 :**

**Solution question 12.1 :** *Représenter les contraintes du problème à l'aide d'un graphe.*

Il est possible de représenter les contraintes de ce problème avec un graphe. Pour cela nous choisissons de représenter les travaux à réaliser par les sommets du graphe et la relation “partage une machine” par des arêtes. Ainsi deux sommets mis en relation par une arête ne peuvent pas être exécutés dans la même journée. La figure 16 donne le graphe correspondant aux contraintes données par la table 5.

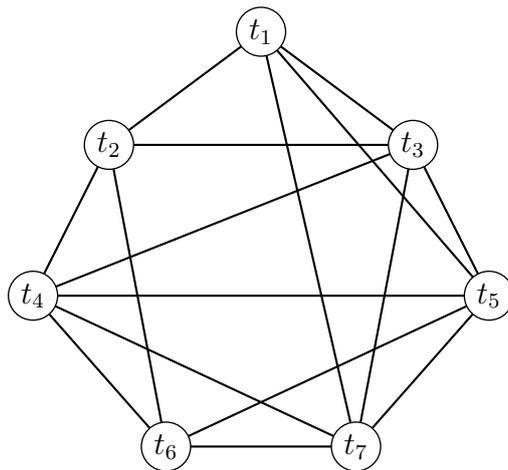


FIGURE 16 – Graphes des contraintes de partage des machines entre travaux

**Solution question 12.2 :** *Proposer une solution pour la réalisation de ces travaux.*

Il est évident qu'en choisissant sept dates différentes pour les sept travaux, ceux-ci pourront être réalisés en disposant de toutes les machines dont ils ont besoin. Notre but est cependant de trouver une planification sur une période la plus courte tout en respectant les contraintes de non-partage. Pour, cela il est possible d'utiliser le graphe 16 en examinant les travaux dans un ordre arbitraire et en leur affectant une date en faisant attention à ne pas choisir une date déjà choisie par un voisin.

Pour être sûr de choisir une date compatible avec ses voisins, il est possible d'utiliser une coloration de graphe puisque, par définition, chaque couleur définit un ensemble de

sommets qui ne sont pas voisins. Nous associons ensuite une date différente à chaque couleur. La figure 17 montre une coloration du graphe 16 avec quatre couleurs. Il est alors possible de proposer une solution avec une date par couleur, par exemple : le premier jour nous exécutons les tâches rouges, donc  $t_1$  et  $t_4$ , le second jour la tâche verte  $t_7$ , le troisième jour les tâches bleues  $t_3$  et  $t_6$  et le quatrième jour les tâches jaunes  $t_2$  et  $t_5$ .

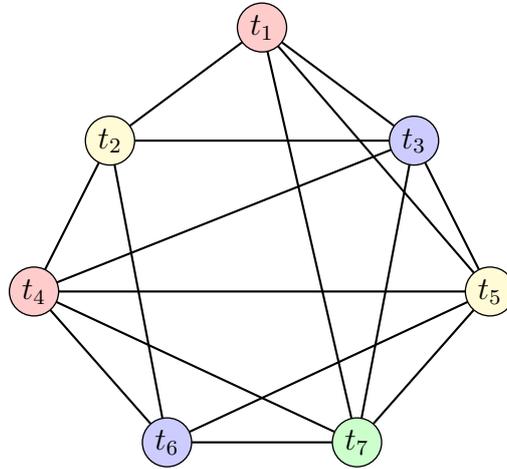


FIGURE 17 – Planification de sept tâches ayant des contraintes de partage de machines entre elles

### Correction Exercice 13 :

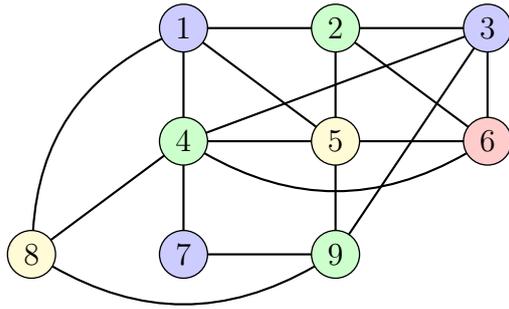
**Solution question 13.1 :** Déterminer les deux colorations possibles du graphe figure 15 selon l'ordre des sommets et les degrés.

La coloration du graphe 15 suivant l'ordre des numéros ou l'ordre décroissant du degré des sommets donne les résultats suivants :

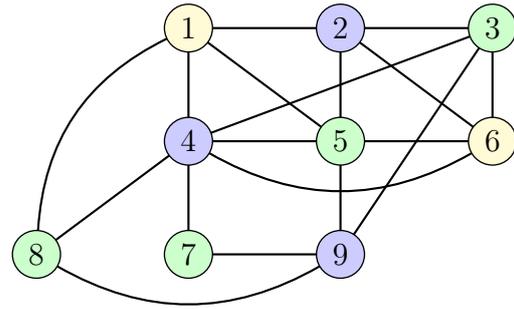
1. 4 couleurs si on examine les sommets dans l'ordre croissant des numéros. Un exemple est donné à la figure 18a.
2. 3 couleurs si on examine les sommets dans l'ordre décroissant des degrés des sommets : 4, 5, 1, 2, 3, 6, 9, 8, 7. Un exemple est donné à la figure 18b.

**Solution question 13.2 :** Écrire un algorithme qui permet de colorier les sommets d'un graphe.

L'algorithme 8 donne une solution possible pour la coloration d'un graphe. Il ne s'agit pas de la coloration avec le plus petit nombre de couleur, mais simplement une coloration possible.



(a) Coloration avec les numéros de sommets



(b) Coloration avec les degrés

FIGURE 18 – Graphe coloré

---

**Algorithme 8 :** Algorithme de la fonction  $coloration(G) : k$

---

**Données :**  $G$  : le graphe à colorier à  $n$  sommets

$ppncl$  : plus petite couleur libre

$tabOccCoul[1..n]$  : tableau des couleurs utilisées par les voisins d'un sommet

**Résultat :**  $couleur(u)$  : numéro de la couleur du sommet  $u$ , initialement à 0

$k$  : nombre de couleurs utilisées

```

1 début
2    $k \leftarrow 0$ 
3   pour chaque sommet  $s$  de  $G$  dans l'ordre décroissant du degré de  $s$  faire
4      $tabOccCoul[1..n] \leftarrow 0$ 
5     pour chaque sommet  $u$  voisin de  $s$  faire
6       if  $couleur(u) \neq 0$  then  $tabOccCoul[couleur(u)] ++$ 
7      $ppncl \leftarrow 1$ 
8     tant que  $tabOccCoul[ppncl] \neq 0$  faire  $ppncl ++$ 
9      $couleur(s) \leftarrow ppncl$ 
10    si  $k < ppncl$  alors  $k \leftarrow ppncl$ 

```

---