

Table des matières

Chapitre I : Introduction et généralités sur les graphes

1.1 Introduction

1.2 Généralités sur les graphes

1.3 Chaînes et cycles d'un graphe

1.4 Graphe eulérien et graphe hamiltonien

1.5 Graphes et matrices

Chapitre II : Parcours des graphes

2.1 Introduction

2.1.1 Définitions

2.2 Spécification d'un algorithme de parcours

2.3 Parcours en largeur

2.3.1 Principe de l'algorithme parcours en largeur

2.3.2 Exemple d'application

2.4 Parcours en profondeur

2.3.1.1 Principe de l'algorithme de parcours en profondeur

2.3.2.2 Exemple d'application

Chapitre III : Le plus court chemin

3.1 Introduction

3.1.1 Définition et exemples

3.2 Algorithme de Dijkstra

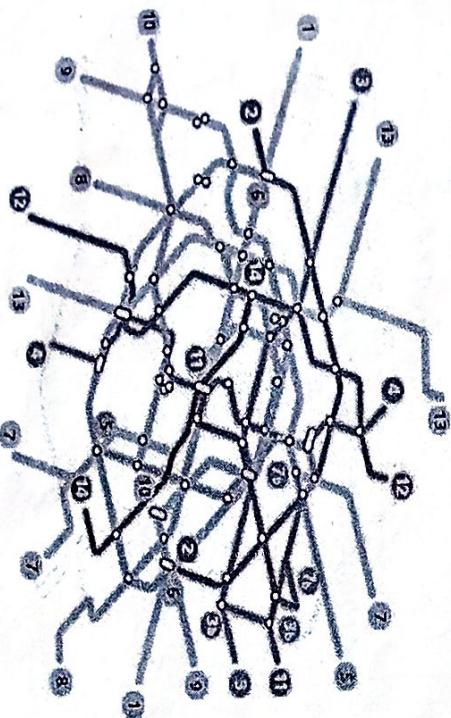
3.2.1 Définition et description de l'algorithme

3.2.2 Exemples

3.3 Algorithme de Bellman Ford

3.3.1 Définition et description de l'algorithme

3.3.2 Exemples



ALGORITHME DES GRAPHES

Hawa Ali Omar

UE

Licence Informatique
Faculté des sciences
Université de Djibouti



Chapitre IV : Coloration des graphes

- 4.1 Introduction
- 4.2 Définition et exemples
- 4.3 Algorithme de glouton
 - 4.3.1 Définition
 - 4.3.2 Principe de l'algorithme et exemples
- 4.4 AlgorithmedSatur (ou Dsat)
 - 4.4.1 Définition
 - 4.4.2 Principe de l'algorithme et des exemples
- 4.5 Comparaison entre les deux algorithmes

Objectif :

Introduction et généralités sur les graphes

Chapitre I

Introduction et généralités sur les graphes

Introduction

En générale la théorie des graphes est une discipline mathématique et informatique qui étudie les graphes. Un graphe est un schéma contenant des points nommés sommets, reliés ou non par des segments appelés arêtes.

Les algorithmes élaborés pour résoudre des problèmes concernant les objets de cette théorie ont de nombreuses applications dans tous les domaines liés à la notion de réseau (réseau social, réseau informatique,

télécommunication, ferroviaires, etc...).

On accorde généralement à Léonard Euler, l'origine de la théorie des graphes. Ce mathématicien et physicien suisse a été le premier en 1736 à résoudre un problème (celui de ponts de Königsberg) en démontrant une propriété des graphes.

Aujourd'hui l'utilisation des graphes est devenue monnaie courante, car ceux-ci offrent une représentation imagée et colorée de situation très variées auxquelles nous faisons face au quotidien.

Dans ce chapitre nous verrons la généralité sur les graphes, le graphe euclidien et hamiltonien. Nous évoquerons ainsi graphes et matrices.

1.2 Généralités sur les graphes

1.2.1. Définition d'un graphe

Un graphe $G = (V; E)$ est un couple d'ensembles finis, dont V est l'ensemble de sommets de G (représentant des objets), et E est l'ensemble d'arêtes de G (représentant des liens/relations entre des objets). Une arête relie deux sommets (pas nécessairement distincts). Si l'arête e relie les sommets u et v , on écrit $e = uv$, on dit que u et v sont voisins ou adjacents.



Pour un graphe G , on note

- $V(G)$ l'ensemble des sommets de G
- $n = |V(G)|$ le nombre de sommets de G – l'ordre de G
- $E(G)$ l'ensemble des arêtes de G
- $m = |E(G)|$ le nombre d'arêtes de G – la taille de G

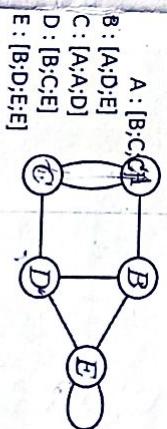
Une arête reliant un sommet à lui-même est une boucle. Des arêtes reliant la même paire de sommets sont des arêtes parallèles (des arêtes multiples). Un graphe est dit simple s'il n'a ni boucles ni arêtes multiples.

Y a-t-il un graphe simple parmi les graphes ci-dessus ?

1.2.2 Représentation d'un graphe

Pour représenter un graphe, il faut retenir le nombre d'arête et le nombre de sommets. Il faudra savoir :

- Les listes d'adjacence : pour chaque sommet du graphe la liste de ses voisins ;



➤ La matrice d'adjacence : pour chaque paire de sommets il est indiqué s'ils sont voisins ou pas ;

	v_1	v_2	v_3	v_4	v_5
v_1	0	1	1	0	1
v_2	1	0	1	0	1
v_3	1	1	0	1	0
v_4	0	0	1	0	1
v_5	1	1	0	1	0

- La matrice d'incidence : pour chaque sommet et pour chaque arête il est indiqué s'ils sont incidents ou pas.

	e_1	e_2	e_3	e_4	e_5	e_6	e_7
v_1	1	1	1	0	0	0	0
v_2	1	0	0	1	1	0	0
v_3	0	1	0	1	0	1	0
v_4	0	0	0	0	0	1	1
v_5	0	0	1	0	1	0	1

Graph 2

1.2.3 Degrés d'un graphe

Le degré $\deg(v)$ d'un sommet v est la longueur de la liste d'adjacence de v . Dans un graphe simple, $\deg(v)$ est égale au nombre d'arêtes qui lui sont incidentes.

Quel est le degré de v_2 dans le graphe 2 ci-dessus ? $\deg(v_2) = 3$.

Théorème (lemme des poignées de main)

Soit G un graphe. La somme des degrés de sommets de G est égale au double du nombre d'arêtes de G :

$$\sum_{v \in V(G)} \deg(v) = 2 \cdot |E(G)|.$$

Exemple : par double comptage

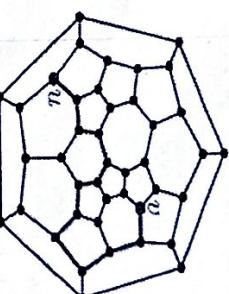
Posons sur chaque sommet du graphe autant de jetons que son degré. Il y a $\sum_{v \in V(G)} \deg(v)$ jetons au total. Tout sommet passe un jeton à toute arête incidente. Chaque arête reçoit deux jetons. Il y a donc $2m$ jetons, d'où l'égalité.

- $\sum_{v \in V(G)} \deg(v)$ est la somme des sommes de lignes de la matrice d'incidence de G ,
- $2m$ est la somme des sommes de colonnes de la dite matrice, d'où l'égalité.

1.3 Chaîne et cycle d'un graphe

1.3.1-Chaîne et connexité

Une chaîne entre deux sommets u et v d'un graphe G est une séquence d'arêtes consécutives par exemple $(u_0u_1; u_1u_2; \dots; u_{k-1}u_k)$, telle que $u = u_0$ et $v = u_k$. La longueur d'une chaîne est le nombre d'arêtes qui la composent (ici k).



On dit qu'un sommet v est accessible à partir du sommet u si il existe une chaîne reliant u et v . Un graphe est dit **connexe** si ses sommets sont tous accessibles entre eux, pour toute paire de sommets u et v il existe une chaîne reliant u et v .

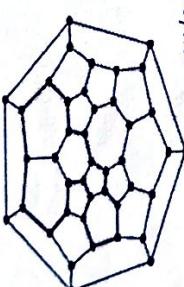
Une chaîne est **simple** si elle passe par toute arête au maximum une fois. Une chaîne est **élémentaire** si elle passe par toute arête et par tout sommet au maximum une fois.

Lemme

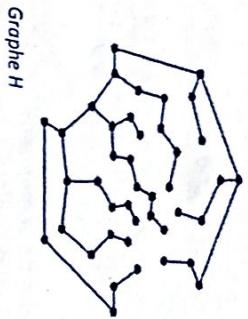
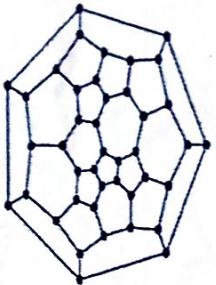
S'il existe une chaîne reliant u et v dans un graphe G , alors il existe aussi une chaîne élémentaire les reliant.

1.3.2-Cycle et arbres

Une séquence d'arêtes distinctes consécutives reliant un sommet à lui-même est appelée un **cycle**; la longueur du cycle est le nombre d'arêtes composant le cycle.



Un arbre est un graphe connexe sans cycle. Le graphe H est un arbre courant du graphe G si H est un arbre que l'on peut obtenir en supprimant des arêtes de G .



Graph G

Arbre

Graph H

Arbre

1.4 Graphes et matrices

1.4.1 Graphes orientés

Un graphe orienté $G = (V; E)$ est un couple d'ensembles finis, dont :

- V est l'ensemble de sommets de G , et
- E est l'ensemble d'arcs de G , où tout arc relie un sommet à un (autre) sommet.

Si l'arc e relie u à v , on écrit $e = uv$, on dit que

- ✓ uv est un arc sortant de u ,
- ✓ v est un voisin sortant / successeur de u ,
- ✓ uv est un arc entrant à/de v ,
- ✓ u est un voisin entrant / prédecesseur de v .
- ✓ Les listes de successeurs : pour chaque sommet du graphe la liste de ses successeurs ;
- ✓ La matrice d'adjacence : pour chaque paire de sommets vi et vj il est indiqué s'il existe un arc de vi à vj ;
- ✓ La matrice d'incidence : pour chaque sommet vi et pour chaque arc ej il est indiqué si l'arc ej est un arc sortant (-1) ou entrant (+1) du sommet vi .

Exemple :

Un graphe orienté (digraph) est donnée par un couple

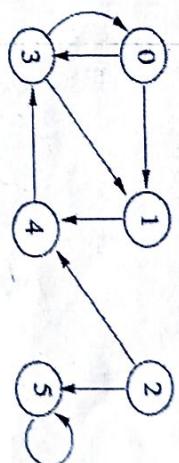
✓ $G = (V; E)$, où

✓ V est un ensemble des sommets.

✓ E est un ensemble des arcs entrants et sortants.

✓ $V = \{0; 1; \dots; 6\}$.

✓ $E = \{(0,1); (3,4); (5,1); (6,3); (6,4); (6,6)\}$



La matrice d'adjacence d'un graphe $G = (S; A)$. La matrice M telle que

✓ $M[si][sj] = 1$ si $(si; sj) \in A$, et $M[si][sj] = 0$ sinon

M	0	1	2	3	4	5
0	0	1	0	1	0	0
1	0	0	0	0	1	0
2	0	0	0	0	1	1
3	1	1	0	0	0	0
4	0	0	0	1	0	0
5	0	0	0	0	0	1

1.4.2 Graphes planaires

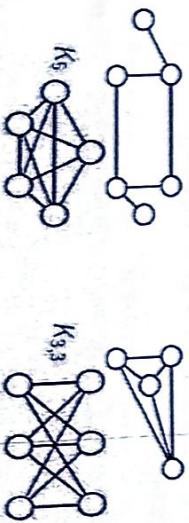
Un graphe est dit planaire s'il peut se représenter sur un plan sans qu'aucune arête n'en croise une autre.

- ✓ Le degré sortant $\deg^+(v)$ d'un sommet v est la longueur de la liste de successeurs de v . Le degré sortant d'un sommet v est égal au nombre d'arcs sortants de v .
- ✓ Le degré entrant $\deg^-(v)$ d'un sommet v est la longueur de la liste de prédécesseurs de v . Le degré entrant d'un sommet v est égal au nombre d'arcs entrants de v .

Exemple :

➤ Graphe planaire :

Graphe non-planaire :

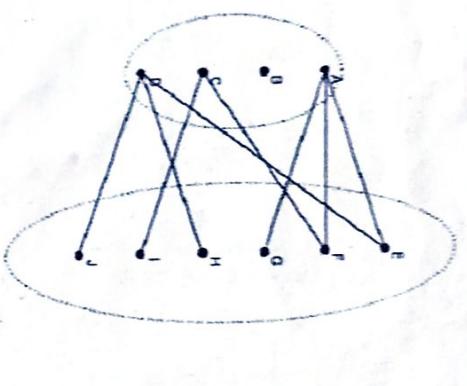


Théorème [Kuratowski-Wagner] : Un graphe fini est planaire si et seulement si il ne contient pas de sous-graphe qui soit une expansion de K_5 ou de $K_{3,3}$.

Une expansion consiste à ajouter un ou plusieurs sommets sur une ou plusieurs arêtes (exemple:  devient ).

1.4.3 Graphes bipartie

Un graphe est dit biparti si on peut partager son ensemble de sommets en deux parties A et B tels qu'il n'y ait aucune arête entre éléments de A et aucune arête entre éléments de B.



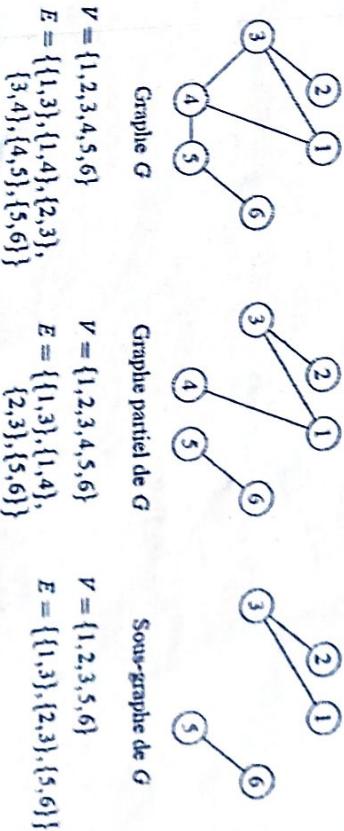
Théorème : Un graphe est biparti si et seulement s'il ne contient pas de cycles de longueur impaire.

Rappelons que la longueur d'un cycle est égale au nombre d'arêtes qu'il contient. En particulier, d'après le théorème précédent, les arbres sont des graphes bipartis.

1.4.4. Graphe partiel et sous-graphe

Soit $G = (V, E)$ un graphe. Le graphe $G' = (V, E')$ est un graphe partiel de G , si E' est inclus dans E . Autrement dit, on obtient G' en enlevant une ou plusieurs arêtes au graphe G .

Pour un sous-ensemble de sommets A inclus dans V , le sous-graphe de G induit par A est le graphe $G = (A, E(A))$ dont l'ensemble des sommets est A et l'ensemble des arêtes $E(A)$ est formé de toutes les arêtes de G ayant leurs deux extrémités dans A . Autrement dit, on obtient G' en enlevant un ou plusieurs sommets au graphe G , ainsi que toutes les arêtes incidentes à ces sommets.



Un graphe partiel d'un sous-graphe est un sous-graphe partiel de G . On appelle clique un sous-graphe complet de G . Dans le graphe G ci-dessus, le sous-graphe $K = (V, E)$, avec $V = \{1, 3, 4\}$ et $E = \{\{1, 3\}, \{1, 4\}, \{3, 4\}\}$ est une clique.

Autrement dit, les graphes bipartis sont ceux que l'on peut colorer en utilisant au plus deux couleurs. Le théorème suivant, dû à Königsberg en 1916, caractérise les graphes bipartis :

Le degré d'un graphe est le degré maximum de tous ses sommets. Dans l'exemple ci-dessous, le degré du graphe est 4, à cause du sommet v3.

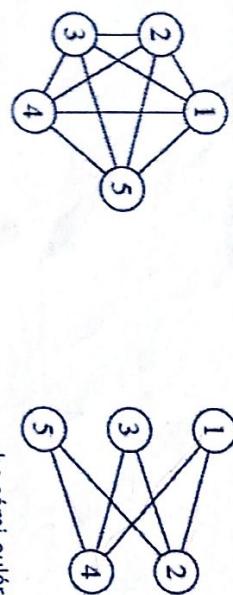
Un graphe dont tous les sommets ont le même degré est dit régulier. Si le degré commun est k , alors on dit que le graphe est k -régulier.



1.5 Graphes eulériens et Graphes Hamiltoniens

On appelle **cycle eulérien** d'un graphe G un cycle passant une et une seule fois par chacune des arêtes de G . Un graphe est dit eulérien s'il possède un cycle eulérien.

On appelle chaîne eulérienne d'un graphe G une chaîne passant une et une seule fois par chacune des arêtes de G . Un graphe ne possédant que des chaînes eulériennes est semi-eulérien. Plus simplement, on peut dire qu'un graphe est eulérien (ou semi-eulérien) s'il est possible de dessiner le graphe sans lever le crayon et sans passer deux fois sur la même arête.

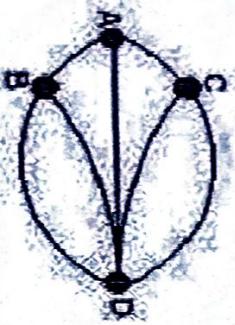


Graph eulérien

Graph sémi-eulérien

La caractérisation des graphes eulériens est très simple. Un graphe est eulérien si et seulement si tous ses sommets ont des degrés pairs. Sur l'exemple des ponts de Königsberg, on constate tout désulte que ce n'est pas le cas, et donc que Leonhard Euler ne pouvait pas effectuer cette promenade.

La question est maintenant de trouver une méthode efficace (en temps polynomial en fonction de n et m) pour mettre en évidence un cycle eulérien.



Cet algorithme repose sur une simple constatation. Dans un graphe eulérien $G = (V,E)$, supposons que l'on ait trouvé un cycle C . Le graphe $G' = G \setminus C$, le graphe G auquel nous enlevons toutes les arêtes contenues dans C , est encore eulérien. En effet, comme C est un cycle, chaque sommet du cycle à un degré pair, par définition, les sommets de G' auront donc tous un degré pair.

L'algorithme est alors très simple, il consiste à chercher un cycle dans G , le retirer de G , de chercher à nouveau un cycle, et ainsi de suite. Pour finir, il suffit de combiner tous les cycles trouvés en un seul cycle pour obtenir un cycle eulérien.

On appelle **cycle hamiltonien** d'un graphe G est un cycle passant une et une seule fois par chacun des sommets de G . Un graphe est dit **hamiltonien** s'il possède un cycle hamiltonien.

On appelle **chaîne hamiltonienne** d'un graphe G une chaîne passant une et une seule fois par chacun des sommets de G . Un graphe ne possédant que des chaînes hamiltoniennes est **semi-hamiltonien**. Contrairement aux graphes eulériens, il n'existe pas de caractérisation simple des graphes (semi-)hamiltoniens. On peut énoncer quelques propriétés et conditions suffisantes :

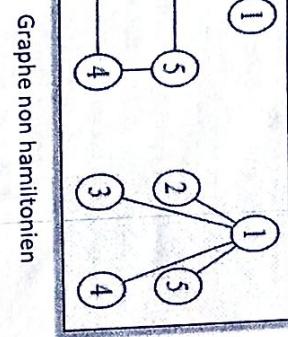
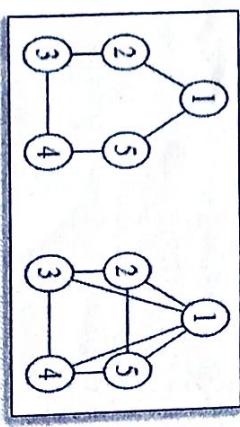
Définition : Un graphe est hamiltonien s'il existe un cycle élémentaire passant par tous les sommets.

L'autre question naturelle que l'on peut se poser sur les cycles dans les graphes est la suivante : "Dans un graphe $G = (V,E)$, existe-t-il un cycle élémentaire qui passe une fois et une seule par tous les sommets ?" Si la réponse est oui, on parle alors d'un **cycle hamiltonien** (*Hamiltonian cycle*), et d'un **graph hamiltonien** (*Hamiltonian graph*) (voir définition ci-dessus).

Malheureusement, caractériser les graphes hamiltoniens n'est pas aussi simple que pour les graphes eulérien, et nous n'avons aujourd'hui qu'un ensemble de conditions nécessaires ou suffisantes, mais rien qui permet de dire efficacement qu'un graphe est hamiltonien.

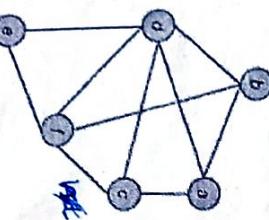
Si on prend par exemple un graphhe complet, il est facile de voir qu'il existe un cycle hamiltonien. Les graphes complets sont donc hamiltoniens. Si l'on prend des graphes qui ont des cycles, la réponse est tout aussi évidente.

Si l'on prend un arbre, la réponse est encore évidente, un arbre n'est pas hamiltonien puisqu'il ne comporte pas de cycle. En revanche, cela se complique déjà si l'on prend un graphhe biparti (problème NPcomplet).

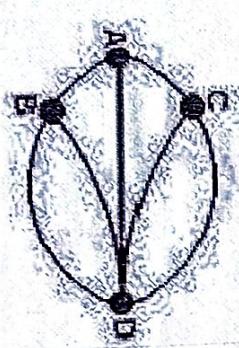


Graphhe hamiltonien

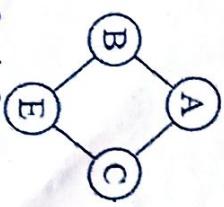
Exercice d'application :



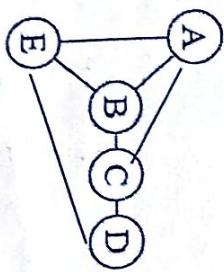
Graphhe 1



Graphhe 2



Graphhe 3



Graphhe 4

- 1) Pour le
- 2) Les graphes ci-dessus y-a-t-il un cycle eulérien ou une chaîne eulérienne ?
- 3) Les graphes ci-dessus y-a-t-il un cycle hamiltonien ou une chaîne hamiltonienne ?
- 4) Si possible transformez les graphes ci-dessus en graphes eulériens ?

TD -Chapitre 1

PARTIE 1 : NOTIONS DE BASE

EXERCICE N°1:

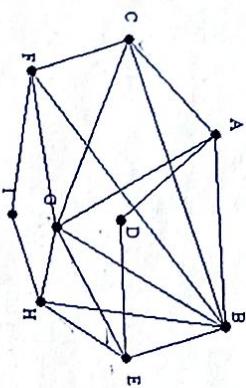
Construire un graphe orienté dont les sommets sont les entiers compris entre 1 et 12 et dont les arêtes représentent la relation « être diviseur de ».

EXERCICE N°2:

Une chèvre, un chou et un loup se trouvent sur la rive d'un fleuve ; un passeur souhaite les transporter sur l'autre rive mais, sa barque étant trop petite, il ne peut transporter qu'un seul d'entre eux à la fois. Comment doit-il procéder afin de ne jamais laisser ensemble et sans surveillance le loup et la chèvre, ainsi que la chèvre et le chou ?

EXERCICE N°3:

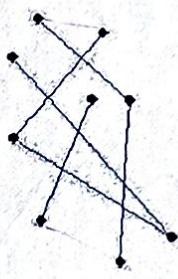
Déterminer le degré de chacun des sommets du graphe suivant :



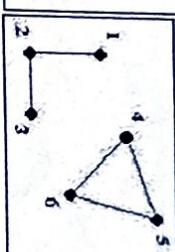
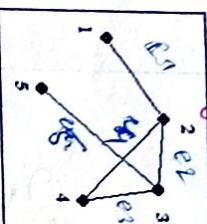
EXERCICE N°6:

On considère la matrice

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



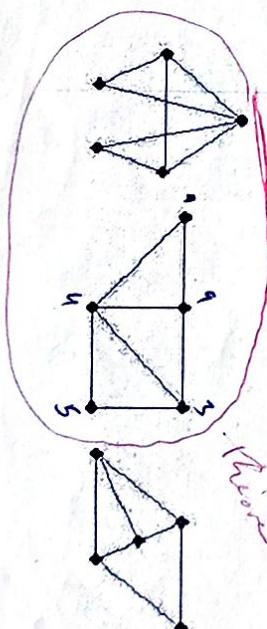
Transformer ce graphe en lui rajoutant un nombre minimal d'arêtes pour qu'il soit connexe.



Graph 1 Graph 2

Les graphes ci-dessous peuvent-ils être associés à A ?

EXERCICE N°4:
Ecrivez la matrice associée à chaque graphe :



EXERCICE N°7:

Soit le graphe simple, non orienté $G = (V, E)$, donné par sa matrice d'incidence ci-dessous.

S0	111000000	(1)
S1	000100000	(2)
S2	000011100	(3)
S3	000001010	(4)
S4	100001001	(5)
S5	010000001	(6)
S6	001000110	(7)
S7	000100000	(8)

Dessiner le graphe ? Quel est le degré minimal et le degré maximal ? Est-il connexe ?

Si chaque joueur ne joue qu'un match par jour, combien de jours faudra-t-il pour terminer le tournoi ?

Aidez-vous du graphe pour proposer un calendrier des matches.

EXERCICE N°8:

Trois professeurs P_1, P_2, P_3 devront donner lundi prochain un certain nombre d'heures de cours à trois classes C_1, C_2, C_3 :

P_1 doit donner 2 heures de cours à C_1 et 1 heure à C_2 ;

P_2 doit donner 1 heure de cours à C_1 , 1 heure à C_2 et 1 heure à C_3 ;

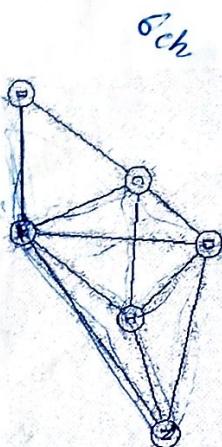
P_3 doit donner 1 heure de cours à C_1 , 1 heure à C_2 et 2 heures à C_3 .

Comment représenter cette situation par un graphe ? Quel type de graphe obtenez-vous ?

EXERCICE N°9:

Un groupe d'amis organise une randonnée dans les Alpes. On a représenté par le graphe ci-dessous les sommets B, C, D, F, T, N par lesquels ils peuvent choisir de passer.

Une arête entre deux sommets coincide avec l'existence d'un chemin entre les deux sommets.



1) a) Recopier et compléter le tableau suivant :

Sommets	B	C	D	F	N	T
Degré des sommets du graphe	2					

b) Justifier que le graphe est connexe.

2) Le groupe souhaite passer par les six sommets en passant une fois et une seule par chaque chemin.

Démontrer que leur souhait est réalisable. Donner un exemple de trajet possible.

EXERCICE N°10:

Un tournoi d'échecs oppose 6 personnes. Chaque joueur doit affronter tous les autres. Construisez un graphe représentant toutes les parties possibles.

Quel type de graphe obtenez-vous ?

Si chaque joueur ne joue qu'un match par jour, combien de jours faudra-t-il pour terminer le tournoi ?

Exercice 11:

Sept ponts enjambent la Pregele, reliant quatre quartiers de la ville. Les habitants se demandent s'il existe un trajet leur permettant d'emprunter une seule fois tous les ponts.

Euler modélise le problème et ouvre ainsi une nouvelle théorie.

Le problème posé induit deux questions :



Existe-t-il un trajet partant d'un point donné, passant par toutes les arêtes une et une seule fois (chaîne eulérienne) ?

Ou bien existe-t-il une chaîne eulérienne revenant au point de départ (cycle eulérien) ?

Le théorème d'Euler énonce qu'un graphe non orienté admet une chaîne eulérienne si et seulement si il est connexe et admet zéros ou deux sommets impairs. Si tous les sommets sont pairs, il s'agit de cycle eulérien. À Königsberg, rebaptisée depuis Kaliningrad, il y a deux nouveaux ponts, l'un entre B et C et l'autre entre B et A.

Y a-t-il une chaîne eulérienne ?
Où faudrait-il construire un autre pont pour obtenir un cycle eulérien ?



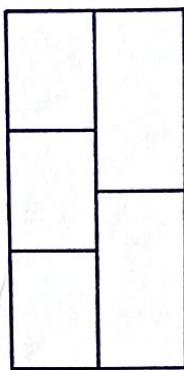
Exercice 12

Soit G un graphe non eulérien. Est-il toujours possible de rendre G eulérien en lui rajoutant un sommet et quelques arêtes ?



Exercice 13

Est-il possible de tracer une courbe, sans lever le crayon, qui coupe chacun des 16 segments de la figure suivante exactement une fois ?



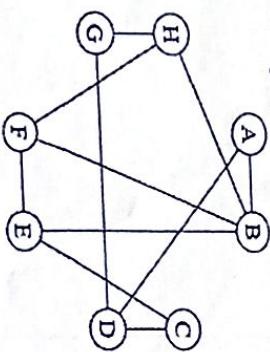
Exercice 14 :

Dessinez un graphe d'ordre au moins 5 qui est...

- 1) hamiltonien et eulérien
- 2) hamiltonien et non eulérien
- 3) non hamiltonien et eulérien
- 4) non hamiltonien et non eulérien.

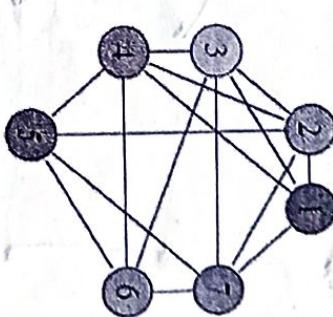
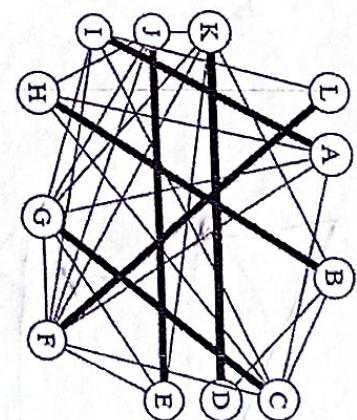
Exercice 15 :

Huit personnes se retrouvent pour un repas de mariage. Le graphe ci-dessous précise les incompatibilités d'humeur entre ces personnes (une arête reliant deux personnes indique qu'elles ne se supportent pas).



(la table est ronde) en évitant de placer côté à côté deux personnes incompatibles.

Exercice 16 :



- 1) Sont-ils des graphes réguliers ?
- 2) Si non, compléter les graphes pour rendre régulier ?

Algorithme de parcours en profondeur et en largeur

Introduction

Qu'est-ce qu'un parcours de graphe (orienté ou non) ?

La visite de tous les sommets sont accessibles depuis un sommet de départ donné.

Comment parcourir un graphe ?

Marquage des sommets par des couleurs :

- ✓ Blanc = Sommet pas encore visité
- ✓ Gris = Sommet en cours d'exploitation
- ✓ Noir = Sommet que l'on a fini d'exploiter

Au début, le sommet de départ est gris et tous les autres sont blancs. À chaque étape, un sommet gris est sélectionné. Si tous ses voisins sont déjà gris ou noirs, alors il est colorié en noir. Sinon, il colorie un (ou plusieurs) de ses voisins blancs en gris ; jusqu'à ce que tous les sommets soient noirs ou blancs

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. La notion d'exploration ou 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.

Définition :

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

Définition :

Parcours : c'est un parcours de racine r est une suite L de sommets telle que :

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

Deux types d'exploration :

- parcours en largeur, alors on utilise une file (FIFO);
- parcours en profondeur, alors on utilise une pile (LIFO).

Spécification d'un algorithme de parcours

1 Fonction Parcours($g; s_0$)

Entrée : Un graphe g et un sommet s_0 de g

Post-condition

: Retourne l'arborescence du parcours de g à partir de s_0

2 Arborescence associée à un parcours :

- si s_i est le père de s_j si s_i qui a colorié s_j en gris
- s_i est racine si $s_i = s_0$ ou si pas de chemin de s_0 jusque s_i
- Méémorisation dans un tableau π tel que $\pi[s_i] = null$ si s_i est racine, $\text{et}\pi[s_i] = \text{père de } s_i$ sinon

nom de tab

Exemple :

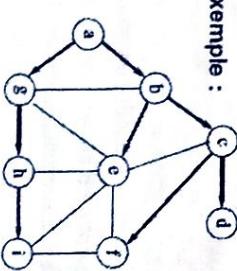


Tableau π correspondant :

-	a	b	c	b	c	a	g	h	i
a	b	c	d	e	f	g	h	i	

Parcours en largeur : Principe de l'algorithme

C'est comme si vous parcouriez toutes les pages d'un site web. Les pages sont les sommets d'un graphe et un lien entre deux pages est une arête entre ces deux sommets :

1. Dans le parcours en largeur, on utilise une file. Et on enfile le sommet de départ (on visite la page index du site).

2. On visite les voisins de la tête de file (pages ciblées par la page de tête de file). On les enfile (en les numérotant au fur et à mesure de leur découverte) si'ils ne sont pas déjà présents dans la file, ni déjà passées dans la file.

3. On défile (c'est à dire : on supprime la tête de file).
4. On recommence au point 2 (tant que c'est possible, c'est à dire tant que la file n'est pas vide).

En d'autres termes, on traite toujours en priorité les liens des pages le plus tôt découvertes.

Algorithme de parcours en largeur :

Fonction BFS($g; s_0$)

2. Soit f une file (FIFO) initialisée à vide

3. pour chaque sommet S_i de g faire

4. $\pi[s_i] = null$

5. Colorier s_i en blanc

6. Ajouter s_0 dans f et colorier s_0 en gris

7. tant que f n'est pas vide faire

8. Soit s_k le sommet le plus ancien dans f

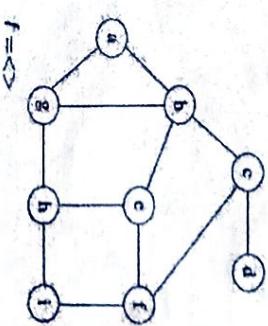
9. tant que $\exists S_j \text{ esucc}(s_k)$ tq s_j soit blanc faire

10. Ajouter s_j dans f et colorier s_j en gris

11. $\pi[s_k] = s_k$

12. Enlever s_k de f et colorier s_k en noir

13. retourner



Exemple avec l'algorithme : ici on met le sommet a dans la file f

1 Fonction $BFS(g, s_0)$

Soit f une file (FIFO) initialisée à vide
pour chaque sommet s_i de g faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Ajouter s_0 dans f et colorier s_0 en gris

tant que f n'est pas vide faire

Soit s_k le sommet le plus ancien dans f
tant que $\exists s_j \in succ(s_k)$ tq s_j soit blanc faire

Ajouter s_j dans f et colorier s_j en gris

$\pi[s_j] \leftarrow s_k$

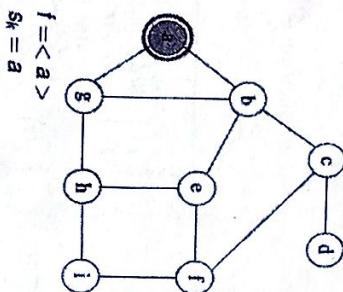
Enlever s_k de f et colorier s_k en noir

retourne π

On affectera qui devient le sommet le plus ancien à s_k

```

1 Fonction  $BFS(g, s_0)$ 
2 Soit  $f$  une file (FIFO) initialisée à vide
3 pour chaque sommet  $s_i$  de  $g$  faire
4    $\pi[s_i] \leftarrow null$ 
5   Colorier  $s_i$  en blanc
6   Ajouter  $s_0$  dans  $f$  et colorier  $s_0$  en gris
7   tant que  $f$  n'est pas vide faire
8     Soit  $s_k$  le sommet le plus ancien dans  $f$ 
9     tant que  $\exists s_j \in succ(s_k)$  tq  $s_j$  soit blanc faire
10       Ajouter  $s_j$  dans  $f$  et colorier  $s_j$  en gris
11        $\pi[s_j] \leftarrow s_k$ 
12     Enlever  $s_k$  de  $f$  et colorier  $s_k$  en noir
13
retourne  $\pi$ 
```



Ici on chercher le successeur de a et le premier candidat est b donc, on met b dans f et en colorie en gris

1 Fonction $BFS(g, s_0)$

Soit f une file (FIFO) initialisée à vide
pour chaque sommet s_i de g faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Ajouter s_0 dans f et colorier s_0 en gris

tant que f n'est pas vide faire

Soit s_k le sommet le plus ancien dans f
tant que $\exists s_j \in succ(s_k)$ tq s_j soit blanc faire

Ajouter s_j dans f et colorier s_j en gris

$\pi[s_j] \leftarrow s_k$

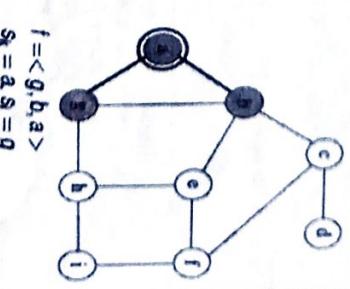
Enlever s_k de f et colorier s_k en noir

retourne π

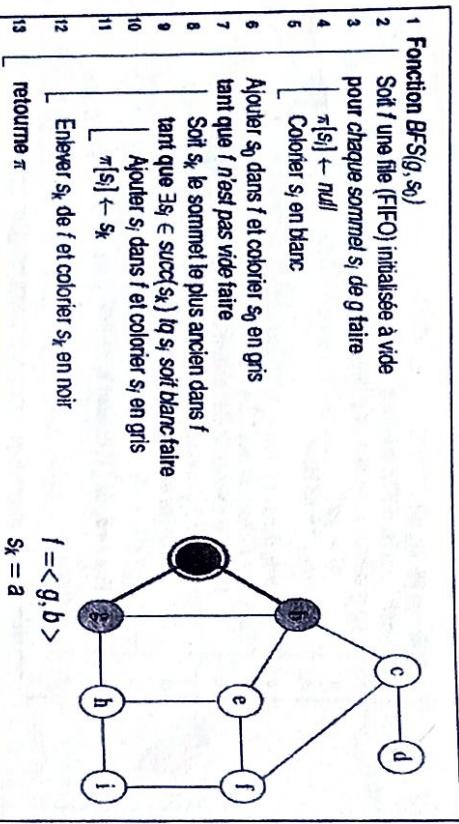
On affectera qui devient le sommet le plus ancien à s_k

```

1 Fonction  $BFS(g, s_0)$ 
2 Soit  $f$  une file (FIFO) initialisée à vide
3 pour chaque sommet  $s_i$  de  $g$  faire
4    $\pi[s_i] \leftarrow null$ 
5   Colorier  $s_i$  en blanc
6   Ajouter  $s_0$  dans  $f$  et colorier  $s_0$  en gris
7   tant que  $f$  n'est pas vide faire
8     Soit  $s_k$  le sommet le plus ancien dans  $f$ 
9     tant que  $\exists s_j \in succ(s_k)$  tq  $s_j$  soit blanc faire
10       Ajouter  $s_j$  dans  $f$  et colorier  $s_j$  en gris
11        $\pi[s_j] \leftarrow s_k$ 
12     Enlever  $s_k$  de  $f$  et colorier  $s_k$  en noir
13
retourne  $\pi$ 
```

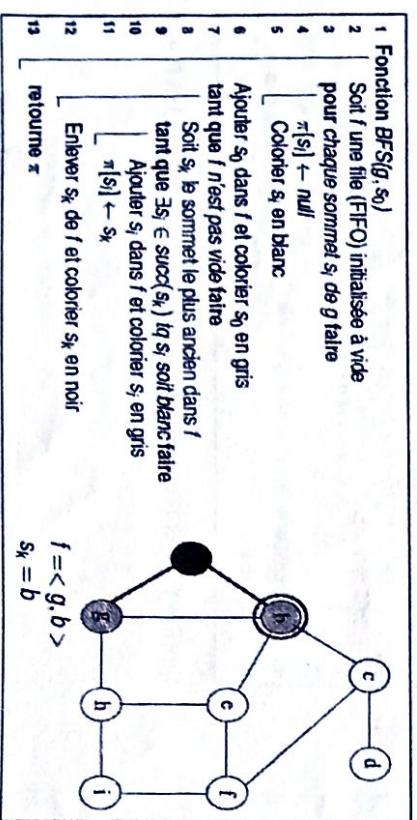


Une fois trouver tous le successeur de a , on enlève a de f et colorier en noir.

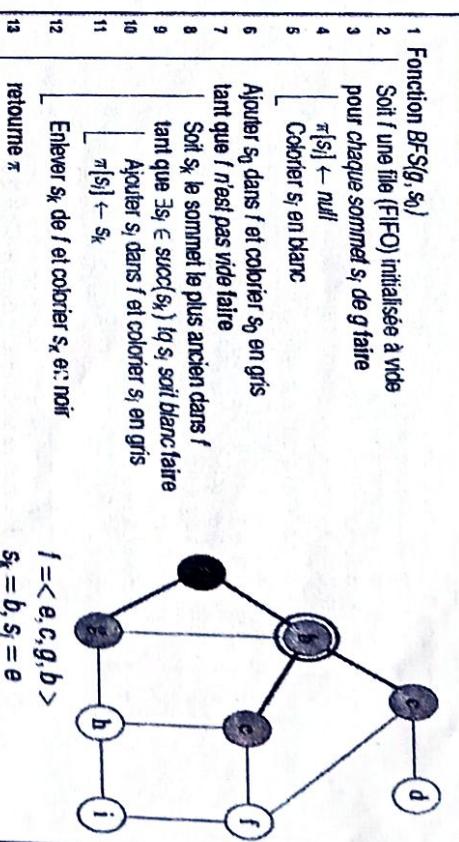


Une fois qu'on a colorié les successeurs du sommet initial, on répète la procédure pour le successeur n°1, afin de colorier ses successeurs.

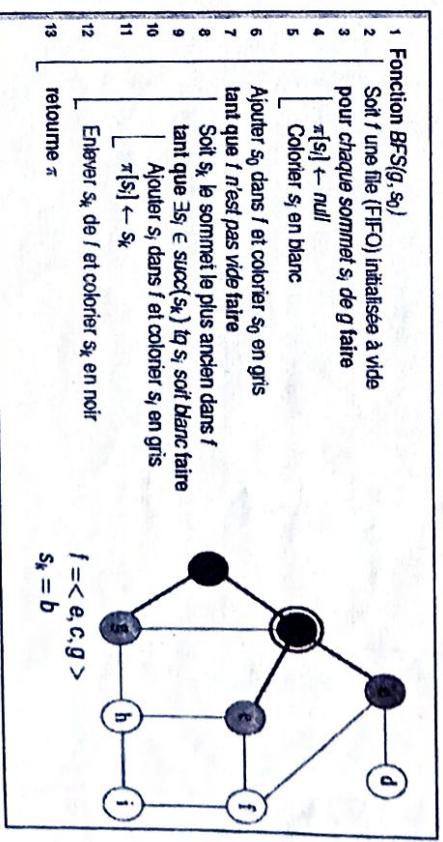
Ici notre $s_k = b$



Les successeurs de b sont c et e



On répète la même procédure jusqu'au dernier sommet.



Fonction $BFS(g, s_0)$

Soit une file (FIFO) initialisée à vide
pour chaque sommet s_i de g faire

$\pi[s_i] \leftarrow null$

Colorier s_0 en blanc

Ajouter s_0 dans f et colorier s_0 en gris

Tant que f n'est pas vide faire

Soit s_k le sommet le plus ancien dans f

Tant que $\exists s_j \in succ(s_k)$ tel que s_j soit blanc faire

Ajouter s_j dans f et colorier s_j en gris

Ajouter s_k de f et colorier s_k en noir

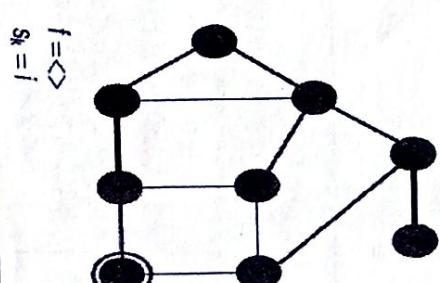
$\pi[s_k] \leftarrow s_k$

Enlever s_k de f et colorier s_k en noir

retourne π

```

1 Fonction  $BFS(g, s_0)$ 
2 Soit une file (FIFO) initialisée à vide
3 pour chaque sommet  $s_i$  de  $g$  faire
4    $\pi[s_i] \leftarrow null$ 
5   Colorier  $s_i$  en blanc
6   Ajouter  $s_0$  dans  $f$  et colorier  $s_0$  en gris
7   Tant que  $f$  n'est pas vide faire
8     Soit  $s_k$  le sommet le plus ancien dans  $f$ 
9     Tant que  $\exists s_j \in succ(s_k)$  tel que  $s_j$  soit blanc faire
10       Ajouter  $s_j$  dans  $f$  et colorier  $s_j$  en gris
11        $\pi[s_k] \leftarrow s_k$ 
12     Enlever  $s_k$  de  $f$  et colorier  $s_k$  en noir
13   
```



Parcours en profondeur : Principe de l'algorithme

Vous devez parcourir toutes les pages d'un site web. Les pages sont les sommets.

1. Dans le parcours en profondeur, on utilise une PILE. On empile le sommet de départ (ex : on visite la page index du site)
2. Si le sommet de la pile présente des voisins qui ne sont pas dans la pile, ni déjà passés dans la pile :

✓ Alors on sélectionne l'un de ces voisins et on empile(en le marquant de son numéro de découvert)

✓ Sinon on dépile (c'est-à-dire on supprime l'élément du sommet de la pile)

3. On recommence au point 2 (tant que la pile n'est pas vide)

En d'autre terme on traite toujours en priorité les liens des pages les plus tard découvertes.

Algorithme de parcours en profondeur

Fonction $DFS(g; s_0)$

2. Soit p une pile (LIFO) initialisée à vide

3. pour tout sommet s_i de S faire

4. $\pi[s_i] \leftarrow null$

5. Colorier s_i en blanc

6. Empiler s_0 dans p et colorier s_0 en gris

7. tant que p n'est pas vide faire

8. Soit s_i le dernier sommet entré dans p

9. si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

10. Empiler s_j dans p et colorier s_j en gris

11. $\pi[s_j] \leftarrow s_i$

12. sinon

13. Dépiler s_i de p et colorier s_i en noir

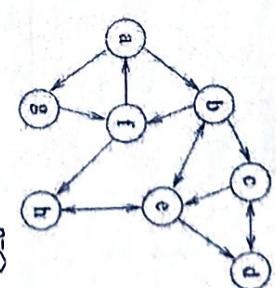
14. retourner

11. $\pi[s_j] \leftarrow s_i$

12. sinon

13. Dépiler s_i de p et colorier s_i en noir

14. retourner



1 Fonction $DFS(g, s_0)$

2 Soit p une pile (LIFO) initialisée à vide
3 pour tout sommet $s_i \in S$ faire
4 $\pi[s_i] \leftarrow null$
5 Colorier s_i en blanc

6 Empiler s_0 dans p et colorier s_0 en gris
7 tant que p n'est pas vide faire
8 Soit s_i le dernier sommet entré dans p
9 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors
10 Empiler s_j dans p et colorier s_j en gris
11 $\pi[s_j] \leftarrow s_i$
12 sinon
13 Dépiler s_i de p et colorier s_i en noir

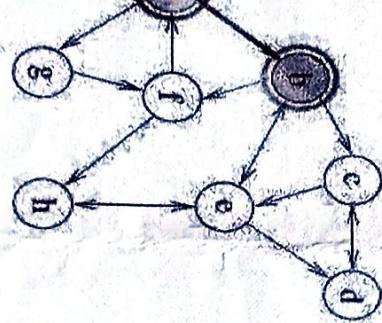
14 retourne π

1 Fonction $DFS(g, s_0)$

2 Soit p une pile (LIFO) initialisée à vide
3 pour tout sommet $s_i \in S$ faire
4 $\pi[s_i] \leftarrow null$
5 Colorier s_i en blanc

6 Empiler s_0 dans p et colorier s_0 en gris
7 tant que p n'est pas vide faire
8 Soit s_i le dernier sommet entré dans p
9 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors
10 Empiler s_j dans p et colorier s_j en gris
11 $\pi[s_j] \leftarrow s_i$
12 sinon
13 Dépiler s_i de p et colorier s_i en noir

14 retourne π



1 Fonction $DFS(g, s_0)$

2 Soit p une pile (LIFO) initialisée à vide
3 pour tout sommet $s_i \in S$ faire
4 $\pi[s_i] \leftarrow null$
5 Colorier s_i en blanc

6 Empiler s_0 dans p et colorier s_0 en gris
7 tant que p n'est pas vide faire
8 Soit s_i le dernier sommet entré dans p
9 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors
10 Empiler s_j dans p et colorier s_j en gris
11 $\pi[s_j] \leftarrow s_i$
12 sinon
13 Dépiler s_i de p et colorier s_i en noir

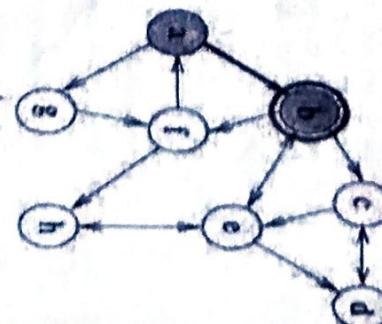
14 retourne π

1 Fonction $DFS(g, s_0)$

2 Soit p une pile (LIFO) initialisée à vide
3 pour tout sommet $s_i \in S$ faire
4 $\pi[s_i] \leftarrow null$
5 Colorier s_i en blanc

6 Empiler s_0 dans p et colorier s_0 en gris
7 tant que p n'est pas vide faire
8 Soit s_i le dernier sommet entré dans p
9 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors
10 Empiler s_j dans p et colorier s_j en gris
11 $\pi[s_j] \leftarrow s_i$
12 sinon
13 Dépiler s_i de p et colorier s_i en noir

14 retourne π



X

Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris

tant que p n'est pas vide faire

 Soit s_i le dernier sommet entré dans p

 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

 Empiler s_j dans p et colorier s_j en gris

 sinon

 Dépiler s_i de p et colorier s_i en noir

retourne π

Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

 Soit s_i le dernier sommet entré dans p

 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

 Empiler s_j dans p et colorier s_j en gris

 sinon

 Dépiler s_i de p et colorier s_i en noir

retourne π

Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

 Soit s_i le dernier sommet entré dans p

 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

 Empiler s_j dans p et colorier s_j en gris

 sinon

 Dépiler s_i de p et colorier s_i en noir

retourne π

Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

 Soit s_i le dernier sommet entré dans p

 si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

 Empiler s_j dans p et colorier s_j en gris

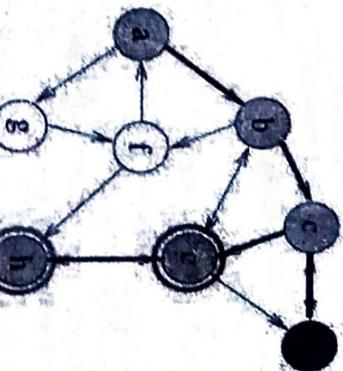
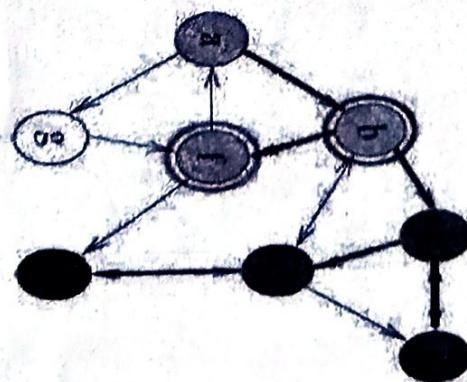
 sinon

 Dépiler s_i de p et colorier s_i en noir

retourne π

$$p = \langle a, b, f \rangle$$

$$s_i = b, s_j = f$$



Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

Soit s_i le dernier sommet entré dans p
si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

Empiler s_j dans p et colorier s_j en gris
 $\pi[s_j] \leftarrow s_i$

sinon

Dépiler s_i de p et colorier s_i en noir

$p = \langle a, b, c \rangle$
 $s_i = b, s_j = c$

retourne π

Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

Soit s_i le dernier sommet entré dans p
si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

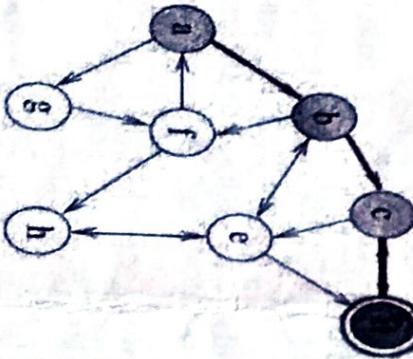
Empiler s_j dans p et colorier s_j en gris
 $\pi[s_j] \leftarrow s_i$

sinon

Dépiler s_i de p et colorier s_i en noir

$p = \langle a, b, c \rangle$
 $s_i = b, s_j = c$

retourne π



Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

Soit s_i le dernier sommet entré dans p
si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

Empiler s_j dans p et colorier s_j en gris
 $\pi[s_j] \leftarrow s_i$

sinon

Dépiler s_i de p et colorier s_i en noir

$p = \langle a, b, c, d \rangle$
 $s_i = c, s_j = d$

retourne π

Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris
tant que p n'est pas vide faire

Soit s_i le dernier sommet entré dans p
si $\exists s_j \in succ(s_i)$ tel que s_j soit blanc alors

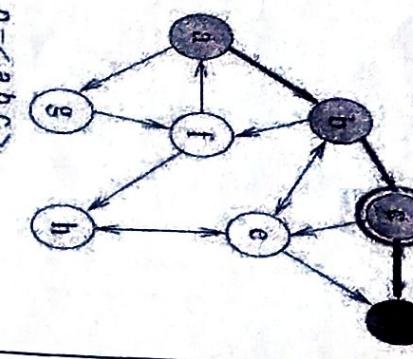
Empiler s_j dans p et colorier s_j en gris
 $\pi[s_j] \leftarrow s_i$

sinon

Dépiler s_i de p et colorier s_i en noir

$p = \langle a, b, c \rangle$
 $s_i = c$

retourne π



1 Fonction $DFS(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris

tant que p n'est pas vide faire

Soit s_j le dernier sommet entré dans p

si $\exists s_j \in succ(s_j)$ tel que s_j soit blanc alors

Empiler s_j dans p et colorier s_j en gris

$\pi[s_j] \leftarrow s_j$

sinon

Dépiler s_j de p et colorier s_j en noir

$\pi[s_j] \leftarrow s_j$

retourne π

1 Fonction $DFSG(g, s_0)$

Soit p une pile (LIFO) initialisée à vide
pour tout sommet $s_i \in S$ faire

$\pi[s_i] \leftarrow null$

Colorier s_i en blanc

Empiler s_0 dans p et colorier s_0 en gris

tant que p n'est pas vide faire

Soit s_j le dernier sommet entré dans p

si $\exists s_j \in succ(s_j)$ tel que s_j soit blanc alors

Empiler s_j dans p et colorier s_j en gris

$\pi[s_j] \leftarrow s_j$

sinon

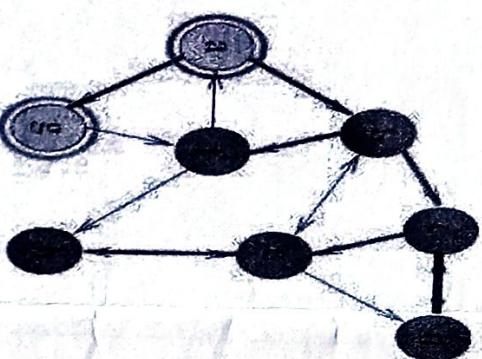
Dépiler s_j de p et colorier s_j en noir

$\pi[s_j] \leftarrow s_j$

retourne π

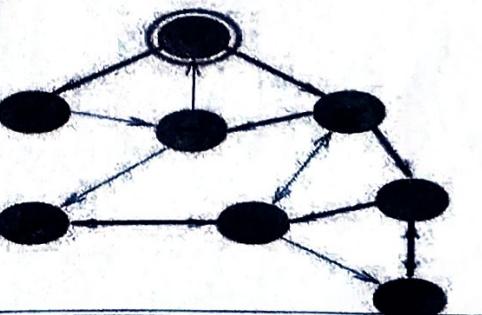
$p = \leftarrow$

$s_i = a$



$p = \leftrightarrow$

$s_i = a$



Conclusion :

Le parcours d'un graphe : l'algorithme de largeur on traite en prioritaire le sommet découvert en premier et en algorithme de profondeur on traite le sommet découvert au plus tard.

TD - Chap 2 Algorithme de Parcours

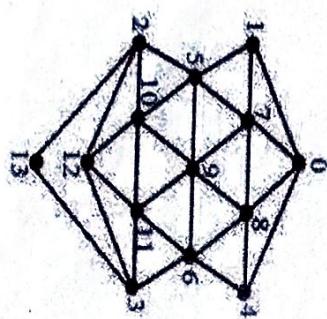
Exercice 1:

Construire un graphe non orienté de 11 sommets ($d(v) = 3$ pour le tous les sommets). Montrez que c'est un graphe régulier. Appliquez l'algorithme de parcours en largeur.

Exercice 2:

Pour ce graphe non orienté à 14 sommets, les voisins de chaque sommet sont supposés écrits dans l'ordre croissant de leurs numéros. Ainsi 0 a pour voisins 1, 4, 7, 8 ; 1 a pour voisins 0, 5, 7 ; 2 a pour voisins 5, 10, 12, 13 ; etc.

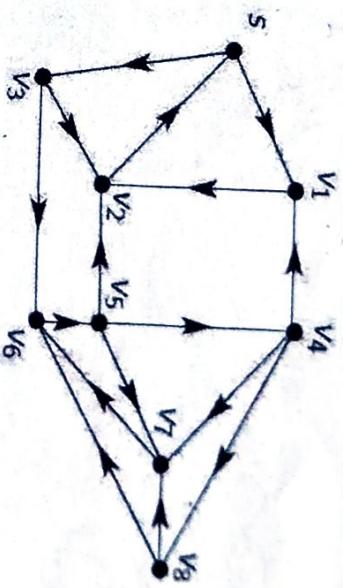
- 1) En partant du sommet 0, faire une exploration en largeur du graphe. On aura intérêt à utiliser l'évolution d'une file, afin de dessiner l'arbre final de l'exploration



- 2) En partant du sommet 0, faire une exploration en profondeur de ce graphe, en utilisant l'ordre de voisins tel qu'il a été défini. Dessiner l'arbre obtenu.

Exercice 3:

Appliquez l'algorithme de parcours en largeur le graphe orienté ci-dessous :



Exercice 4:

Ce graphe à cinq sommets est tel que deux sommets quelconques sont reliés par un arc unique, dans un sens ou dans l'autre. Ce type de graphe a comme propriété d'avoir un nombre impair de chemins hamiltoniens (avec le sommet final différent du sommet initial) lorsque l'on prend les cinq points de départ possibles à tour de rôle.

- 1) Déterminer tous les chemins hamiltoniens partant de chacun des 5 sommets (0, puis 1, puis 2, puis 3 et enfin 4), et se terminant en un sommet différent du point de départ. Les dessiner dans chacun de ces 5 cas. Combien y en a-t-il ?
- 2) Vérifier que pour chacun des points de départ, un des chemins hamiltoniens peut être prolongé pour donner un cycle hamiltonien. Les cinq cycles hamiltoniens obtenus constituent le même cycle si l'on ne tient pas compte du point de départ.
- 3) Est-il possible d'appliquer l'algorithme de parcours en largeur.

Itens

Chapitre III : Le plus court chemin

INTRODUCTION

En théorie des graphes, le problème de plus court chemin est le problème algorithmique qui consiste à trouver un chemin d'un sommet à un autre de façon que la somme des poids des arcs de ce chemin soit minimale. Pour calculer un plus court chemin, il existe de nombreux algorithmes, selon la nature des valeurs et des contraintes supplémentaires qui peuvent être imposées. Dans de nombreux cas, il existe des algorithmes de complexité en temps polynomial, comme l'algorithme de Dijkstra dans des graphes avec poids positifs.

Pour accéder à un sommet u depuis s , plusieurs chaînes/chemins optimaux peuvent exister. La chaîne optimale (le chemin optimal) n'est pas forcément celle (celui) d'un plus petit nombre d'arêtes (arcs). Si le préédécesseur de u dans une chaîne optimale (un chemin optimal) est un voisin [entrant] v de u , alors la sous-chaîne (le sous-chemin) de s à v est optimale aussi.

On a donc :

$$\text{Dist}(s; u) = \min \{ \text{Dist}(s; v) + W(vu) \};$$
$$v \in E(G)$$

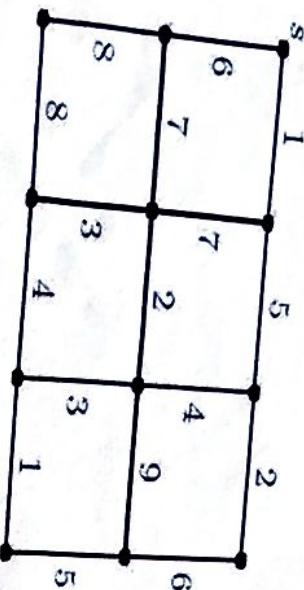
Exemple : Calcul de distances

Entrée : Un graphe (orienté ou pas) G , avec une pondération des arêtes $W : E(G) \rightarrow \mathbb{R}^+$; un sommet de départ s .

Sortie : La distance $\text{dist}(s; v)$ pour tout sommet v

Rappel : la distance entre deux sommets est le poids minimum d'une chaîne (d'un chemin) les reliant.

Graph G avec les arêtes pondérées, calculer la distance minimale; sommet de départ est s .



Pour accéder à un sommet u depuis s , plusieurs chaînes/chemins optimaux peuvent exister.

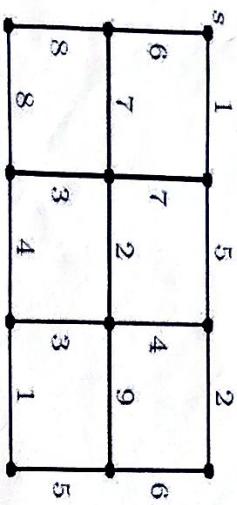
I. Algorithme de Dijkstra

Edsger Wybe Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de calculer le plus court chemin entre un sommet particulier et tous les autres. Les sommets sont visités dans l'ordre croissant de leur distance à partir du sommet de départ s . À chaque étape, la distance d'un sommet est marquée définitive (le sommet est colorié noir);

Les distances d'autres sommets sont éventuellement mises à jour. Si pour un sommet u la valeur de $d[u]$ est différente de ∞ , alors il existe au moins une chaîne de s à u ; parmi les chaînes qui relient s et u , dans celle de longueur (poids total) $d[u]$ le prédecesseur de u est mémorisé comme $père[u]$.

Le résultat est une arborescence, c'est-à-dire un arbre avec un sommet particulier appelé racine. Numérotons les sommets du graphe $G = (V, E)$ de 1 à n . Supposons que l'on s'intéresse aux chemins partant du sommet 1. On construit un vecteur $\lambda = (\lambda(1), \lambda(2), \dots, \lambda(n))$ ayant n composantes tel que $\lambda(j)$ soit égal à la longueur du plus court chemin allant de 1 au sommet j . On initialise ce vecteur à $C1j$, c'est-à-dire à la première ligne de la matrice des coûts du graphe, définie comme indiqué ci-dessous :

ù $\delta(i, j) > 0$ est le poids de l'arc (i, j) . On construit un autre vecteur p pour émporter le chemin pour aller du sommet 1 au sommet voulu. La valeur $p(i)$ nne le sommet qui précède i dans le chemin.



Pour accéder à un sommet u depuis s , plusieurs chaînes/chemins optimaux peuvent exister.

I. Algorithme de Dijkstra

Edgser Wybe Dijkstra (1930-2002) a proposé en 1959 un algorithme qui permet de calculer le plus court chemin entre un sommet particulier et tous les autres. Les sommets sont visités dans l'ordre croissant de leur distance à partir du sommet de départ s . A chaque étape, la distance d'un sommet est marquée définitive (le sommet est colorié noir) ;

Les distances d'autres sommets sont éventuellement mises à jour. Si pour un sommet u la valeur de $d[u]$ est différente de ∞ , alors il existe au moins une chaîne de s à u ; parmi les chaînes qui relient s et u , dans celle de longueur (poids total) $d[u]$ le prédecesseur de u est mémorisé comme $père[u]$.

Le résultat est une arborescence, c'est-à-dire un arbre avec un sommet particulier appelé racine. Numérotons les sommets du graphe $G = (V, E)$ de 1 à n . Supposons que l'on s'intéresse aux chemins partant du sommet 1. On construit un vecteur $\lambda = (\lambda(1); \lambda(2); \dots; \lambda(n))$ ayant n composantes tel que $\lambda(j)$ soit égal à la longueur du plus court chemin allant de 1 au sommet j . On initialise ce vecteur à C_{1j} , c'est-à-dire à la première ligne de la matrice des coûts du graphe, définie comme indiqué ci-dessous :

où $\delta(i, j) > 0$ est le poids de l'arc (i, j) . On construit un autre vecteur p pour mémoriser le chemin pour aller du sommet 1 au sommet voulu. La valeur $p(i)$ donne le sommet qui précède i dans le chemin.

$$c_{ij} = \begin{cases} 0 & \text{si } i = j \\ \infty & \text{si } i \neq j \text{ et } (i, j) \notin E \\ \delta(i, j) & \text{si } i \neq j \text{ et } (i, j) \in E \end{cases}$$

On considère ensuite deux ensembles de sommets, S initialisé à $\{1\}$ et T initialisé à $\{2, 3, \dots, n\}$. À chaque pas de l'algorithme, on ajoute à S un sommet jusqu'à ce que $S = V$ de telle sorte que le vecteur λ donne à chaque étape la longueur minimale des chemins de 1 aux sommets de S .

I.1. Principe de l'algorithme de Dijkstra

On suppose que le sommet de départ (qui sera la racine de l'arborescence) est le sommet numéroté 1. Notons qu'on peut toujours renommerer les sommets pour que ce soit le cas.

Initialisation :

$\lambda(j) = c_{1j}$ et $p(j) = \text{NIL}$, pour $1 \leq j \leq n$

Pour $2 \leq j \leq n$ faire

Si $c_{1j} < \infty$ alors $p(j) = 1$.
 $S = 1$; $T = \{2, 3, \dots, n\}$.

Iterations :

Tant que T n'est pas vide faire

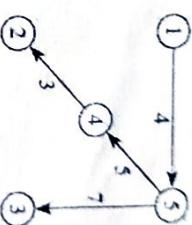
Choisir i dans T tel que $\lambda(i)$ est minimum

Retirer i de T et l'ajouter à S

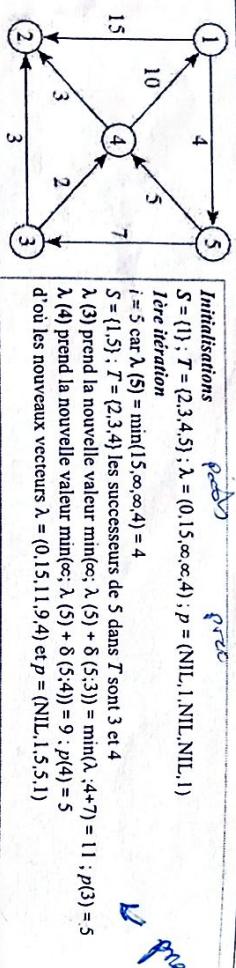
Pour chaque successeur j de i , avec j dans T , faire

Si $\lambda(j) > \lambda(i) + \delta(i, j)$ alors
 $\lambda(j) = \lambda(i) + \delta(i, j)$
 $p(j) = i$

Voici la réponse sous forme d'arborescence :



Exercice d'application:



2ème itération

$i = 4; \lambda(4) = 9$
 $S = \{1, 5, 4\}; T = \{2, 3\}$ le seul successeur de 4 dans T est 2
 $\lambda(2)$ prend la nouvelle valeur $\min(15, \lambda(4) + \delta(4; 2)) = \min(15, 9+3) = 12; p(2) = 4$
d'où les nouveaux vecteurs $\lambda = (0, 12, 11, 9, 4)$ et $p = (\text{NIL}, 4, 5, 5, 1)$

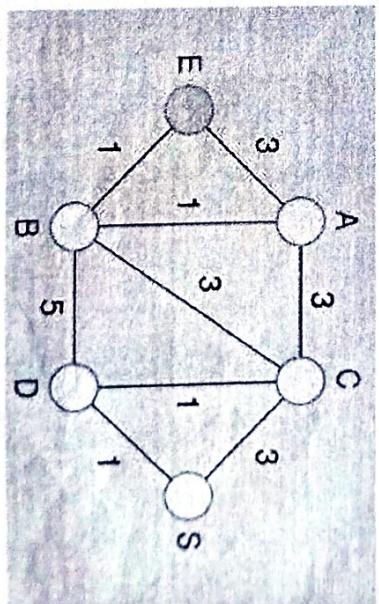
3ème itération

$i = 2; \lambda(2) = 12$
 $S = \{1, 5, 4, 3, 2\}; T = \{\}$
 $\lambda = (0, 12, 11, 9, 4)$
 $p = (\text{NIL}, 4, 5, 5, 1)$

L'algorithme se termine, car $T = \{\}$.

On peut lire les coûts des chemins les plus courts dans λ et les chemins eux-mêmes grâce à la vectrice p . Par exemple, le chemin minimal de 1 à 4 est de coût 9, car $\lambda(4) = 9$. C'est le chemin 1 → 5 → 4, car $p(4) = 5$ et $p(5) = 1$.

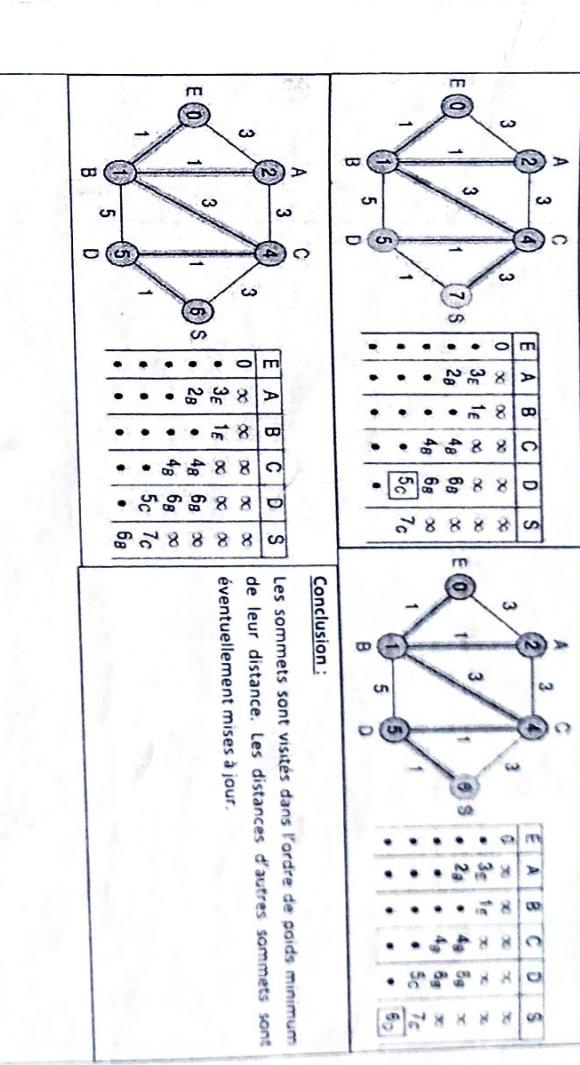
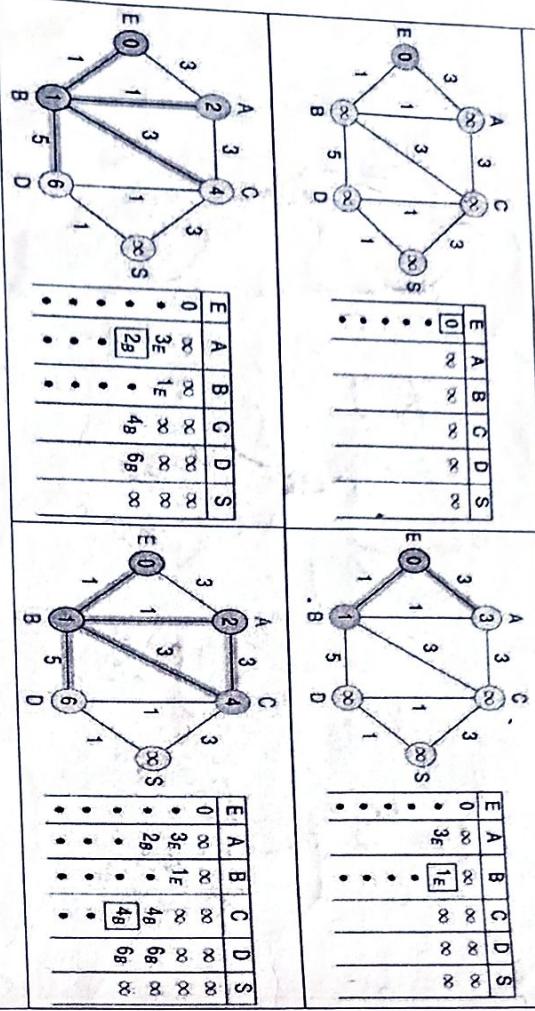
$p \cdot g = 45 | 63$



Donnez le résultatat de plus court chemin avec l'algorithme de DIJKSTRA
Dans un graphe non orienté cherchons les plus courts chemins d'origine E du graphé ci-dessous :

La

solution :



III) Algorithme de Bellman-Ford

L'**algorithme de Bellman-Ford**, aussi appelé **algorithme de Bellman-Ford-Moore**, est un algorithme qui calcule des plus courts chemins depuis un sommet source donné dans un **graphe orienté pondéré**. Il porte le nom de ses inventeurs **Richard Bellman** et **Lester Randolph Ford Junior** (publications en 1956 et 1958), et de **Edward Forrest Moore** qui le redécouvrit en 1959. Contrairement à l'**algorithme de Dijkstra**, l'algorithme de Bellman-Ford autorise la présence de certains arcs de poids négatif et permet de détecter l'existence d'un circuit absorbant, c'est-à-dire de poids total strictement négatif, accessible depuis le sommet source.

Principe de l'algorithme:

Soit $\mathbf{G} = (\mathbf{X}, \mathbf{U})$ un graphe dont les arcs sont munis de longueurs réelles quelconques. On cherche les plus courts chemins de la source à tous les autres sommets du graphe.

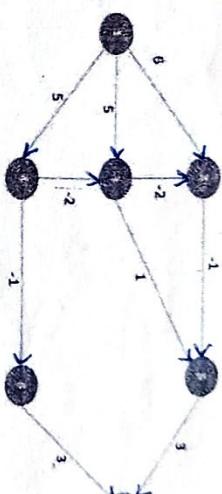
Remarque préliminaire : Soit un graphe G présentant sur certains arcs de poids négative et, envisageons de rendre les valuations toutes positives ou nulles en ajoutant à chaque valeur la valeur absolue maximale des valeurs négatives.

- initialise les distances de la source à tous les sommets en tant qu'infini et la distance à la source elle-même en tant que 0. Créez un tableau $dist[]$ de taille $|V|$ avec toutes les valeurs infinies sauf $dist[src]$ où src est le sommet source.
- Faire $|V|-1$ fois où $|V|$ est le nombre de sommets dans le graphe les instructions suivantes.
 - Répéter pour chaque arête $u-v$
 - Si $dist[v] > dist[u] + \text{poids}$ - l'arête $u-v$, mettez à jour $dist[v] = dist[u] + \text{poids}$ de l'arête $u-v$
- Si $dist[v] > dist[u] + \text{poids}$ de l'arête $u-v$, alors "le graphe contient un cycle de pondération négatif"

L'idée de l'étape 3 est que l'étape 2 garantit les distances les plus courtes si le graphe ne contient pas de cycle de pondération négatif. Si nous

parcourons toutes les arêtes une fois de plus et nous obtenons un chemin plus court pour tout sommet, il y a un cycle de pondération négatif.

Exemple d'application



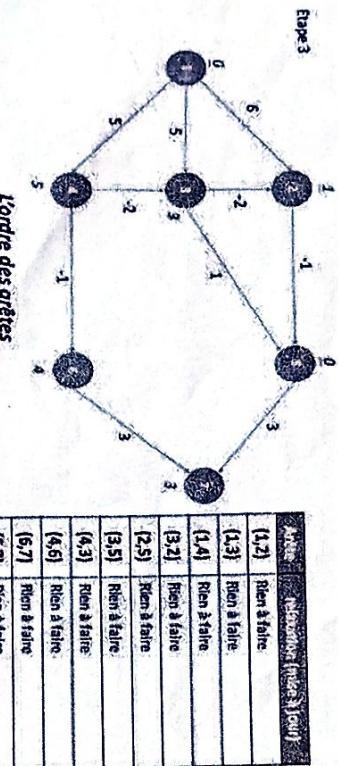
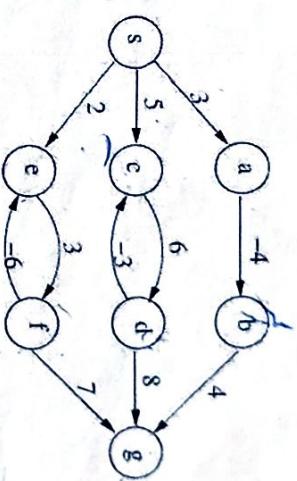
L'ordre des arêtes
[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]



L'ordre des arêtes
[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

Exercice d'application

Trouvez le plus court chemin du graphe ci-dessous en appliquant l'algorithme de Bellman-Ford.



l'ordre des arêtes
[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

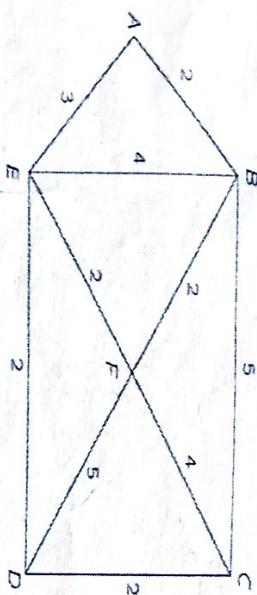
l'ordre des arêtes
[(1,2), (1,3), (1,4), (3,2), (2,5), (3,5), (4,3), (4,6), (6,7), (5,7)]

STOP

TD1-Chapitre III : Plus court chemin

Exercice : QCM

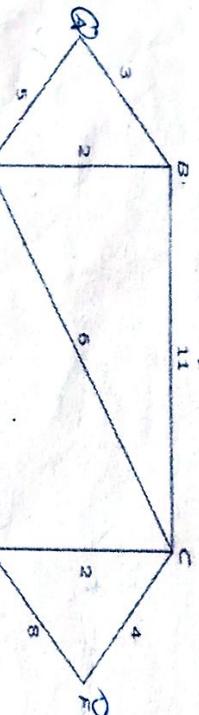
On représente sur le graphe G ci-dessous les liaisons routières entre six places (A, B, C, D, E, F) d'un centre-ville. Sur chaque route est indiqué le nombre de feux tricolores présents entre les deux places qu'elle relie.



Un automobiliste souhaite se rendre de B à D en empruntant le trajet comportant le moins de feux tricolores possible. Quel itinéraire cet automobiliste doit-il emprunter ?

- F - B - D - E
- D - E - F - B
- A - C - B - D
- B - F - E - D

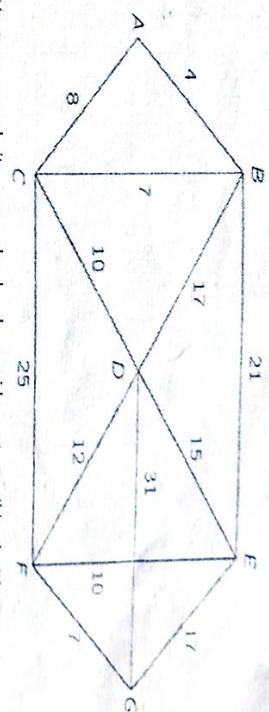
On représente sur le graphe ci-dessous les liaisons routières entre six places (A, B, C, D, E, F) d'un centre-ville. Sur chaque route est indiqué le nombre de feux tricolores présents entre les deux places qu'elle relie.



Un automobiliste souhaite se rendre de A à F en empruntant le trajet comportant le moins de feux tricolores possible. Quel itinéraire cet automobiliste doit-il emprunter ?

- A-B-F-E-C-D
- A-B-C-D-E-F
- F-C-E-D-B-A
- A-B-E-D-C-F

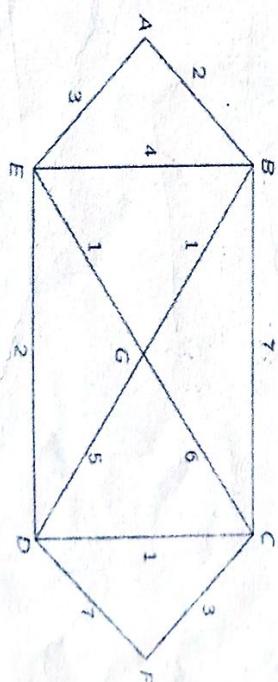
On représente sur le graphe ci-dessous les liaisons ferroviaires entre sept gares (A, B, C, D, E, F, G). Sur chaque ligne est indiqué le temps de trajet en minutes (correspondance comprise) entre les deux gares qu'elle relie.



Un usager souhaite se rendre le plus rapidement possible de B à G . Quel itinéraire cet automobiliste doit-il emprunter ?

- B-C-D-A-F
- B-C-D-F-G
- G-F-D-C-B
- B-C-D-A-G

On représente sur le graphe ci-dessous les liaisons ferroviaires entre sept gares (A, B, C, D, E, F, G). Sur chaque ligne est indiqué le temps de trajet en minutes (correspondance comprise) entre les deux gares qu'elle relie.

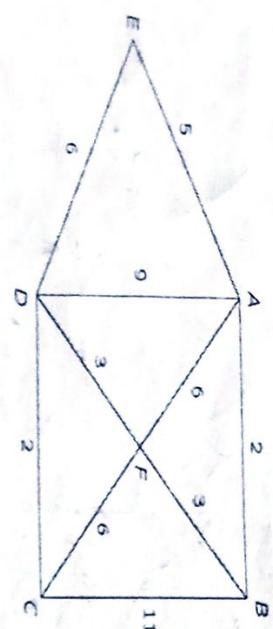


Un usager souhaite se rendre le plus rapidement possible de B à F . Quel itinéraire cet automobiliste doit-il emprunter ?

- F-C-D-E-G-B
- F-C-D-E-G-A
- B-G-E-D-C-F
- B-C-A-D-G-F

On représente sur le graphe G ci-dessous les liaisons routières entre six villes (A, B, C, D, E, F). Sur chaque route est indiqué le temps de trajet (en minutes) entre les deux villes qu'elle relie.

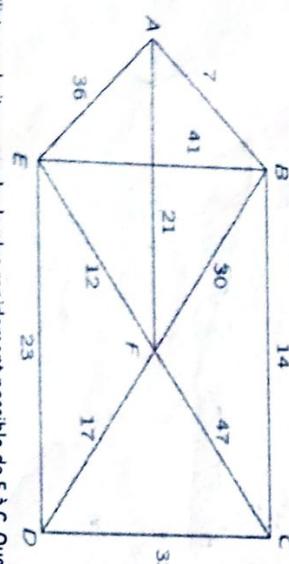
E - F - C



Un automobiliste souhaite se rendre le plus rapidement possible de C à A . Quel itinéraire cet automobiliste doit-il emprunter ?

- A - B - F - D - C
- C - E - F - D - A
- C - F - D - E - A
- C - D - F - B - A

On représente sur le graphe G ci-dessous les liaisons routières entre six villes (A, B, C, D, E, F). Sur chaque route est indiqué le temps de trajet (en minutes) entre les deux villes qu'elle relie.



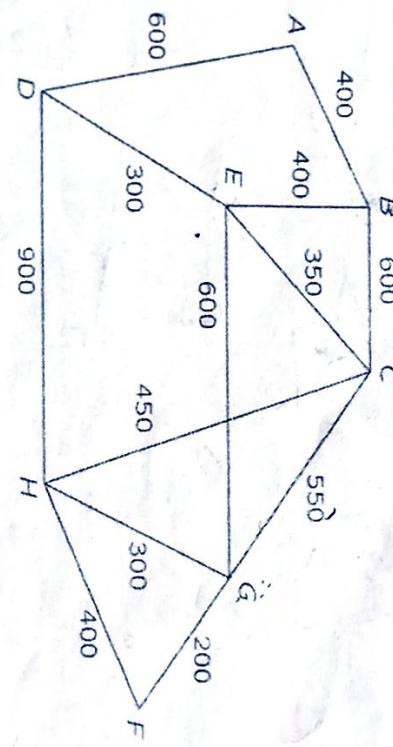
Un automobiliste souhaite se rendre le plus rapidement possible de E à C . Quel itinéraire cet automobiliste doit-il emprunter ?

- E - D - C
- E - F - D - C
- E - F - A - B - C

Exercice 2 : Algorithme de Dijkstra

On représente sur le graphe G ci-dessous les liaisons routières entre 8 villes (A, B, C, D, E, F, G, H). Sur chaque route est indiquée la distance en km entre les deux villes qu'elle relie.

E - F - C



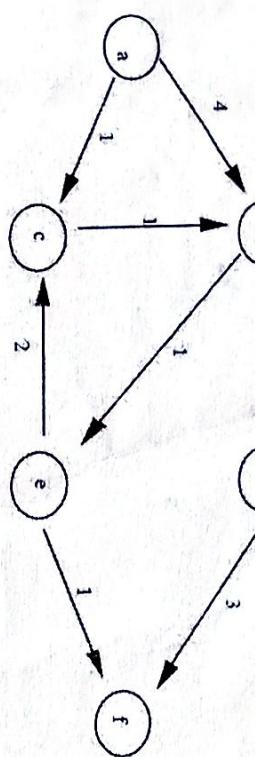
En partant de la ville A quelle serait le plus court chemin entre A et les autres villes. Appliquez l'algorithme de DIJKSTRA ?

TD-2 Plus Court Chemin

Exercice 1:

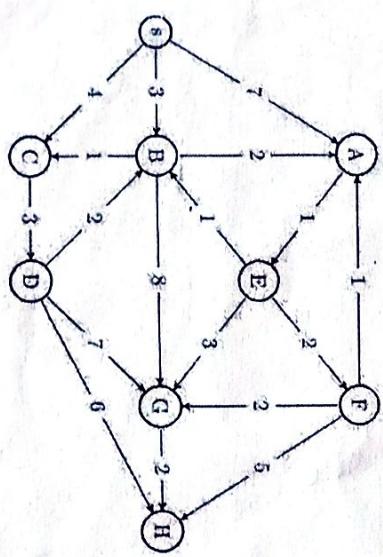
Déterminer dans le graphe suivant les plus courts chemins à partir du sommet f.
Utiliser l'algorithme de Ford-Bellman (préciser pourquoi cela est nécessaire)

en parcourant les sommets dans l'ordre alphabétique.



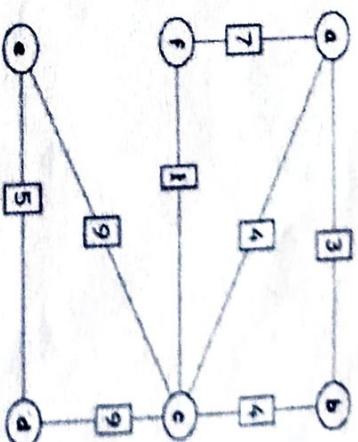
Exercice 2:

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet. S
Utiliser l'algorithme de Dijkstra.



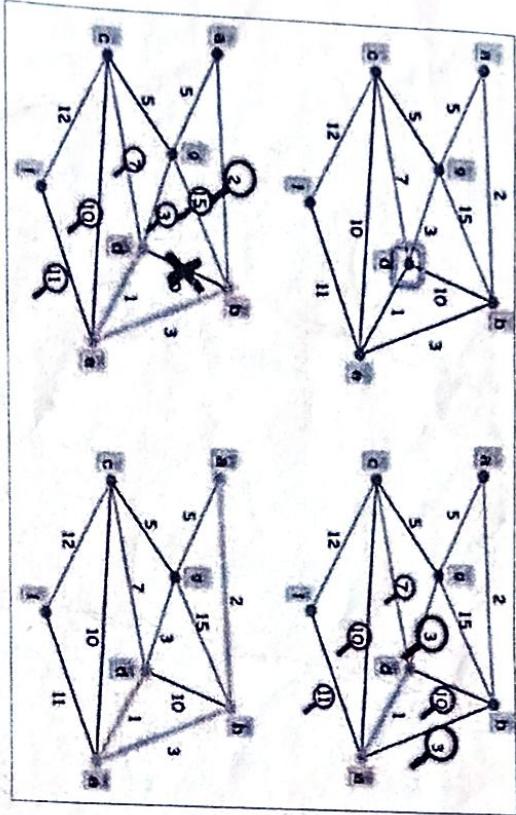
Exercice 3:

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet.
Utiliser l'algorithme de KRUSKAL.



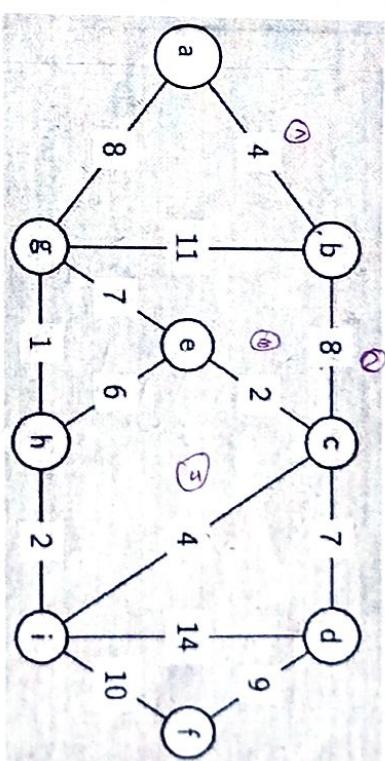
Exercice 4:

Déterminer dans le graphe suivant les plus courts chemins à partir du sommet.
Utiliser l'algorithme de PRIM.



(1)

Principe:
L'algorithme de PRIM construit un arbre T en rajoutant à chaque étape le sommet qui n'est pas encore dans l'arbre et qui possède l'arête de poids minimum parmi celles reliant les sommets de T aux sommets de $G - T$.



Algorithme de Welsh et Powell

1. Repérer le degré de chaque sommet.
2. Ranger les sommets par ordre de degrés décroissants (dans certains cas plusieurs possibilités).
3. Attribuer au premier sommet (A) de la liste une couleur.
4. Suivre la liste en attribuant la même couleur au premier sommet (B) qui ne soit pas adjacent à (A).
5. Suivre (si possible) la liste jusqu'au prochain sommet (C) qui ne soit adjacent ni à A ni à B.
6. Continuer jusqu'à ce que la liste soit finie.
7. Prendre une deuxième couleur pour le premier sommet (D) non encore coloré de la liste.
8. Continuer jusqu'à avoir coloré tous les sommets.

On appelle **coloriage** ou **coloration usuelle** d'un graphe $G=(X,E)$, une application ϕ de X dans l'ensemble des entiers telle que $\phi(x) \neq \phi(y)$ lorsque x et y sont adjacents. Deux sommets adjacents ne peuvent pas être coloriés de la même couleur et tous les sommets doivent être coloriés.

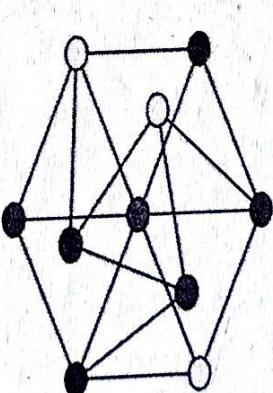
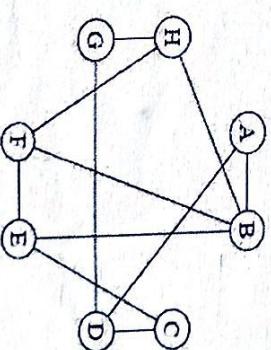
Définition : Nombre chromatique d'un graphe

On appelle **nombre chromatique** d'un graphe $G=(X,E)$, le plus petit nombre de couleurs nécessaires pour colorier ce graphe. Ce nombre est noté $\chi(X,E)$.

Le nombre chromatique est toujours compris entre 1 et le nombre de points.

Principe des Algorithmes de coloriage :

La notion de coloration n'est définie que pour les graphes sans boucle, et la multiplicité des arêtes ne joue aucun rôle. Donc, soit G un graphe simple (sans boucle ni arête multiple)

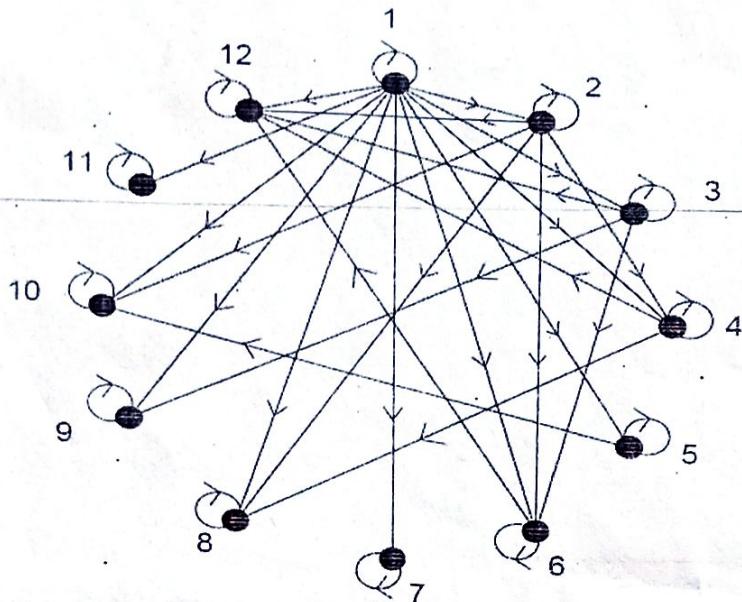
Exercice d'application :

LA THEORIE DES GRAPES

Les solutions

Solutions N°1 :

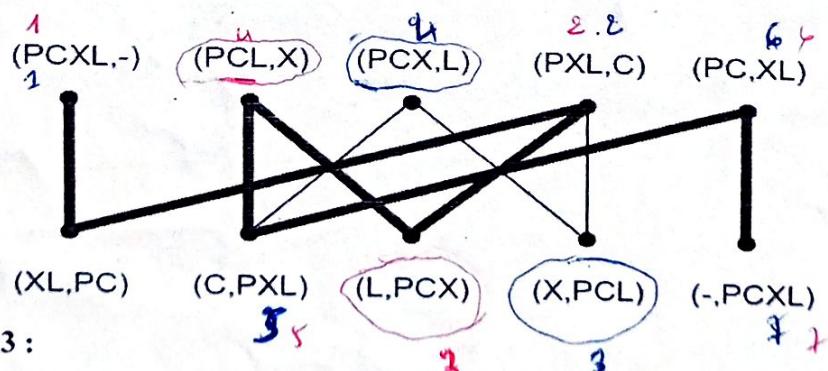
Aucune difficulté : dans cette exercice, on peut désigner un graphe orienté (n'oublier pas les boucles)



Solutions N°2 :

Cette situation peut être modélisée à l'aide d'un graphe. Désignons par P le passeur, par C la chèvre, par X le chou et par L le loup. Les sommets du graphe sont des couples précisant qui est sur la rive initiale, qui est sur l'autre rive. Ainsi, le couple (PCX,L) signifie que le passeur est sur la rive initiale avec la chèvre et le chou (qui sont donc sous surveillance), alors que le loup est sur l'autre rive. Une arête relie deux sommets lorsque le passeur peut passer (sic) d'une situation à l'autre. En transportant la chèvre, le passeur passe par exemple du sommet (PCX,L) au sommet (X,PCL). Le graphe ainsi obtenu est biparti : les sommets pour lesquels le passeur est sur la rive initiale ne sont reliés qu'aux sommets pour lesquels le passeur est sur l'autre rive...

Naturellement, on ne considérera pas les sommets dont l'une des composantes est CX ou LC car ces situations sont interdites. Il suffit ensuite de trouver un chemin (le plus court par exemple) entre la situation initiale (PCXL,-) et la situation finale souhaitée (-,PCXL). La figure suivante donne un tel chemin :



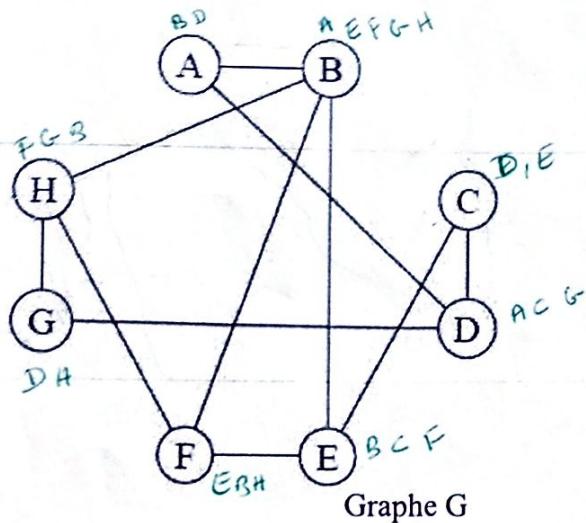
Solution N°3 :

Sommet	A	B	C	D	E	F	G	H	I
Degré	4	6	4	2	4	4	6	4	2

SUJET D'ENTRAINEMENT AU CC

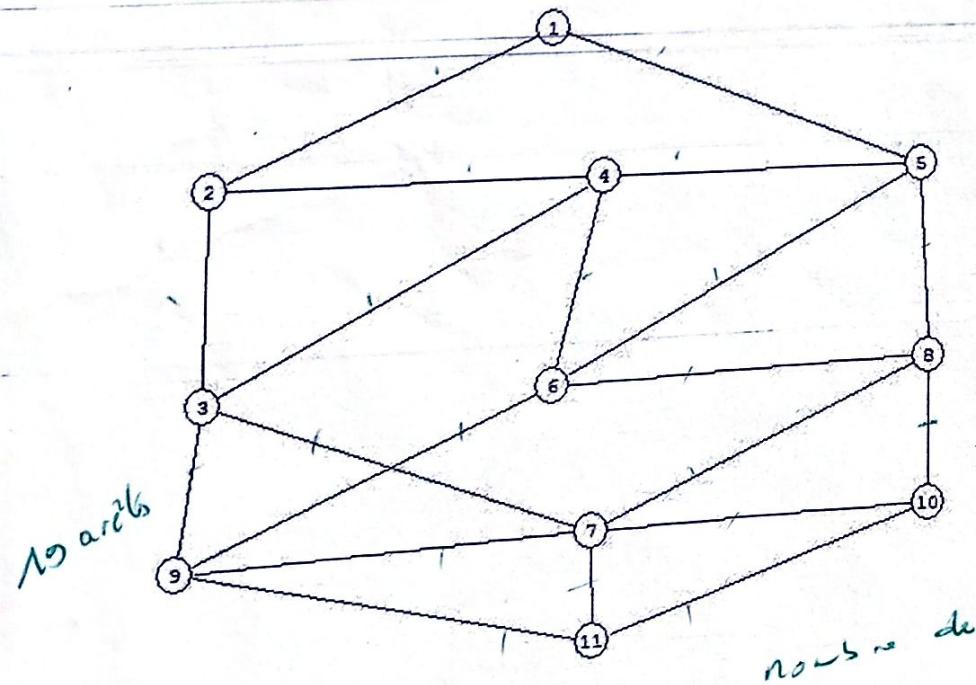
Exercice 1 :

Huit personnes se retrouvent pour un repas de mariage. Le graphe ci-dessous précise les incompatibilités d'humeur entre ces personnes (une arête reliant deux personnes indique qu'elles ne se supportent pas).

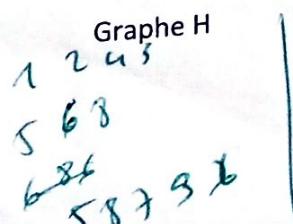


- 1) Y-a-t-il un cycle hamiltonien ?
- 2) Y-a-t-il une chaîne eulérienne ?
- 3) Y-a-t-il un cycle eulérien ?
- 4) Proposez un plan de table en évitant de placer côté à côté deux personnes incompatibles. Et élaborer la matrice d'incidence ? justifier

EXERCICE 2 :



- 1) Est-il connexe ? justifier ?
- 2) Combien des sous-graphes y a t-il dans le graphe H ?
- 3) Elaborer la matrice adjacente du graphe H ?
- 4) Rendre le graphe H en graphe régulier ? Justifier ?
- 5) Appliquer l'algorithme de parcours en profondeur ?



FICHE DU DÉROULEMENT DE LA SÉANCE DU TP 2

Théorie et Algorithme des Graphes

Noms des enseignant(e)s :	Mme. Hawa Ali Omar et Mr. Mahsoud Mahamoud Aden		
Filière :	Licence INFOR 3		
TP :	2		
Date	du 24/10/2022 au 7/11/2022	Durée	4 heures
Dans le programme :	Généralité		
Objectifs pédagogiques :	Les étudiants vont créer une application en langage PYTHON. Cette séance leur permettra d'assimiler la création des objets (sous-menus) et comment actionner des boutons en langage PYTHON.		
Pré requis :	Les langages de programmation : PYTHON, JAVA, C#, PHP ou C		
Travaux pratique de la séance précédente	Réalisation d'une interface graphique. Dans cette interface vous allez créer une fenêtre principale qui possède un menu de 5 boutons (fichier, création, affichage, exécution, édition)		
Travaux pratique de la séance	<p>Réalisation des sous-menus dans la fenêtre principale. Le menu principal est composé des 5 boutons et actionnez le menu Fichier et ses sous-menus.</p> <ul style="list-style-type: none"> - Fichier (nouveau, ouvrir, enregistrer, enregistre sous, fermer) - Création (sommet, arête) - Affichage (graphe, chaînes, matrices) - Exécution (plus court chemin, coloration) - Edition (graphe) 		
Travaux pratique de la séance prochaine	Vous actionnez le menu création et ses sous-menus (sommet et arête) en créant un graphe non-orienté.		
Matériels nécessaire pour la séance	Ordinateurs, connexion d'internet pour les étudiants. Un vidéo projecteur pour l'enseignante.		