Université Mosteafa BENBOULAID Batna 2

Faculté des Mathématiques et Informatique

Département d'Informatique

Théorie des Graphes

Dr. KADRI Ouahab

Batna le : 19/04/2021

Table des matières

Introduction

Ce cours est destiné aux étudiants de deuxième année en informatique. Son objectif est de présenter à l'étudiant d'une part la modélisation des problèmes sous forme des graphes et d'autre part ce cours contiendra un ensemble de techniques permettant à l'étudiant de résoudre ces problèmes à travers des algorithmes comme la recherche de chemin minimal, le flot maximal etc. Le contenu du cours est essentiellement le suivant :

- Définitions de base,
- Cycles,
- Flots,
- Problèmes de cheminement,
- Problèmes Hamiltonien et Eulérien
- Coloration

Un ensemble d'exercices sont intégrés à la fin du document. Nous avons ajouté cette partie de travaux dirigés pour permettre aux étudiants d'appliquer les notions vues dans les différents chapitres.

Ce document est un support de cours : il ne vise certainement pas être complet ni présenter en détail tous les sujets abordés. L'assistance au cours proprement dit, ainsi qu'aux travaux dirigés est fortement recommandée.

Pour toute question, **remarque** ou suggestion, vous pouvez me contacter par Email à l'adresse suivante :

ouahabk@gmail.com

Pour ceux qui veulent en savoir plus, il existe une liste de références bibliographies présentée à la fin de ce document. Ainsi, j'invite les étudiants à lire le cours de Théorie des graphes et optimisation dans les graphes de Pr. Christine Solnon, dont une grande partie de ce cours est extraite. Ce dernier est disponible en ligne à l'adresse suivante : https://perso.liris.cnrs.fr/christine.solnon/polyGraphes.pdf

J'ai créé une chaîne YouTube dédiée aux théories des graphes accessible via le lien suivant :

https://www.youtube.com/channel/UCIbVQphG2gxp4OVtCirBc2g

Je tiens à remercier toutes les personnes (enseignants et étudiants) ayant participé à améliorer ce support de cours, et notamment Dr. KADACHE Nabil qui a proposé les solutions des exercices.

Chapitre I. Définitions de base

1. Définition "intuitive" d'un graphe

Conceptuellement, un graphe est un ensemble de relations entre un ou plusieurs objets. On l'utilise pour modéliser les réseaux tels que les circuits électriques et les réseaux de transport. Chaque objet peut être représenté sous forme d'un point (sommet, nœud) et on utilise des arcs ou des arêtes pour représenter les relations entre les sommets. Selon le choix de la relation, on peut distinguer deux types de graphes (orienté et non orienté). Et si on associe des valeurs aux relations, on parle de graphe pondéré. La figure 1 représente un exemple d'un graphe non orienté composé de cinq sommets et cinq arêtes

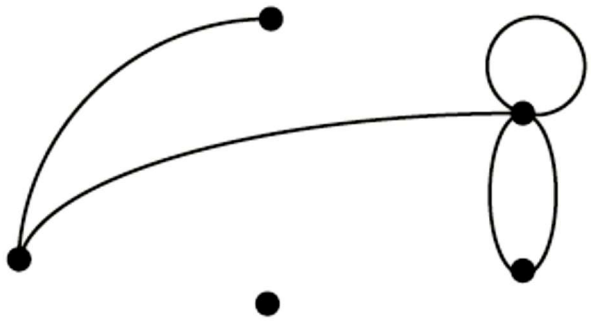


Figure 1: Exemple d'un graphe

2. Définition mathématique d'un graphe

De manière formelle, on peut définir un graphe comme un couple G = (S, A) tel que :

- S est un ensemble fini de sommets (Nœuds),

- A est un ensemble de couples de sommets $(s_i, s_j) \in S^2$.

3. Ordre, orientation et multiplicité

3.1. Ordre

L'ordre d'un graphe est le nombre de ses sommets. Par exemple, l'ordre du graphe de la figure 1 égale à 5 et on écrit |G|=5.

3.2. Orientation

On peut distinguer deux types de graphes :

Graphe orienté : les couples $(s_i, s_j) \in A$ sont orientés, c'est à dire que (s_i, s_j) est un couple ordonné où s_i est le sommet initial, et s_j le sommet terminal. Un couple (s_i, s_j) est appelé un arc, et il est représenté graphiquement par $s_i \rightarrow s_j$.

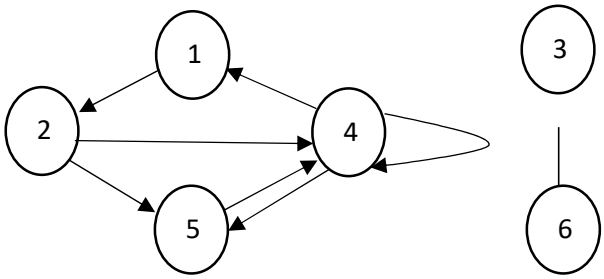


Figure 2 : Graphe orienté

Le graphe orienté G1 = (S, A) avec S = {1, 2, 3, 4, 5, 6} et

A = {(1, 2), (2, 4), (2, 5), (4, 1), (4, 4), (4, 5), (5, 4), (6, 3)}.

Remarques :

- Le graphe de la figure 2 est composé de deux composants.
- L'arc (4, 4) représente une boucle.

Graphe non orienté : on a chaque couple $(s_i, s_j) \in A$ n'est pas orienté, c'est à dire que (s_i, s_j) est équivalent à (s_j, s_i). Le couple (s_i, s_j) est appelé une arête, et il est représenté graphiquement par s_i—s_j.

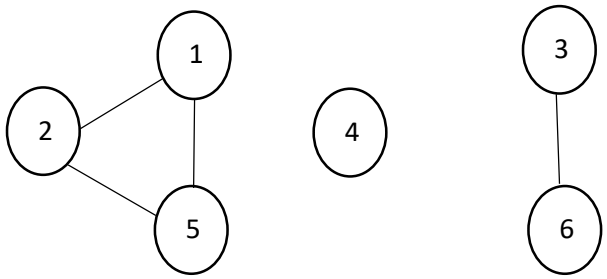


Figure 3 : graphe non orienté

Le graphe non orienté G2 = (S, A) avec S = {1, 2, 3, 4, 5, 6} et

A = {(1, 2), (1, 5), (5, 2), (3, 6)}.

Remarque :

Le graphe de la figure 3 contient un nœud isolé (Sommet 4)

3.3. Multiplicité

Entre deux sommets donnés, il peut y avoir plusieurs arêtes, ce que l'on appelle aussi une arête multiple.

– Une boucle est un arc ou une arête reliant un sommet à lui-même.

Exemple

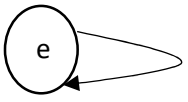


Figure 4 : boucle

– Un graphe non-orienté est dit simple s'il ne comporte pas de boucle, et s'il ne comporte jamais plus d'une arête entre deux sommets. Un graphe non orienté qui n'est pas simple est un multigraphe. Dans le cas d'un multi-graphe, A n'est plus un ensemble mais un multi-ensemble d'arêtes. Exemple : Le graphe 3 est simple.

4. Relations entre les éléments d'un graphe

4.1 Relations entre sommets

– Dans un graphe non orienté, un nœud s_i est dit adjacent à un autre nœud s_j s'il existe une arête entre s_i et s_j. L'ensemble des nœuds voisins à un nœud s_i est défini par :

$\Gamma(s_i)=adj(s_i) = \{s_j /(s_i , s_j) \in A$ ou $(s_j , s_i) \in A\}$

Exemple : dans le graphe 3, l'ensemble des sommets adjacents au sommet 2 sont les sommets {1,5}

– Dans un graphe orienté, on distingue les sommets successeurs des sommets prédécesseurs :

$\Gamma^+(s_i)=succ(s_i) = \{ s_j /(s_i, s_j) \in A\}$

Exemple : dans le graphe 2, l'ensemble des successeurs au sommet 1 est {2}

$\Gamma^-(s_i)=pred(s_i) = \{ s_j /(s_j , s_i) \in A\}$

Exemple : dans le graphe 2, l'ensemble des prédécesseurs au sommet 1 est {4}

$\Gamma(s_i)= \Gamma^+(s_i) \cup \Gamma^-(s_i)$

Exemple : dans le graphe 2, l'ensemble des adjacents au sommet 1 est {2, 4}

4.2 Relations entre arcs et sommets

Dans le cas d'un graphe non orienté, le degré d'un sommet est le nombre d'arêtes dont ce sommet est une extrémité (les boucles étant comptées deux fois). Ce degré vaut 0 si le sommet est isolé.

Exemple : Dans le graphe de la figure 3, les degrés des sommets sont :

X	1	2	3	4	5	6
d(x)	2	2	1	0	2	1

Dans le cas un graphe orienté, on peut distinguer deux types de degré d'un sommet. On note $d^+(s)$ le degré extérieur du sommet s, c'est-à-dire le nombre d'arcs ayant s comme extrémité initiale.

On note $d^-(s)$ le degré intérieur du sommet s, c'est-à-dire le nombre d'arcs ayant s comme extrémité finale.

Le degré du sommet s est : $d(s)=d^+(s) + d^-(s)$

Exemple : Dans le graphe de la figure 2, les degrés des sommets sont :

X	$d^+(X)$	$d^-(X)$	d(X)
1	1	0	1
2	2	1	3
3	0	0	0
4	3	3	6
5	1	2	3
6	0	0	0

4.3 Qualificatifs des graphes

– Un graphe orienté est un p-graphe s'il comporte au plus p arcs entre deux sommets.

Exemple : 3-graphe, il existe trois arcs entre a et b.

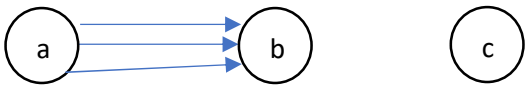


Figure 5 : un 3-graphe

– Un graphe partiel d'un graphe orienté ou non est le graphe obtenu en supprimant certains arcs ou arêtes. Exemple d'un graphe partiel du graphe de la figure 2

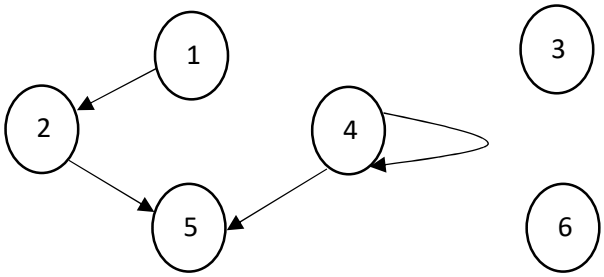


Figure 6 : Un graphe partiel

– Un sous-graphe d'un graphe orienté ou non est le graphe obtenu en supprimant certains sommets et tous les arcs ou arêtes incidents aux sommets supprimés.

Exemple d'un sous graphe de la figure 2

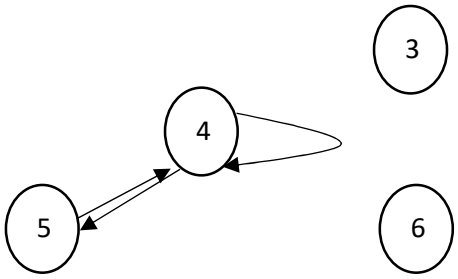


Figure 7 : Un sous graphe

– Un graphe partiel d'un sous-graphe est un sous-graphe partiel. Exemple : un sous graphe du graphe partiel de la figure 5.

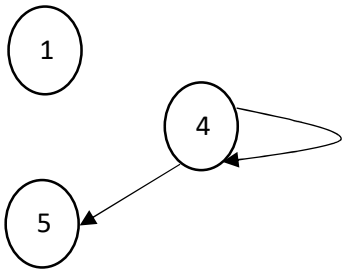


Figure 8 : un sous graphe du graphe partiel de la figure 5.

– Un graphe orienté est dit élémentaire s'il ne contient pas de boucle. Exemple le graphe G1 n'est pas élémentaire puisqu'il contient une boucle.

– Un graphe orienté est dit complet s'il comporte un arc (s_i, s_j) et un arc (s_j, s_i) pour tout couple de sommets différents $s_i, s_j \in S^2$. De même, un graphe non-orienté est dit complet s'il comporte une arête (s_i, s_j) pour toute paire de sommets différents $s_i, s_j \in S^2$.

Exemple

Les figures 9 et 10 représentent des graphes complets

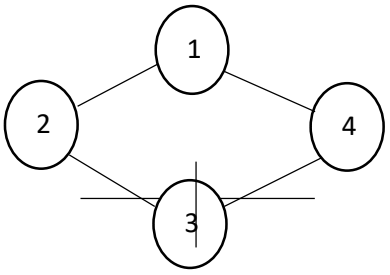


Figure 9 : Un graphe non orienté complet

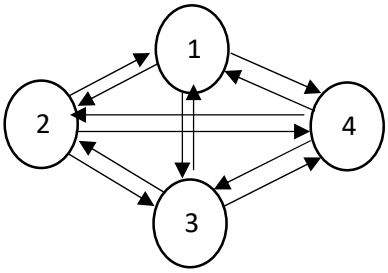


Figure 10 : Un graphe orienté complet

On dit qu'un graphe orienté est symétrique si pour tout arc allant de s_i à s_j, il existe un arc inverse allant de s_j à s_i. Un graphe orienté est dit antisymétrique si pour tout arc allant de s_i à s_j, il n'existe pas un arc allant de s_j à s_i. Dans le cas d'un graphe non orienté, il n'y a aucune logique de dire que le graphe symétrique ou antisymétrique. D'autre part, un graphe orienté peut être symétrique ou antisymétrique, ou ni l'un ni l'autre.

5. Matrices associées à un graphe

5.1 Matrice d'incidence sommet-arc

On peut représenter les relations entre les sommets et les arcs dans un graphe orienté en utilisant une matrice n*m. Sachant que n est le nombre de sommets et m est le nombre des arcs.

Chaque case peut prendre la valeur 1 si le sommet en question est l'extrémité initiale de l'arc.

Elle prend la valeur -1 si le sommet est l'extrémité initiale de l'arc. Elle prend la valeur 0 si le sommet est indépendant de l'arc. Dans le cas d'une boucle, on utilise la valeur 2.

Exemple

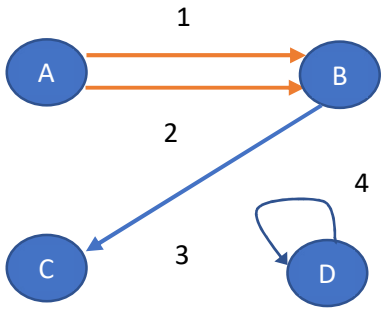


Figure 11: Un graphe orienté multiple

	1	2	3	4

A	+1	+1	0	0
B	-1	-1	+1	0
C	0	0	-1	0
D	0	0	0	2

5.2 Matrice d'adjacence ou d'incidence sommets-sommets

Soit le graphe G = (S, A). On suppose que les sommets de S sont numérotés de 1 à n, avec n =|S|. La représentation par matrice d'adjacence de G consiste en une matrice booléenne M de taille n × n telle que M[i][j] = 1 si (i, j) ∈ A, et M[i][j] = 0 sinon.

Exemple : le graphe de la figure 2 est présenté par la matrice d'adjacence suivante :

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	1	1	0
3	0	0	0	0	0	1
4	1	0	0	1	1	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0

Si le graphe est valué (par exemple, si des distances sont associées aux arcs), on peut utiliser une matrice d'entiers, de telle sorte que M[i][j] soit égal à la valuation de l'arc (i, j) si (i, j) ∈

A. S'il n'existe pas d'arc entre 2 sommets i et j, on peut placer une valeur particulière (par exemple 0 ou -∝ ou null) dans M[i][j].

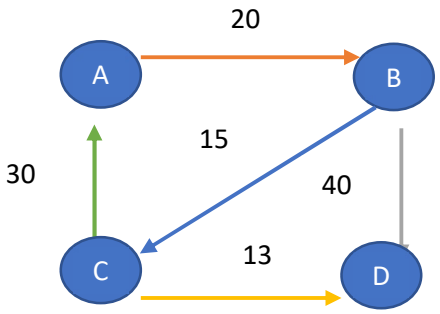


Figure 12: graphe valué

	A	B	C	D
A	0	20	0	0
B	0	0	15	40
C	30	0	0	13
D	0	0	0	0

5.3 Représentation par listes d'adjacence

Soit le graphe G = (S, A). On a l'ensemble S composé des sommets {1, 2, …, n}, avec n = |S|. Pour construire les listes d'adjacence de G, on utilise un tableau T de n listes. Chaque case de ce tableau est un sommet du graphe et utilisée comme une tête d'une liste. Pour chaque sommet $s_i \in S$, la liste d'adjacence $T[s_i]$ est une liste chainée de tous les sommets s_j tels qu'il existe un arc ou une arête $(s_i, s_j) \in A$. Autrement dit, $T[s_i]$ contient la liste de tous les sommets successeurs

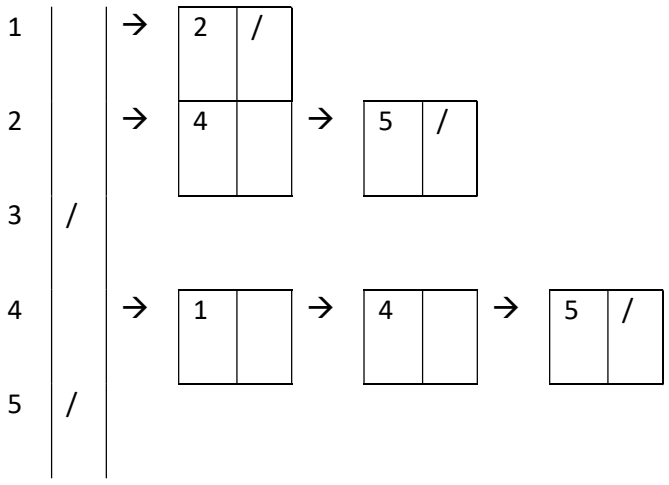
de s_i. Les sommets de chaque liste d'adjacence sont généralement chainés selon un ordre arbitraire.

Si le graphe est valué (par exemple, si les arêtes représentent des distances), on peut stocker dans les listes d'adjacence, en plus du numéro de sommet, la valuation de l'arête.

Dans le cas de graphes non orientés, pour chaque arête (s_i, s_j), on aura s_j qui appartiendra à la liste chainée de T[s_i], et aussi s_i qui appartiendra à la liste chainée de T[s_j].

Exemple

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	0	1	1	0
3	0	0	0	0	0	0
4	1	0	0	1	1	0
5	0	0	0	1	0	0
6	0	0	0	0	0	0



6	/

6. Vocabulaire lié à la connexité

6.1 Chaîne, chemin, longueur

Dans un graphe orienté, un **chemin** allant d'un sommet s_0 vers un sommet s_n est une suite de sommets $< s_0, s_1, s_2, ..., s_n >$ tel que (s_{i-1}, s_i) est un arc appartient à l'ensemble A et cela pour $i \in [1..n]$. La longueur du chemin est le nombre d'arcs dans le chemin, c'est-à-dire n. On dira que le chemin contient les sommets $s_0, s_1, ..., s_n$, et les arcs $(s_0, s_1), (s_1, s_2), ..., (s_{n-1}, s_n)$. S'il existe un chemin de s_0 à s_n, on dira que s_n est accessible à partir de s_0. Un chemin est élémentaire si les sommets qu'il contient sont tous distincts, c'est-à-dire on passe qu'une seule fois par chaque sommet du chemin.

Considérons par exemple le graphe orienté suivant :

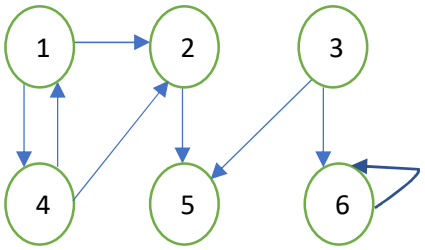


Figure 13: Graphe orienté composé de six sommets

Un chemin élémentaire dans ce graphe est < 1, 4, 2, 5 >.

Un chemin non élémentaire dans ce graphe est < 3, 6, 6, 6 >.

Une « chaîne » dans un graphe est une succession d'arêtes qui permet de relier deux sommets du graphe.

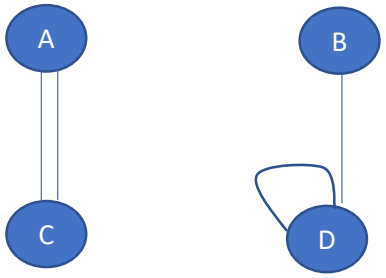


Figure 14 : un graphe illustrant les chaines possibles entre deux sommets

Par exemple, en reliant les deux arêtes {b,d} et {c,d} dans le graphe précèdent, on obtient une chaîne entre les sommets b et c. on peut facilement identifier d'autres chaînes reliant ces deux sommets. On cite entre autres la chaine suivante {b,d}, {d,d}, {d,d}, {d,d}, {d,d}, {c,d}.

Une matrice d'adjacence à la puissance n permet de connaître le nombre de chemins de longueurs n entre n'importe quel couple de point du graphe.

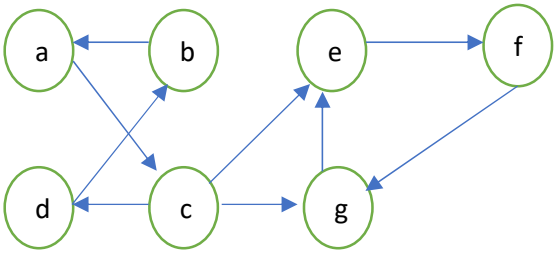


Figure 15: Graphe orienté pour construire une matrice d'adjacence

La matrice d'adjacence associée à ce graphe est la matrice M suivante :

	a	b	c	d	e	f	g

a	0	0	1	0	0	0	0
b	1	0	0	0	0	0	0
c	0	0	0	1	1	0	1
d	0	1	0	0	0	0	0
e	0	0	0	0	0	1	0
f	0	0	0	0	0	0	1
g	0	0	0	0	1	0	0

En multipliant cette matrice par elle-même, on obtient la matrice M2 des chemins de longueur 2 :

	a	b	c	d	e	f	g
a	0	0	0	1	1	0	1
b	0	0	1	0	0	0	0
c	0	1	0	0	1	1	0
d	1	0	0	0	0	0	0
e	0	0	0	0	0	0	1
f	0	0	0	0	1	0	0
g	0	0	0	0	0	1	0

En additionnant M et M^2, on obtient la matrice M +M^2 des chemins de longueur inférieure ou égale à 2 :

	a	b	c	d	e	f	g
a	0	0	1	1	1	0	1
b	1	0	1	0	0	0	0

c	0	1	0	1	1	1	1
d	1	1	0	0	0	0	0
e	0	0	0	0	0	1	1
f	0	0	0	0	1	0	1
g	0	0	0	0	1	1	0

6.2 Connexité

Un graphe non orienté est connexe si chaque sommet est accessible à partir de n'importe quel autre sommet.

Par exemple, le graphe non orienté suivant n'est pas connexe car il n'existe pas de chaîne entre les sommets a et e. En revanche, le sous-graphe défini par les sommets {a, b, c, d} est une **composante connexe.**

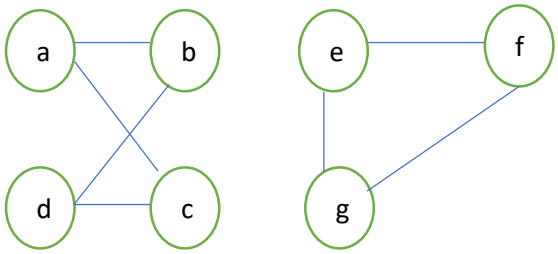


Figure 16: Deux composantes connexes

Un graphe orienté est fortement connexe si chaque sommet est accessible à partir de n'importe quel autre. Par exemple, le graphe de gauche ci-dessous est fortement connexe, tandis que celui de droite ne l'est pas :

Figure 17: Graphe fortement connexe et un autre non connexe

Une composante fortement connexe d'un graphe orienté G est un sous-graphe G^0 de G qui est fortement connexe et maximal (c'est à dire qu'aucun autre sous-graphe fortement connexe de G ne contient G^0). Par exemple, le graphe orienté suivant :

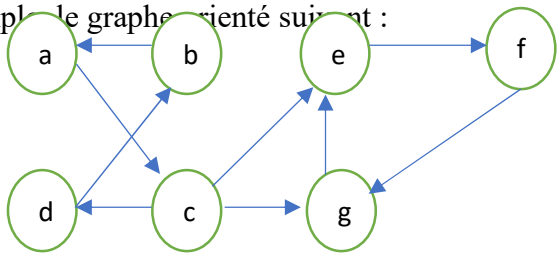


Figure 18: Graphe contenant deux composants fortement connexes

6.3 Cycle et circuit

Dans un graphe orienté, tout **chemin** $< s_0, s_1, s_2, ..., s_k >$ est un circuit si le sommet de départ est le même d'arrivé $s_0 = s_k$ et si le chemin contient au minimum un arc ($k \geq 1$). Ce circuit est élémentaire si en plus les sommets $s_1, s_2, ..., s_k$ sont tous distincts. Une boucle est un circuit de longueur 1.

Par exemple :

Dans la figure 13, le chemin < 1, 2, 5, 4, 1 > est un circuit élémentaire

Dans la figure 13, le chemin < 1, 2, 5, 4, 2, 5, 4, 1 > n'est pas un circuit élémentaire

Toute chaine dont le sommet de départ est le même sommet d'arrivé est appelée un « cycle ». On dit qu'un cycle est élémentaire si les sommets du graphe ne sont visités qu'au plus une fois, et on dit qu'un cycle est simple si aucune arête n'est utilisée deux fois.

Sachant qu'un cycle élémentaire est forcément simple puisque si on passe plusieurs fois sur la même arête implique le passage plusieurs fois sur les sommets composants les extrémités de l'arête en question.

6.4 Cocycle et cocircuit.

Un cocircuit associé à A est l'ensemble des arcs ont comme extrémité un sommet appartenant à A, les arcs sortant de A seront notés positivement et ceux qui entrent vers A seront notés négativement.

Donc, on peut définir un cocycle comme un ensemble des arêtes qui relient un ensemble des sommets aux autres sommets du graphe Si on supprime les arêtes du cocycle, le sous ensemble A devient un composant déconnecté du reste de graphe. Un Cycle et un cocycle sont des notions duales.

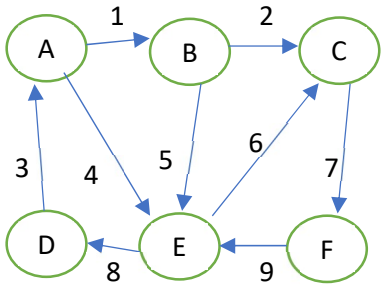


Figure 19: Cocircuit

Exemple : Dans la figure 15, on considère l'ensemble A composé des sommets b et e. On note ω(A) le cocircuit de A. Alors ω(A) = {-1, +2, -4, +6, +8, -9}

Un cocircuit est dit élémentaire s'il est composé d'arcs reliant deux sous-ensembles de sommets connexes qui partitionnent une composante connexe du graphe (donc tous les sommets en cas de graphe connexe).

Le cocircuit donné dans l'exemple ci-dessus n'est pas élémentaire parce qu'il est composé d'arcs reliant {b, e} à {a, c, d, f} et si {b, e} est connexe, ce n'est pas le cas de {a, c, d, f}.

Le cocycle associé à {a, b, d} est élémentaire.

Tout cocycle est la somme de cocycles élémentaires sans arc commun.

Un cocycle est élémentaire si et seulement s'il est minimal.

Chapitre II. Cycles

1. Nombres cyclomatique et cocyclomatique

1.1. Décomposition des cycles et des cocycles en sommes élémentaires

On appelle un Nombres cyclomatique (une base de cycles), un ensemble de cycles linéairement indépendants et générateur. Le nombre cyclomatique, noté V(G) est le nombre d'éléments dans une base de cycles.

Le nombre cyclomatique est calculé en utilisant la formule suivante V(G) = m - n + p où m représente le nombre d'arcs e, n représente le nombre de sommets et p est le nombre de composantes connexes.

Exemple

Sur le graph ci-dessous

m=6 car il y a 6 arcs

n=4 car on a 4 sommets

p=1 car on a 1 connexité

On calcule de v(G)=m-n+p

V(G)=6-4+1=3

Donc nous avons 3 cycles sur le graphe

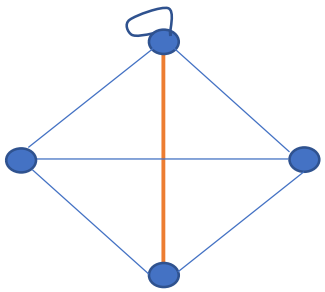


Figure 20 : Nombre cyclomatique

1.2. Lemme des arcs colorés (Minty 1960)

Soit G un graphe dont les arcs sont $u_1, \ldots, u_m$ sont coloriés en rouge, ou vert ou noir et supposons que l'arc u1 est en noir. L'une des deux propositions suivantes est vérifiée

- Il passe par u1 un cycle élémentaire rouge et noir avec tous les arcs noirs sont orientés dans le même sens
- Il passe par u1 un cocycle élémentaire vert et noir avec tous les arcs noirs sont orientés dans le même sens.

Conséquence : Tout arc appartient soit à un circuit élémentaire ou à un cocircuit élémentaire (est un cocycle dont les arcs sont orientés dans le même sens)

1.3. Base de cycles et base de cocycles

On peut calculer Le nombre cyclomatique à partir d'un arbre couvrant, chaque arête qui n'appartient pas à l'arbre couvrant crée un cycle ; l'ensemble des cycles créés par les arêtes manquantes crée une base de cycles.

Soit G=(X, U) un graphe d'ordre n et ayant m arcs et p composantes connexes. La dimension de la base de cocycles appelée nombre cocyclomatique est : W(G)=n-p

2. Planarité

2.1. Graphe Planaire

On appelle graphe planaire tout graphe non orienté pouvant être dessiné sur un plan de telle sorte que les sommets soient des points distincts, et que les arêtes ne se rencontrent pas en dehors de leurs extrémités.

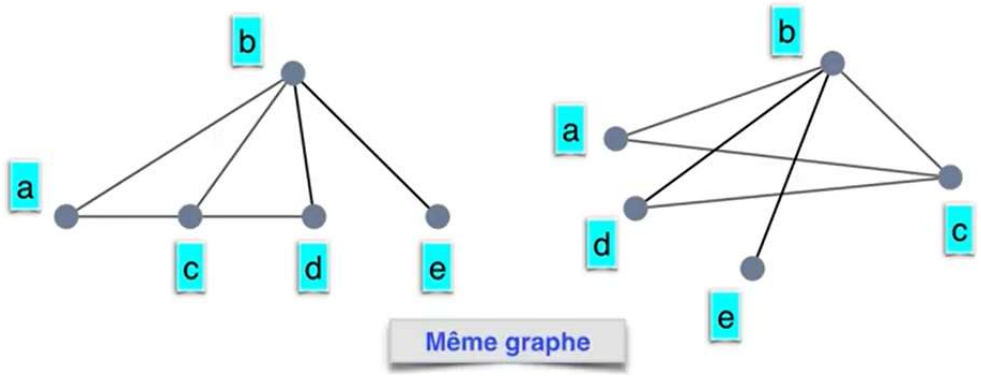


Figure 21: Graphe planaire exemple 1

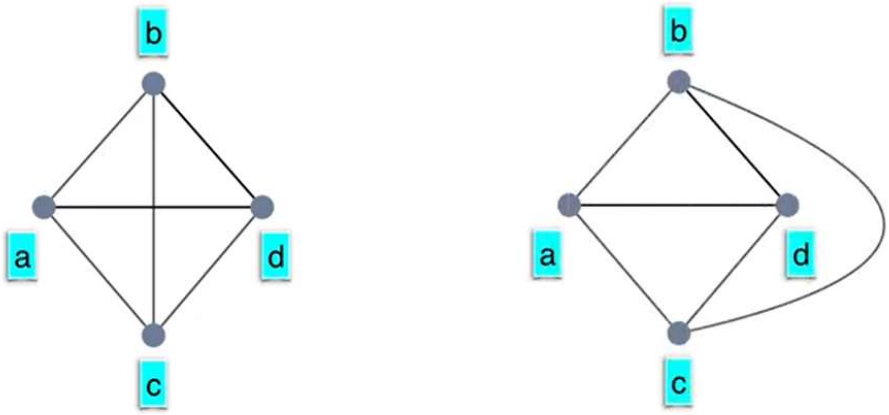


Figure 22: Graphe planaire exemple 2

Caractérisation des graphes planaires : Les deux graphes non planaires les plus simples sont K_5 et $K_{3,3}$:

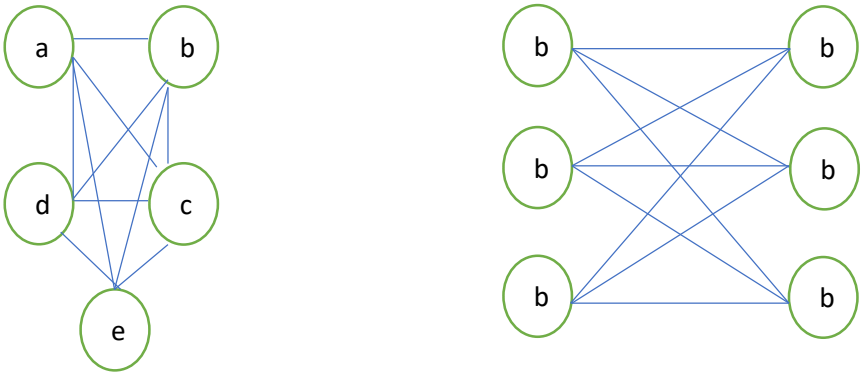


Figure 23: Graphes K5 et K3,3

2.2. Formule d'Euler

Soit G un graphe planaire connexe possédant n sommets, m arêtes et f faces, on a : n-m+f=2.

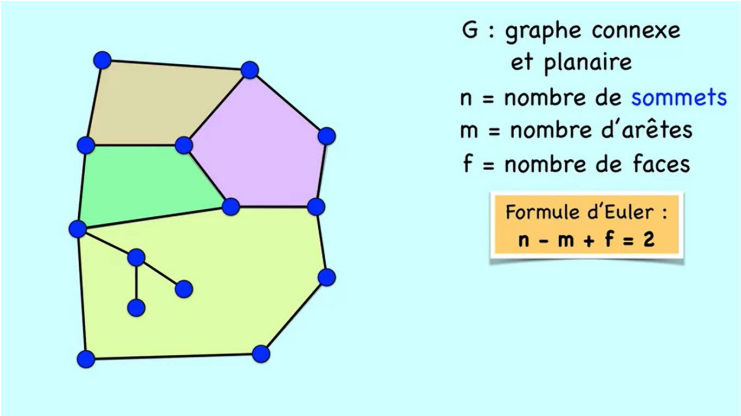


Figure 24: Formule d'Euler

Par exemple pour le graphe précèdent en appliquant le formule d'Euler, on trouve la valeur 2

F=5, N=14, M=17 ; 14-17+5= 2

2.3. Théorème de Kuratowski (1930)

Un graphe est planaire si et seulement s'il ne possède aucun mineur isomorphe à K_5 et $K_{3,3}$. (Un mineur d'un graphe est obtenu par une succession d'opérations de suppression et de "fusion" de sommets). Autrement dit, tout graphe non planaire contient une "copie" d'au moins un de ces deux graphes.

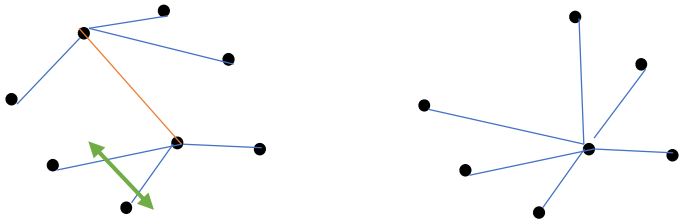


Figure 25: Mineur d'un graphe

2.4. Graphe Dual

Faces d'un graphe planaire : Etant donnée une représentation planaire d'un graphe G, le plan se retrouve divisé en un certain nombre de régions qu'on appelle les faces de la représentation planaire.

Par exemple, le graphe suivant possède 4 faces (notées A, B, C et D). On dira que les arêtes (1, 2), (1, 4), (4, 3), (3, 2), (5, 6) et (5, 7) constituent des frontières entre des faces différentes, tandis que l'arête (5, 1) constitue un isthme.

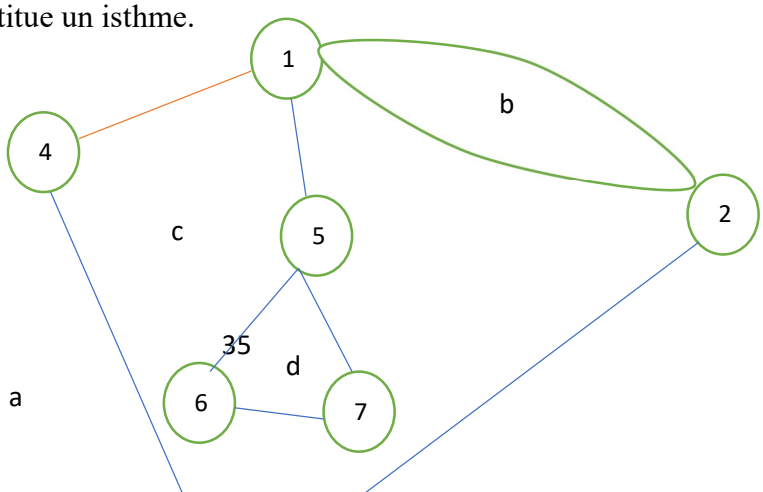


Figure 26: Les faces d'un graphe

Graphe Dual : On appelle dual d'un graphe planaire —appelé primal—le graphe obtenu de la façon suivante :

- dans toute face du primal on dessine un sommet du dual,

- pour toute arête séparant deux faces du primal, on dessine une arête joignant les deux sommets correspondants du dual (et qui traverse l'arête correspondante du primal).

Remarquons que cette relation est symétrique : si G2 est le dual de G1, alors G1 est le dual de G2.

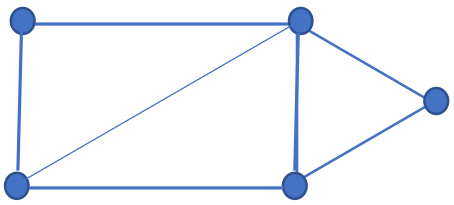


Figure 27: Graphe primal

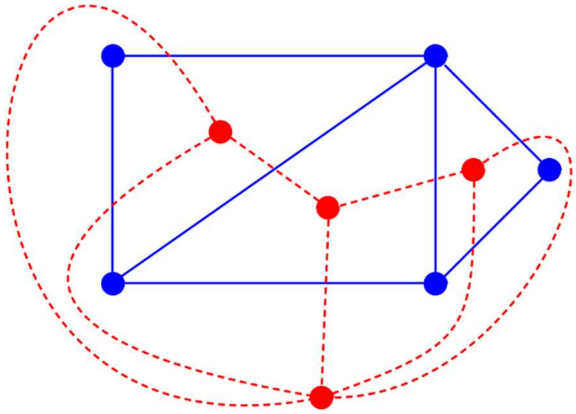


Figure 28: Graphe dual

3. Arbre, forêt et arborescence

3.1. Définitions

Les arbres et les arborescences sont des graphes particuliers très souvent utilisés en informatique pour représenter des données.

Arbre :

Étant donné un graphe G non orienté G=(S,A). On peut dire que G est un arbre s'il existe une et une seule chaîne reliant chaque deux sommets distincts de S. Donc, G est connexe ne contient pas de cycle. Dans un arbre, une feuille est un sommet de degré 1.

Exemple :

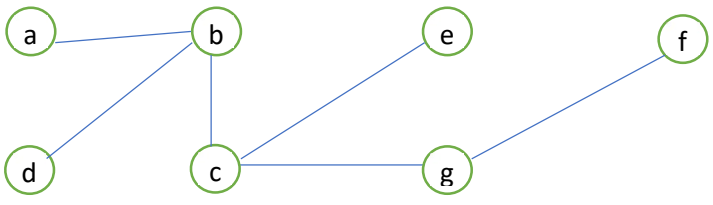


Figure 29: Un arbre

Arborescence :

Étant donné un graphe orienté G=(S,A). On peut dire que G représente une arborescence s'il a qu'un seul point sans arcs entrants nommée racine de l'arborescence. Il existe un seul chemin entre la racine à chaque sommet du graphe.

Exemple :

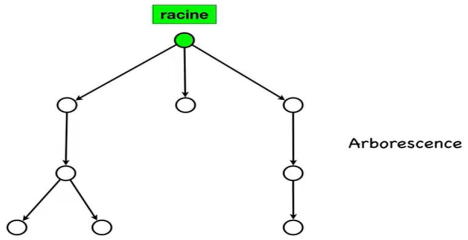


Figure 30 : Arborescence

Forêt :

Une forêt est un graphe qui contient plusieurs arbres ou arborescences distinctes.

3.2. Propriétés

Etant donné un graphe non orienté comportant n sommets, les propriétés suivantes sont équivalentes pour caractériser un arbre :

1. G est connexe et sans cycle,
2. G est sans cycle et possède n - 1 arêtes,
3. G est connexe et admet n - 1 arêtes,
4. G est sans cycle, et en ajoutant une arête, on crée un et un seul cycle élémentaire,
5. G est connexe, et en supprimant une arête quelconque, il n'est plus connexe,
6. Il existe une chaine et une seule entre 2 sommets quelconques de G.

Par exemple, le graphe suivant est un arbre :

Une arborescence est un graphe orienté sans circuit admettant une racine $s_0 \in S$ telle que, pour tout autre sommet $s_i \in S$, il existe un chemin unique allant de s_0 vers s_i. Si l'arborescence comporte n sommets, alors elle comporte exactement n - 1 arcs.

3.3. Arbre maximal (ou couvrant).

On étudie dans la suite les deux principales stratégies d'exploration : le parcours en largeur et le parcours en profondeur.

Dans les deux cas, l'algorithme procède par coloriage des sommets :

– Initialement, tous les sommets sont blancs. Lorsqu'un sommet est "découvert", il est colorié en gris. Le sommet reste gris tant qu'il reste des successeurs de ce sommet qui sont blancs. Un sommet est colorié en noir lorsque tous ses successeurs sont gris ou noirs.

3.3.1. Arborescence couvrante

On parcourt un graphe à partir d'un sommet donné s_0. Cette arborescence contient un arc (s_i, s_j) si et seulement si le sommet s_j a été découvert à partir du sommet s_i.

L'arborescence associée à un parcours de graphe sera mémorisée dans un tableau π tel que $\pi[s_j] = s_i$. Si s_j a été découvert à partir de si, et $\pi[s_k]$ = nil si s_k est la racine, ou s'il n'existe pas de chemin de la racine vers s_k.

3.3.2. Parcours en largeur (Breadth First Search = BFS)

Le parcours en largeur est obtenu en gérant la liste d'attente au coloriage comme une file d'attente (FIFO = First In First Out). Autrement dit, on enlève à chaque fois le plus vieux sommet gris dans la file d'attente, et on introduit tous les successeurs blancs de ce sommet dans la file d'attente, en les coloriant en gris.

Structures de données utilisées :

– On utilise une file F, pour laquelle on suppose définies les opérations init_file(F) qui initialise la file F à vide, ajoute_fin_file(F,s) qui ajoute le sommet s à la fin de la file F, est_vide(F) qui

retourne vrai si la file F est vide et faux sinon, et enleve_debut_file(F,s) qui enlève le sommet s au début de la file F.

– On utilise un tableau π qui associe à chaque sommet le sommet qui l'a fait entrer dans la file, et un tableau couleur qui associe à chaque sommet sa couleur (blanc, gris ou noir).

– On va en plus utiliser un tableau d qui associe à chaque sommet son niveau de profondeur par rapport au sommet de départ s0 (autrement dit, $d[s_i]$ est la longueur du chemin dans l'arborescence

π de la racine s_0 jusque s_i). Ce tableau sera utilisé plus tard.

Algorithme de parcours en largeur (BFS)(S, A, s_0)

```
init_file(F)
pour tout sommet si ∈ S faire
        π[si] ← nil
        d[si] ← 1
        couleur[si] ← blanc
fin pour
d[s0] ← 0
ajoute_fin_file(F, s0)
couleur[s0] ← gris
tant que est_vide(F) = faux faire
        enleve_debut_file(F, si)
        pour tout sj ∈ succ(si) faire
                si couleur[sj ] = blanc alors
                        ajoute_fin_file(F, sj)
                        couleur[sj ] ← gris
                        π[sj ] ← si
                        d[sj ] ← d[si] + 1
                finsi
```

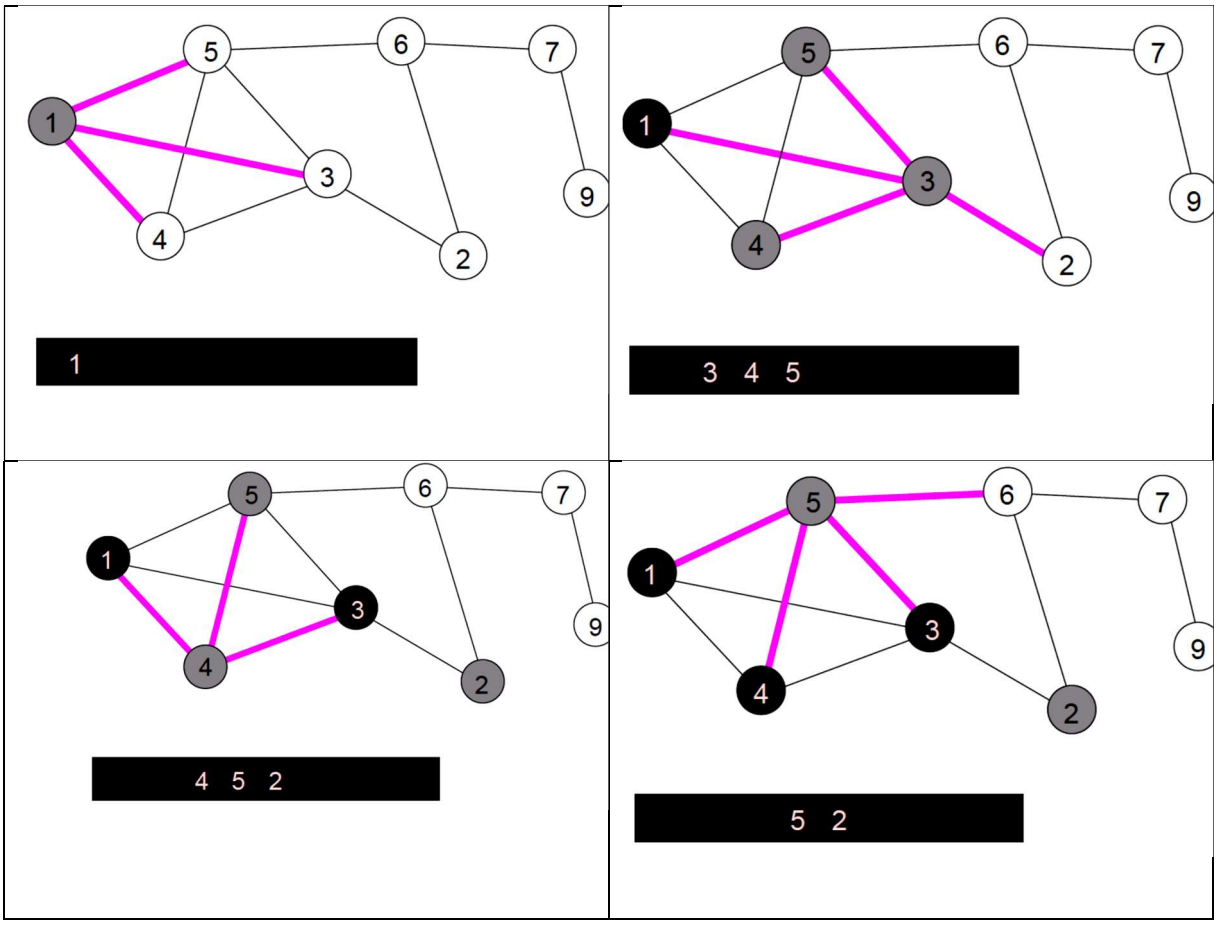
fin pour

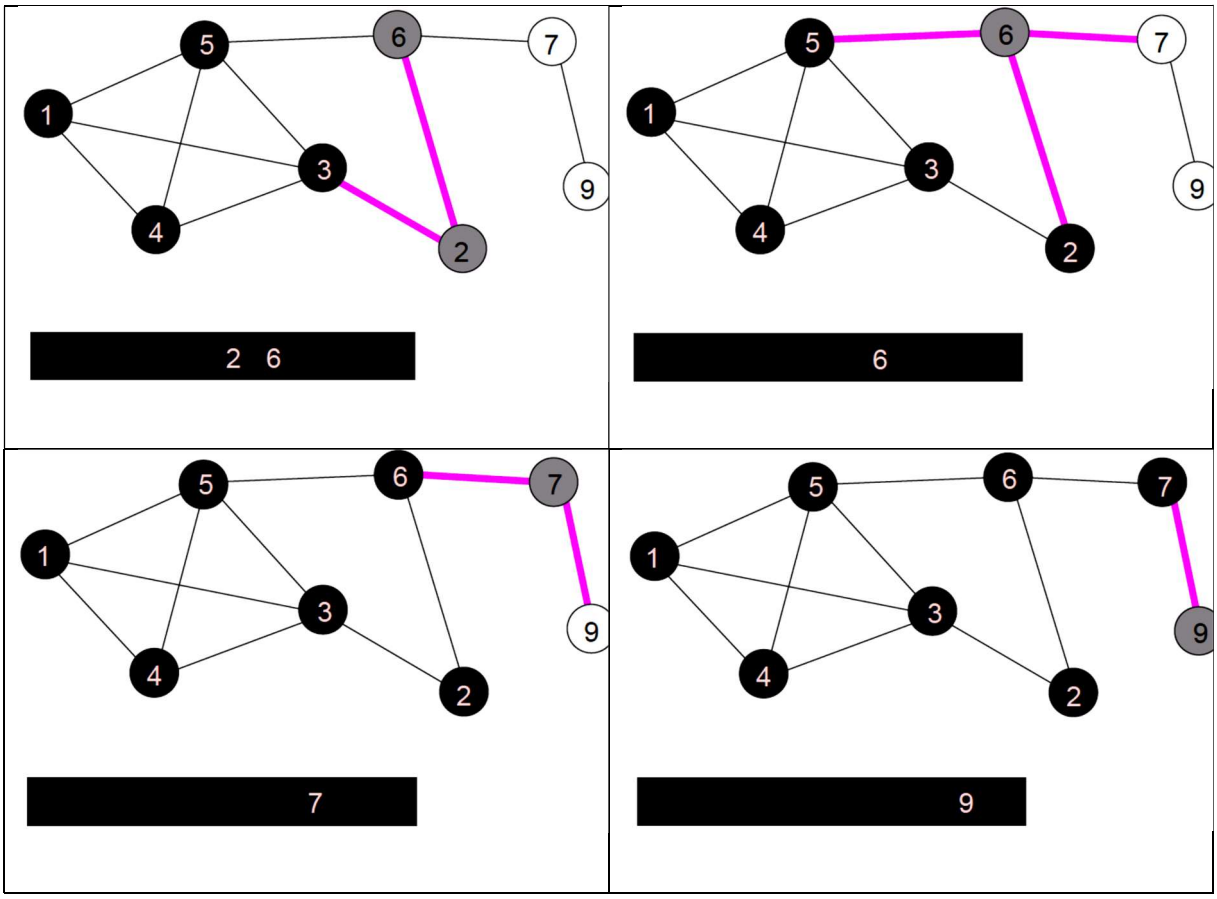
couleur[s_i] noir

fin tant

fin BFS

Exemple d'application de l'algorithme sur un graphe non orienté. L'application de l'algorithme pour un graphe orienté est la même, il faut tout simplement appliquer la notion de successeur c.à.d. suivre l'orientation des arecs.




1
5
6
7
3
9
4
2
2 6
1
5
6
7
3
9
4
2
6
1
5
6
7
3
9
4
2
7
1
5
6
7
3
9
4
2
9

Chapitre III. Flots

1. Définitions

Le problème du flot de coût est un problème algorithmique de théorie des graphes, Dans un réseau de transport de l'électricité, de données ou une autre matière doit s'écouler depuis un sommet appelé **source** (par exemple une usine de production) vers un autre sommet appelé **puits** (où la matière est consommée). Un réseau de transport sera défini par un quadruplet (G, c, s, p) tel que

- G = (S, A) est un graphe orienté,

- c : A → R^+ est une fonction qui associe à chaque arc sa capacité,

- s ∈ S est la source, et p ∈ S est le puits.

2. Recherche d'un flot maximum dans un réseau de transport

2.1. Définition

Etant donné un réseau de transport (G, c, s, p), il s'agit de trouver un flot f tel que |f| soit maximal.

Modélisation en programmation linéaire : Le problème du flot maximal peut être exprimé comme un problème de programmation linéaire, c'est-à-dire comme une fonction linéaire à maximiser tout en respectant un certain nombre de contraintes linéaires. Etant donné le réseau de transport (G = (S,A), c, s, p), il s'agit de résoudre le problème linéaire suivant :

Maximiser $\sum_{s_i \in S} f(s, s_i)$

Telque $f(s_i, s_j) \leq c(s_i, s_j) \quad \forall (s_i, s_j) \in A$

$f(s_i, s_j) = -f(s_j, s_i) \qquad \forall (s_i, s_j) \in S^2$

$\sum_{s_j \in S} f(s_i, s_j) = 0 \quad \forall s_i \in S$

2.2. Théorème de Ford-Fulkerson

– Au départ, le flot est nul, c'est-à-dire que $f(s_i, s_j) = 0$ pour tout couple de sommets $(s_i, s_j) \in S^2$.

– On augmente ensuite itérativement le flot f en cherchant à chaque fois un "chemin améliorant", c'est-à-dire un chemin allant de la source s jusqu'au puits p et ne passant que par des arcs dont le flot actuel est inférieur à la capacité. Pour cela, à chaque itération, on calcule la "capacité résiduelle" de chaque arc, c'est-à-dire la quantité de flot pouvant encore passer.

2.3. Algorithme de Ford-Fulkerson

fonc Ford-Fulkerson(G = (S,A), c : $S \times S \rightarrow R^+$, $s \in S$, $p \in S$)

retourne un flot maximal

pour chaque $(s_i, s_j) \in S^2$ faire

$f(s_i, s_j) \leftarrow 0$

$c_f(s_i, s_j) \leftarrow c(s_i, s_j)$

fin pour

tant que il existe un chemin améliorant ch dans le graphe résiduel faire

/* calcul de la capacité résiduelle du chemin améliorant */

$c_f(ch) \leftarrow \min_{(si, sj) \in ch}(c_f(s_i, s_j))$

/* mise à jour du flot et de la capacité résiduelle le long des arcs de ch */

pour tout arc (s_i, s_j) du chemin améliorant ch faire

$f(s_i, s_j) \leftarrow f(s_i, s_j) + c_f(ch)$

$f(s_j, s_i) \leftarrow f(s_j, s_i) - c_f(ch)$

$c_f(s_i, s_j) \leftarrow c_f(s_i, s_j) - c_f(ch)$

$c_f(s_j, s_i) \leftarrow c_f(s_j, s_i) + c_f(ch)$

fin pour

fin tant

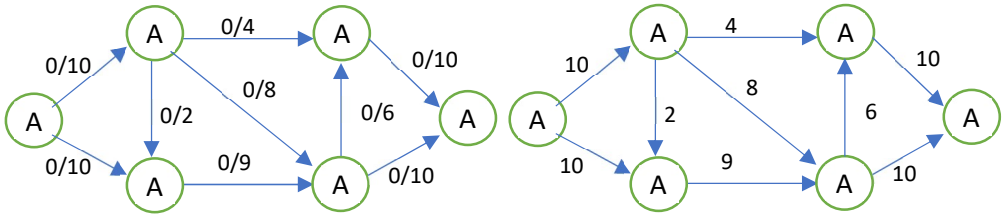
retourne(f)

fin Ford-Fulkerson

Une coupe est un ensemble de sommets contenant s et ne contenant pas p. La capacité d'une coupe est la somme des capacités des arcs sortant de cette coupe.

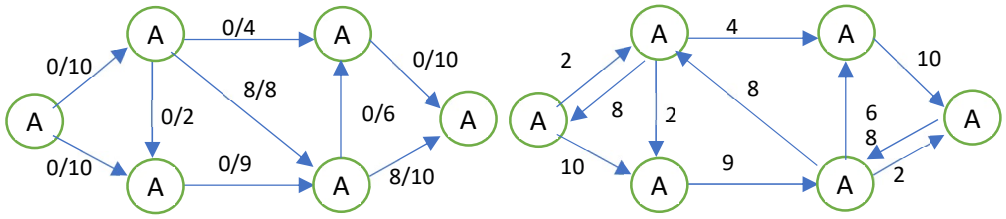
Exemple. Déterminer le flot maximal à partir du graphe de flot ci-dessous

Étape 1: mettre le flux de chaque arc sur 0

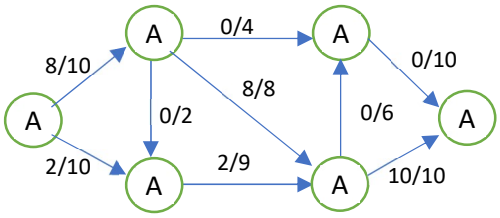


Maintenant, trouvez un chemin améliorant dans le réseau résiduel. Ici, on sélectionne le chemin

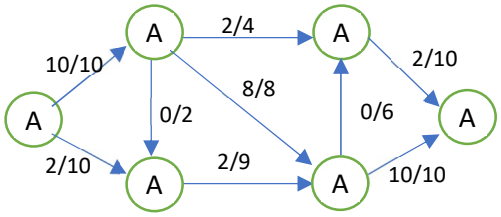
s -> A -> D -> t. mettre à jour les valeurs de flux des arêtes



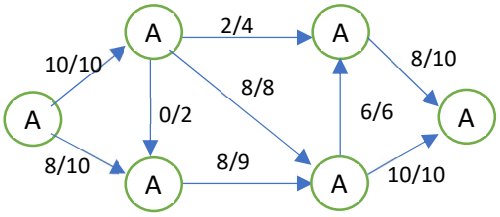
sélectionner le chemin améliorant s -> C -> D -> t. Désormais, la capacité résiduelle est de 2.



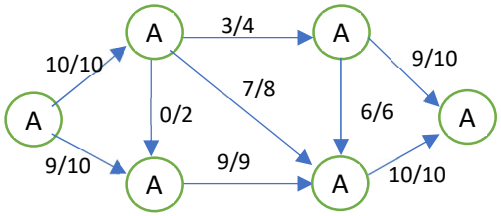
sélectionner le chemin améliorant s -> A -> B->t, Désormais, la capacité résiduelle est de 2.



sélectionner le chemin améliorant s -> C -> D -> B->t. Désormais, la capacité résiduelle est de 6.



sélectionner le chemin améliorant s -> C -> D -> A-> B->t. Désormais, la capacité résiduelle est de 1.



Il n'existe un chemin améliorant ch dans le graphe résiduel. Cela signifie que la méthode Ford Fulkerson est terminée. Le débit maximum étant égal au débit sortant de la source, dans cet exemple, le débit maximum est de 10 + 9 = 19.

3. Recherche d'un flot compatible

Algorithme de recherche d'un flot compatible :

(I) Partir du flot f nul.

(II) Chercher un arc u tel que fu < b(u)

- si un tel arc n'existe pas alors FIN 1;

- sinon poser s = extrémité terminale de u et p = extrémité initiale de u.

(III) Chercher une chaîne de s à p dont les arcs dans le sens + sont insaturés et les arcs dans le sens - ont un flux strictement supérieur à la borne ; si ce n'est pas possible alors FIN 2.

(IV) Utiliser cette chaîne pour améliorer le flot en cherchant à satisfaire la contrainte de borne sur l'arc u et retourner en (ii).

Chapitre IV. Problèmes de cheminement

1. Recherche des composantes connexes

1.1. Présentation des objectifs

L'algorithme de Tarjan permet de déterminer les composantes fortement connexes d'un graphe orienté. L'algorithme prend en entrée un graphe orienté et renvoie une partition des sommets du graphe correspondant à ses composantes fortement connexes. Le principe de l'algorithme est le suivant : on lance un parcours en profondeur depuis un sommet arbitraire. Les sommets explorés sont placés sur une pile P. Un marquage spécifique permet de distinguer certains sommets : les racines des composantes fortement connexes, c'est-à-dire les premiers sommets explorés de chaque composante (ces racines dépendent de l'ordre dans lequel on fait le parcours, elles ne sont pas fixées de façon absolue sur le graphe). Lorsqu'on termine l'exploration d'un sommet racine v, on retire de la pile tous les sommets jusqu'à v inclus. L'ensemble des sommets retirés forme une composante fortement connexe du graphe. S'il reste des sommets non atteints à la fin du parcours, on recommence à partir de l'un d'entre eux.

Parcours en profondeur (Depth First Search = DFS)

Le parcours en profondeur est obtenu en gérant la liste d'attente au coloriage en noir comme une pile (LIFO = Last In First Out). Autrement dit, on considère à chaque fois le dernier sommet gris entré dans la pile, et on introduit devant lui tous ses successeurs blancs.

Structures de données utilisées :

– On utilise une pile P, pour laquelle on suppose définies les opérations init_pile(P) qui initialise la pile P à vide, empile(P,s) qui ajoute s au sommet de la pile P, est_vide(P) qui retourne vrai si la pile P est vide et faux sinon, sommet(P) qui retourne le sommet s au sommet de la pile P, et depile(P,s) qui enlève s du sommet de la pile P.

– On utilise, comme pour le parcours en largeur, un tableau π qui associe à chaque sommet le sommet qui l'a fait entrer dans la pile, et un tableau couleur qui associe à chaque sommet sa couleur (blanc, gris ou noir).

– On va en plus mémoriser pour chaque sommet s_i :

- dec[s_i] = date de découverte de s_i (passage en gris)

- fin[s_i] = date de fin de traitement de s_i (passage en noir) où l'unité de temps est une itération.

La date courante est mémorisée dans la variable tps.

Algorithme de parcours en profondeur des sommets accessibles depuis s_0 DFSinit_pile(P)

Pour tout sommet $s_i \in S$ faire

    π[s_i] ← nil

    couleur[s_i] ←blanc

fin pour

tps ← 0

dec[s0] ← tps

empile(P, s_0)

couleur[s_0] ← gris

tant que est_vide(P) = faux faire

    tps ← tps + 1

    si ← sommet(P)

    si ∃ $s_j \in$ succ(s_i) tel que couleur[s_j]=blanc alors

        empile(P, s_j)

        couleur[s_j] ← gris

        π[s_j] ← s_i

        dec[s_j] ← tps

    sinon /* tous les successeurs de si sont gris ou noirs */

        depile(P, s_i)

        couleur[s_i] ← noir

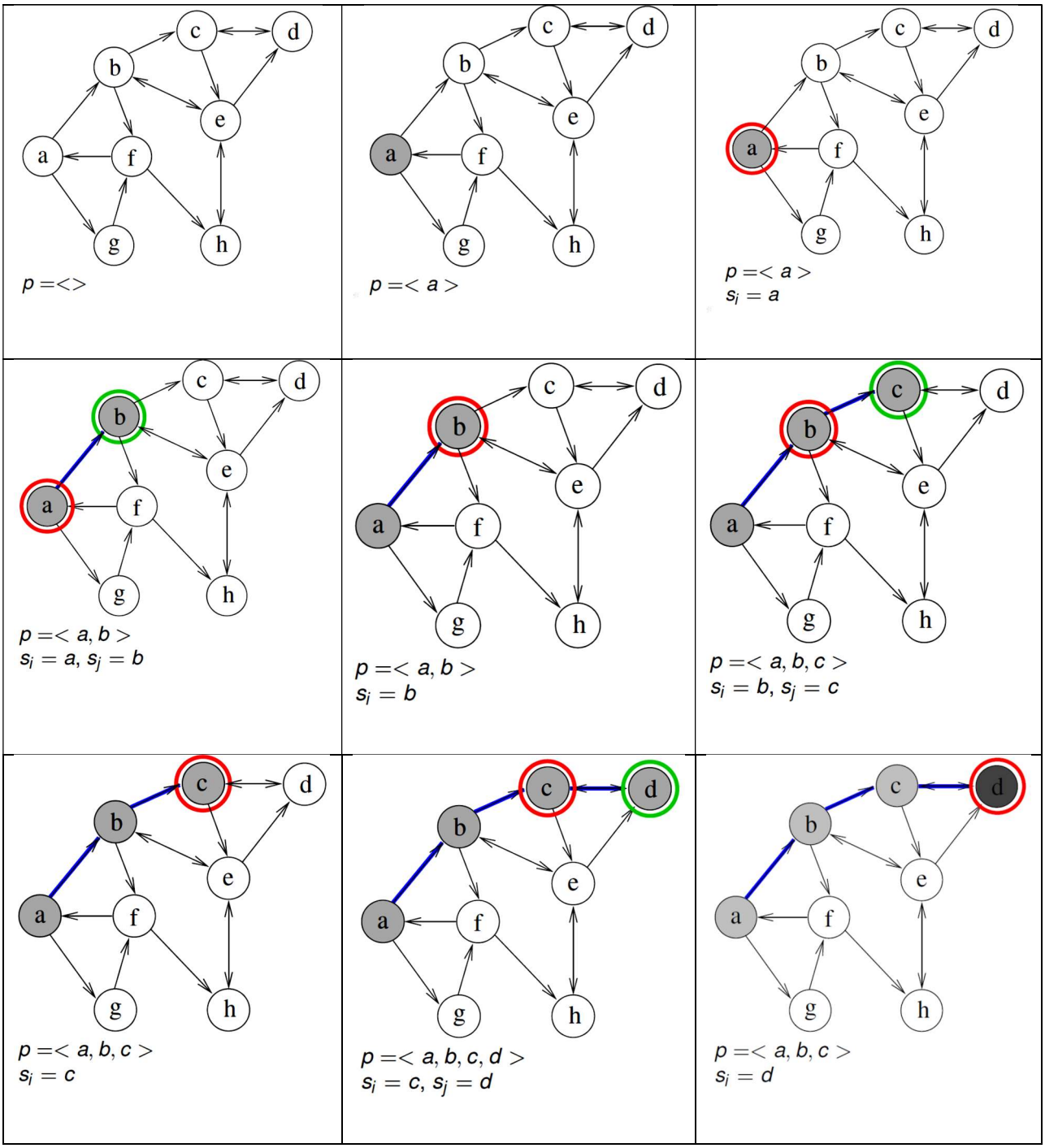
fin[s_i] ← tps

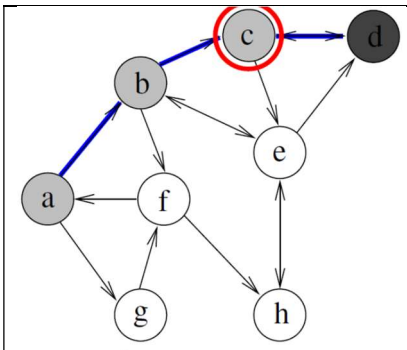
finsi

fin tant

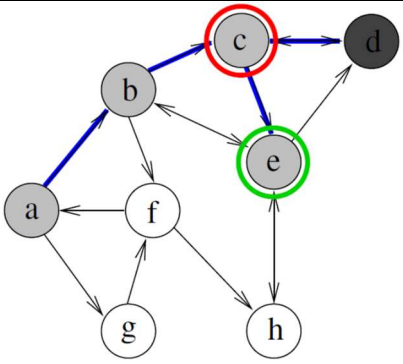
fin DFS

EXEMPLE

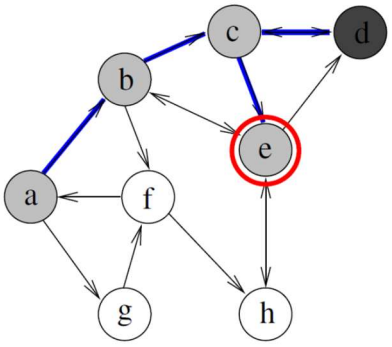




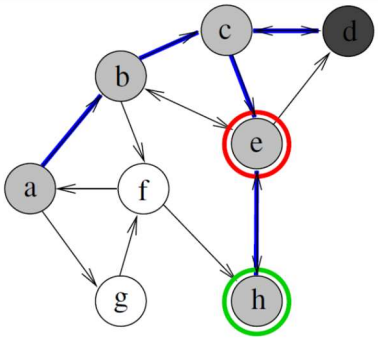
$p =< a, b, c >$
$s_i = c$



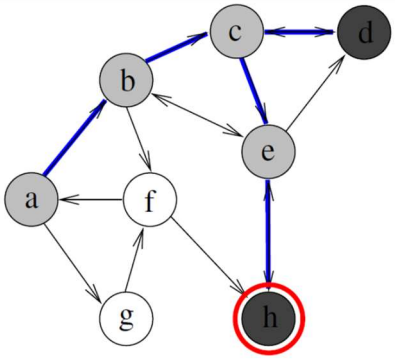
$p =< a, b, c, e >$
$s_i = c, s_j = e$



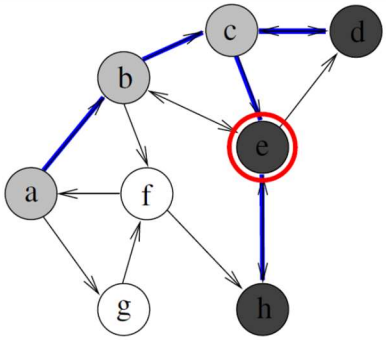
$p =< a, b, c, e >$
$s_i = e$



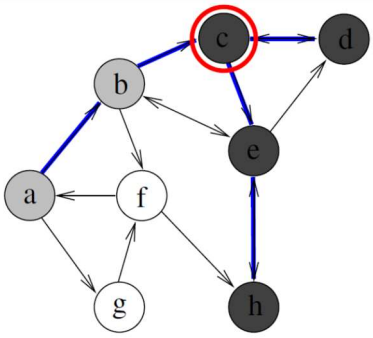
$p =< a, b, c, e, h >$
$s_i = e, s_j = h$



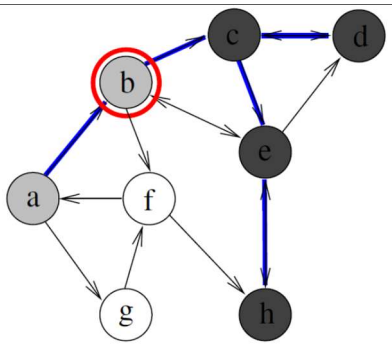
$p =< a, b, c, e >$
$s_i = h$



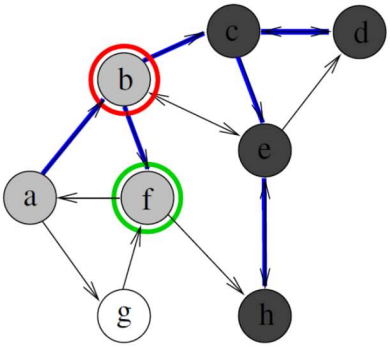
$p =< a, b, c >$
$s_i = e$



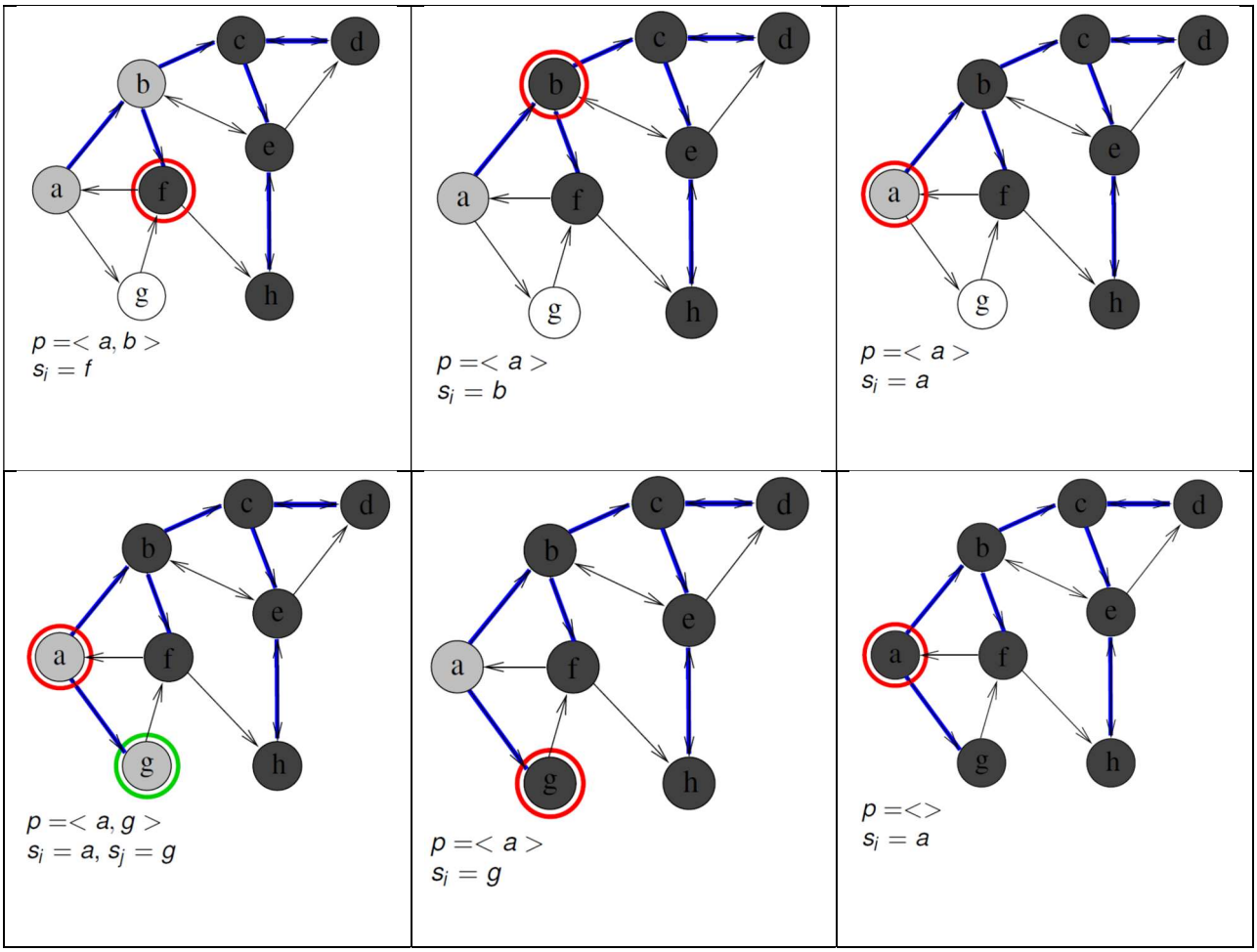
$p =< a, b >$
$s_i = c$



$p =< a, b >$
$s_i = b$



$p =< a, b, f >$
$s_i = b, s_j = f$



1.2. Algorithme de Trémeaux-Tarjan

Les sommets sont numérotés dans l'ordre où ils sont explorés. Le numéro d'un sommet est noté v.num. Les arêtes empruntées par le parcours en profondeur forment un arbre. Dans ce contexte, on peut définir le sous-arbre associé à tout sommet v. Au cours de l'exploration de ce sous-arbre, on calcule une seconde valeur v.numAccessible. Elle est initialisée à v.num et décroît lors du parcours des successeurs de v. Lorsque le parcours de v se termine, v.numAccessible correspond au numéro du plus petit sommet situé soit dans le sous-arbre de v, soit successeur direct appartenant à P d'un sommet de ce sous-arbre. Deux cas sont possibles : v est une racine. Alors tous les sommets accessibles depuis v sont dans le sous-arbre de v (à l'exception éventuelle de ceux explorés lors d'un parcours antérieur). On a v.numAccessible = v.num ; v

n'est pas une racine. Alors il existe un sommet accessible depuis v qui est dans P mais pas dans le sous-arbre de v. On a alors v.numAccessible < v.num.

Algorithme de Tarjan

```
fonction tarjan(graphe G)
  num := 0
  P := pile vide
  partition := ensemble vide

  fonction parcours(sommet v)
    v.num := num
    v.numAccessible := num
    num := num + 1
    P.push(v), v.dansP := oui

    // Parcours récursif
    pour chaque w successeur de v
      si w.num n'est pas défini
        parcours(w)
        v.numAccessible := min(v.numAccessible, w.numAccessible)
      sinon si w.dansP = oui
        v.numAccessible := min(v.numAccessible, w.num)

    si v.numAccessible = v.num
      // v est une racine, on calcule la composante fortement connexe associée
      C := ensemble vide
      répéter
        w := P.pop(), w.dansP := non
```

```
        ajouter w à C
      tant que w différent de v
      ajouter C à partition
  fin de fonction

  pour chaque sommet v de G
    si v.num n'est pas défini
      parcours(v)

  renvoyer partition
 fin de fonction
```

2. Recherche du plus court chemin

2.1. Présentation des conditions

Soit G = (S,A) un 1-graphe orienté valué tel que la fonction cout : A →R associe à chaque arc (s_i, s_j) de A un coût réel $cout(s_i, s_j)$.

Le coût d'un chemin $p =< s_0, s_1, s_2, \dots, s_k >$ est égal à la somme des coûts des arcs composant le chemin, c'est à dire,

$$cout\ (p) = \sum_{i=1}^{k} cout\ (s_{i-1}, s_i)$$

Le coût d'un chemin sera aussi appelé poid du chemin.

Le coût (ou poid) d'un plus court chemin entre deux sommets s_i et s_j est noté $\delta(s_i, s_j)$ et est défini par

$\delta(s_i, s_j) = +\infty$, s'il n'existe pas de chemin entre s_i et s_j

$\delta(s_i, s_j) = \min\{cout(p)/p =$ chemin de s_i à $s_j\}$, s'il existe un chemin entre s_i et s_j

Exemple

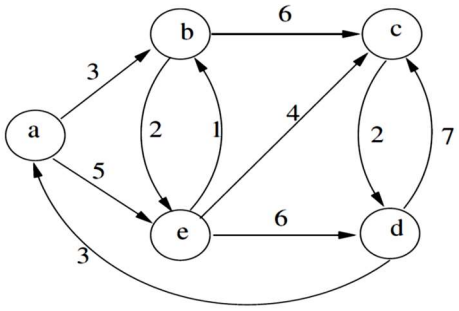


Figure 31: Le plus court chemin

Conditions d'existence d'un plus court chemin : s'il existe un chemin entre deux sommets u et v contenant un circuit de coût négatif, alors $\delta(u, v) = -\infty$, et il n'existe pas de plus court chemin entre u et v. Un circuit négatif est appelé un circuit absorbant.

Exemple

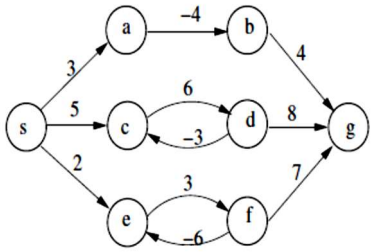


Figure 32: Circuit absorbant

Le chemin < s, e, f, e, f, g > contient le circuit < e, f, e > de coût négatif -3. Autrement dit, à chaque fois que l'on passe dans ce circuit, on diminue de 3 le coût total du chemin. Par conséquent, $\delta(s, g) = -\infty$ et il n'existe pas de plus court chemin entre s et g.

Définition du problème des plus courts chemins à origine unique : Etant donné un graphe orienté G = (S,A), une fonction cout : A → R et un sommet origine $s_0 \in S$, on souhaite calculer pour chaque sommet $s_j \in S$ le coût $\delta(s_0, s_j)$ du plus court chemin de s_0 à s_j . On supposera que le graphe G ne comporte pas de circuit absorbant.

Arborescence des plus courts chemins : l'arborescences couvrante calculée lors d'un parcours d'un graphe est mémorisée dans un tableau π tel que π[s0] = nil et $\pi[s_j] = s_i$ si $s_i \rightarrow s_j$ est un arc de l'arborescence. Pour connaitre le plus court chemin entre s_0 et un sommet s_k donné, il faudra alors "remonter" de s_k jusque s_0 en utilisant π.

Remarque : $\delta(s_0, s_i) = \delta(s_0, \pi[s_i]) + cout(\pi[s_i], s_i)$.

Considérons par exemple le graphe valué orienté suivant :

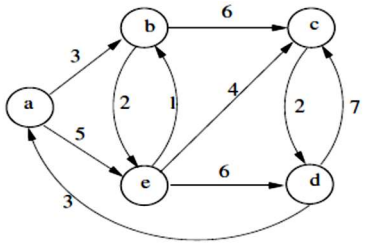


Figure 33: Arborescences des plus courts chemins 1

Ce graphe possède plusieurs arborescences des plus courts chemins dont l'origine est a, par

Exemple

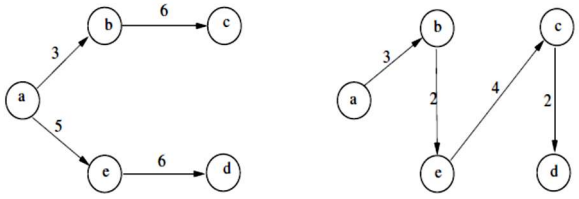


Figure 34: Arborescences des plus courts chemins 2

La première de ces 2 arborescences est représentée par le tableau π tel que π[a] = nil, π[b] = a, π[c] = b, π[d] = e et π[e] = a.

2.2. Algorithme de Moore-Dijkstra

On maintient 2 ensembles disjoints E et F tels que $E \cup F = S$. L'ensemble E contient chaque sommet si pour lequel on connait un plus court chemin depuis s0 (c'est-à-dire pour lequel d[si] = δ(s0, si)). L'ensemble F contient tous les autres sommets. A chaque itération de l'algorithme,

on choisit le sommet si dans F pour lequel la valeur d[si] est minimale, on le rajoute dans E, et on relâche tous les arcs partant de ce sommet s_i.

fonc Dijkstra(G = (S,A), cout : A→R^+, $s_0 \in S$)

retourne une arborescence des plus courts chemins d'origine s

pour chaque sommet $s_i \in S$ faire

$d[s_i] \leftarrow +\infty$

$\pi[s_i] \leftarrow$ nil

fin pour

$d[s_0] \leftarrow 0$

$E \leftarrow \varnothing$

$F \leftarrow S$

tant que $F \neq \varnothing$, faire

soit s_i le sommet de F tel que $d[s_i]$ soit minimal

/* $d[s_i] = \delta(s_0, s_i)$ */

$F \leftarrow F - \{s_i\}$

$E \leftarrow E \cup \{s_i\}$

pour tout sommet $s_j \in succ(s_i)$ faire : relacher(s_i, s_j)

fin tant

retourner(π)

fin Dijkstra

Exemple : les plus courts chemins à partir de A

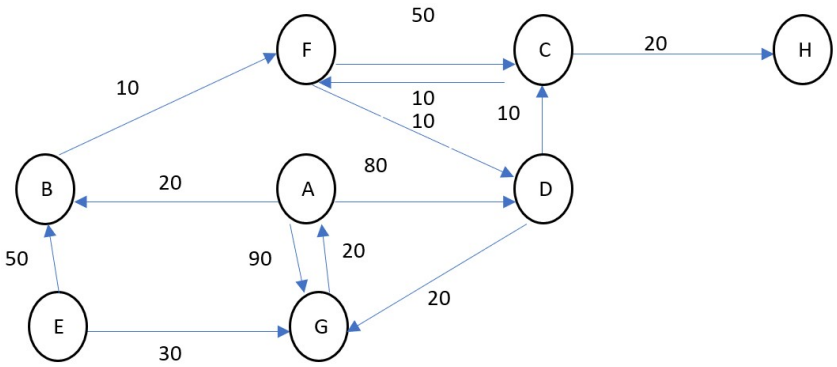


Figure 35: Les plus courts chemins à origine unique

Itération		s_i	E	B	C	D	E	F	G	H
1	d	A	∅	20	∞	80	∞	∞	90	∞
	pred			A		A			A	
1	d	B	{A}	20	∞	80	∞	30	90	∞
	pred			A		A		B	A	
2	d	F	{A,B}	20	40	70	∞	30	90	∞
	pred			A	F	F		B	A	
3	d	C	{A,B,F}	20	40	50	∞	30	90	60
	pred			A	F	C		B	A	C
4	d	D	{A,B,F,C}	20	40	50	∞	30	70	60
	pred			A	F	C		B	D	C
4	d	H	{A,B,F,C,H}	20	40	50	∞	30	70	60
	pred			A	F	C		B	D	C
4	d	G	{A,B,F,C,H,G}	20	40	50	∞	30	70	60
	pred			A	F	C		B	D	C

2.3. Algorithme de Bellman-Ford

L'algorithme de Bellman-Ford permet de trouver les plus courts chemins à origine unique dans le cas où le graphe contient des arcs dont le coût est négatif, sous réserve que le graphe ne contienne pas de circuit absorbant (dans ce cas, l'algorithme de Bellman-Ford va détecter l'existence de circuits absorbants).

Principe : on associe à chaque sommet s_i une valeur $d[s_i]$ qui représente une borne maximale du coût du plus court chemin entre s_0 et s_i. L'algorithme diminue alors progressivement les valeurs $d[s_i]$ en relâchant les arcs. Chaque arc va être relâché plusieurs fois : on relâche une première fois tous les arcs, après quoi, tous les plus courts chemins de longueur 1, partant de s_0, auront été trouvés. On relâche alors une deuxième fois tous les arcs, après quoi tous les plus courts chemins de longueur 2, partant de s_0, auront été trouvés... et ainsi de suite... Après la $k^{ième}$ série de relâchement des arcs, tous les plus courts chemins de longueur k, partant de s_0, auront été trouvés. Si le graphe contient un circuit absorbant, au bout de n - 1 passages, on aura encore au moins un arc (s_i, s_j) pour lequel un relâchement permettrait de diminuer la valeur de $d[s_j]$. L'algorithme utilise cette propriété pour détecter la présence de circuits absorbants.

fonc Bellman-Ford(G = (S,A), cout : S → R, $s_0 \in S$)

retourne une arborescence des plus courts chemins d'origine s_0

pour chaque sommet $s_i \in S$ faire

$d[s_i] \leftarrow +\infty$

$\pi[s_i] \leftarrow$ nil

fin pour

$d[s_0] \leftarrow 0$

pour k variant de 1 à |S|-1 faire

pour chaque arc $(s_i, s_j) \in A$ faire relacher(s_i, s_j) fin pour

fin pour

pour chaque arc $(s_i, s_j) \in A$ faire

si d[s_j] > d[s_i] + cout(s_i, s_j) alors afficher("circuit absorbant") finsi

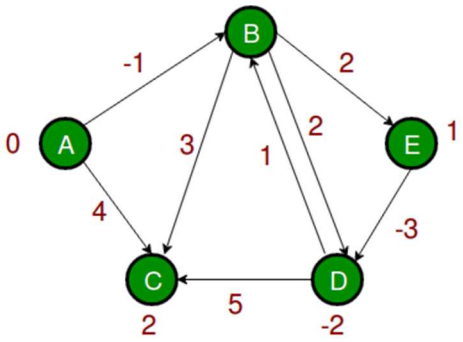
fin pour

retourner(_)

fin Bellman-ford

Exemple :

A	B	C	D	E
0	∞	∞	∞	∞
0	-1	∞	∞	∞
0	-1	4	∞	∞
0	-1	2	∞	∞
0	-1	2	∞	1
0	-1	2	1	1
0	-1	2	-2	1



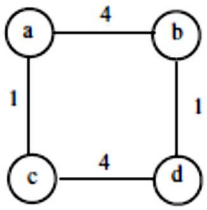
3. Recherche d'un arbre de poids extrémum

3.1. Présentation des objectifs

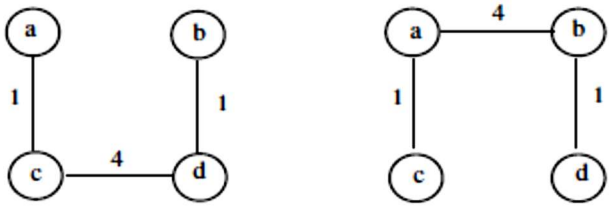
Pour définir le réseau câblé de telle sorte que la longueur totale de câble soit minimale et qu'un certain nombre de lieux soient desservis. On peut modéliser ce problème de câblage à l'aide d'un graphe non orienté connexe G = (S,A), où S associe un sommet à chaque lieu devant être desservi, et A contient une arête pour chaque portion de route entre 2 lieux. Ce graphe est valué par une fonction coût qui spécifie pour chaque (s_i, s_j) la longueur de câble nécessaire pour connecter s_i à s_j. Il s'agit alors de trouver un arbre qui recouvre l'ensemble des sommets du graphe. Ce graphe est appelé arbre couvrant. On cherche à minimiser le poids total des arêtes de l'arbre. On dira qu'on cherche l'arbre couvrant minimal, abrégé par ACM.

De façon plus formelle, un ACM d'un graphe G = (S,A) est un graphe partiel G_0 = (S,A_0) de G tel que G_0 est connexe et sans cycle (G_0 est un arbre), et la somme des coûts des arêtes de A0 est minimale.

Remarque : il peut exister plusieurs ACM, de même coût, associés à un même graphe.



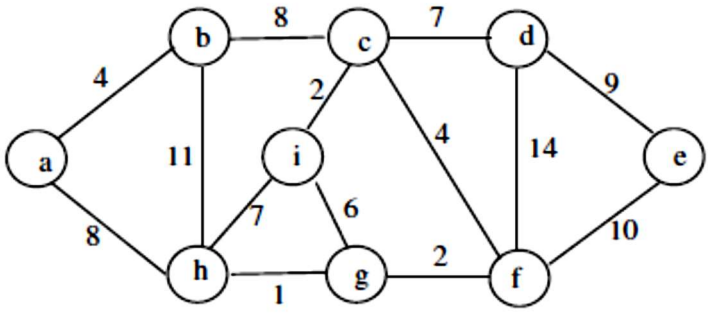
Par exemple, le graphe possède les 2 ACMs suivants :



3.2. Algorithme de Kruskal 1956

Principe de l'algorithme de Kruskal : On commence par trier l'ensemble des arêtes du graphe par ordre de coût croissant. On va sélectionner de proche en proche les arêtes devant faire partie de l'ACM. Au début, cet ensemble est vide. On considère ensuite chacune des arêtes du graphe selon l'ordre que l'on vient d'établir (de l'arête de plus faible coût jusqu'à l'arête de plus fort coût). A chaque fois, si l'arête que l'on est en train de considérer peut-être ajoutée à l'ensemble des arêtes déjà sélectionnées pour l'ACM sans générer de cycle, alors on la sélectionne, sinon on l'abandonne.

Considérons par exemple le graphe suivant :



On trie les arêtes du graphe. On obtient l'ordre suivant :

hg < ic = gf < ab = cf < ig < hi = cd < bc = ah < de < fe < bh < df

On ajoute alors successivement dans l'ACM les arêtes : hg, ic, gf, ab, cf, cd, bc, de

Algorithme de Kruskal

fonc Kruskal(G = (S,A), cout : A → R)

retourne un ACM $G_0 = (S,K)$

pour chaque sommet $s_i \in S$ faire $\pi[s_i] \leftarrow$ nil

trier les arêtes de A par ordre de coût croissant

$K \leftarrow \varnothing$,

tant que $|K| < |S| - 1$ faire

soit (s_i, s_j) la k^{ieme} plus petite arête de A

/* recherche de la racine r_i de la composante connexe de s_i */

$r_i \leftarrow s_i$

tant que $\pi[r_i] \neq$ nil faire $r_i \leftarrow \pi[r_i]$

/* recherche de la racine r_j de la composante connexe de s_j */

$r_j \leftarrow s_j$

tant que $\pi[r_j] \neq$ nil faire $r_j \leftarrow \pi[r_j]$

si $r_i \neq r_j$ alors

/* on ajoute (s_i, s_j) à l'ACM */

$K \leftarrow K \cup \{(s_i, s_j)\}$

/* on fusionne les deux composantes connexes */

$\pi[r_i] \leftarrow r_j$

fin si

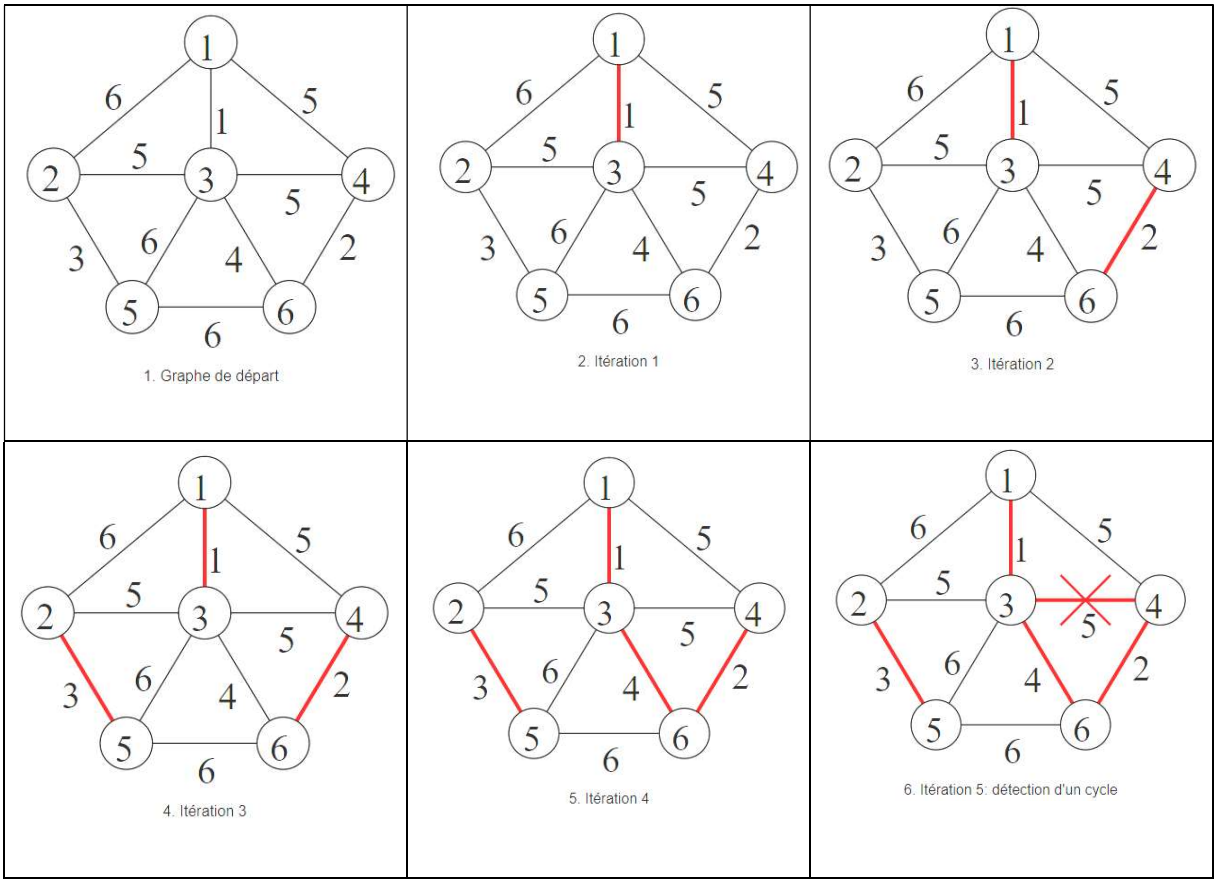
fin pour

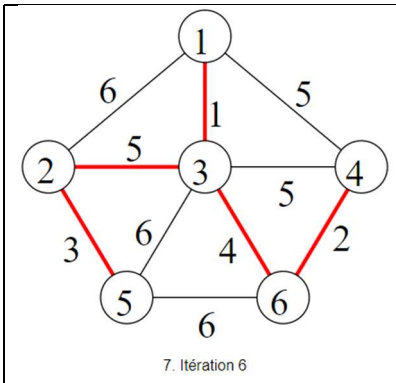
retourne(S,K)

fin Kruskal

Exemple

L'exemple détaille les différentes itérations de l'algorithme



 7. Itération 6		

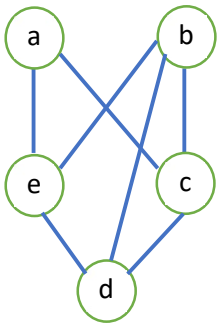
Chapitre V. Problèmes Hamiltonien et Eulérien

1. Problème Hamiltonien

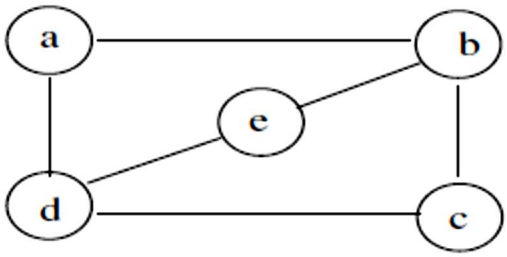
1.1. Définitions

Une chaine hamiltonienne passe une et une seule fois par chacun des n sommets du graphe. On appelle cycle hamiltonien un cycle élémentaire de longueur n. Un graphe possédant un cycle ou une chaine hamiltonien sera dit graphe hamiltonien.

Par exemple, le graphe suivant possède un cycle hamiltonien (< a, e, b, d, c, a >)



En revanche, le graphe suivant ne possède pas de cycle hamiltonien, mais possède une chaine hamiltonienne (< a, b, e, d, c >).



1.2. Condition nécessaire d'existence d'un cycle hamiltonien

Si l'on ne connait pas encore de conditions nécessaires pour affirmer qu'un graphe est hamiltonien ou non, il existe quelques conditions suffisantes non restrictives.

Condition 1 : Soit G=(V,E) un graphe non orienté. Si G possède un sommet de degré 1 il ne peut pas être hamiltonien. Ce résultat est assez évident, car pour qu'un graphe possède un cycle hamiltonien on doit pouvoir arriver et repartir par n'importe quel sommet, et donc le degré de chacun d'eux doit être au moins égal à 2.

Condition 2 : Condition de Dirac Gabriel Andrew Dirac (1925-1984) Soit G=(V,E) un graphe non orienté d'ordre n, avec n≥3. Si pour tout sommet de G on a d(x)≥ n/2 G est hamiltonien.

Condition 3 : Soit G=(V,E) un graphe non orienté d'ordre n, avec n≥3. Si le graphe G est complet alors il est hamiltonien.

2. Problème Eulérien

Une chaine eulérienne est une chaine qui emprunte une et une seule fois chaque arête du graphe.

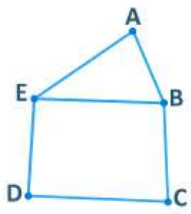
Un cycle eulérien est un cycle qui emprunte une et une seule fois chaque arête du graphe.

Un graphe comportant une chaine ou un cycle eulérien est appelé graphe eulérien.

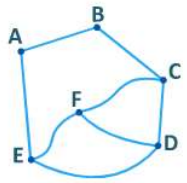
Théorème : Un graphe (simple ou multiple) connexe admet un cycle eulérien si et seulement s'il n'a pas de sommet de degré impair.

Théorème : Un graphe (simple ou multiple) connexe admet une chaine eulérienne entre deux sommets u et v si et seulement si le degré de u et le degré de v sont impairs, et les degrés de tous les autres sommets du graphe sont pairs.

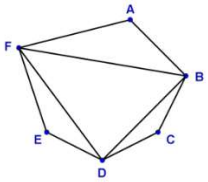
Exemples



Ce graphe possède la chaine eulérienne suivante :B,A,E,D,C,B,E



Ce graphe ne contient pas une chaine eulérienne puisqu'il existe trois sommets d'ordres impair



Ce graphe contient le cycle eulérien suivant : A,B,C,D,B,F,D,E,F,A

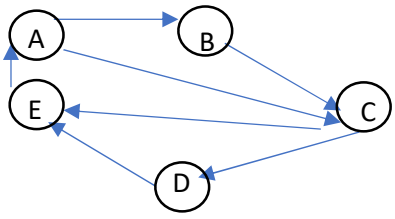
Un chemin eulérien est un chemin qui emprunte une et une seule fois chaque arc du graphe.

Un circuit eulérien est un circuit qui emprunte une et une seule fois chaque arc du graphe.

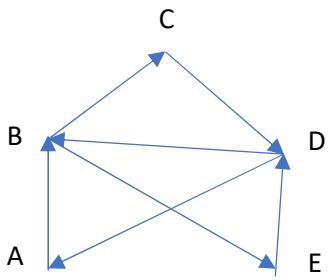
Théorème : Un multigraphe orienté fortement connexe admet un circuit eulérien si et seulement si $d^+(s_i) = d^-(s_i)$ pour tout sommet $s_i \in S$.

Théorème : Un graphe orienté connexe admet un chemin eulérien de u vers v si et seulement si

- $d^+(u) = d^-(u) + 1$
- $d^+(v) = d^-(v) - 1$
- $d^+(s_i) = d^-(s_i)$ pour tout autre sommet si $s_i \in S - \{u, v\}$



Le chemin A,B,C,D,E,A,C,E est chemin eulérien de longueur 7.



A,B,E,D,B,C,D,A est un circuit eulérien

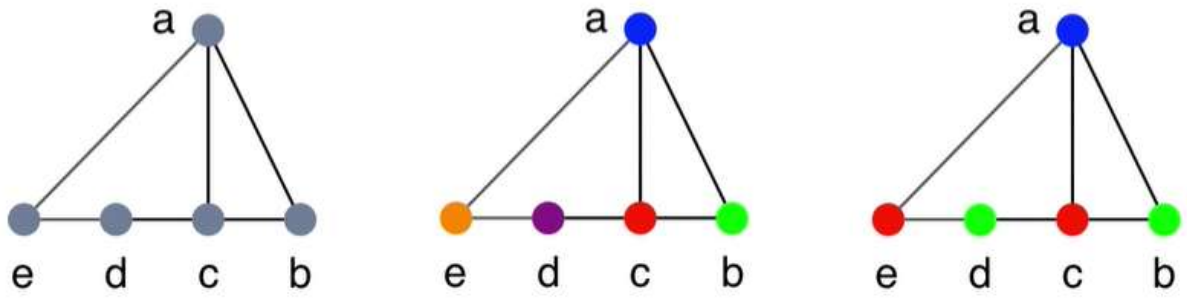
Chapitre VI. Coloration

1. Définitions

Pour un graphe non orienté G, le Coloriage consiste à attribuer une couleur à chaque sommet, de telle sorte qu'une même couleur ne soit pas attribuée à deux sommets adjacents.

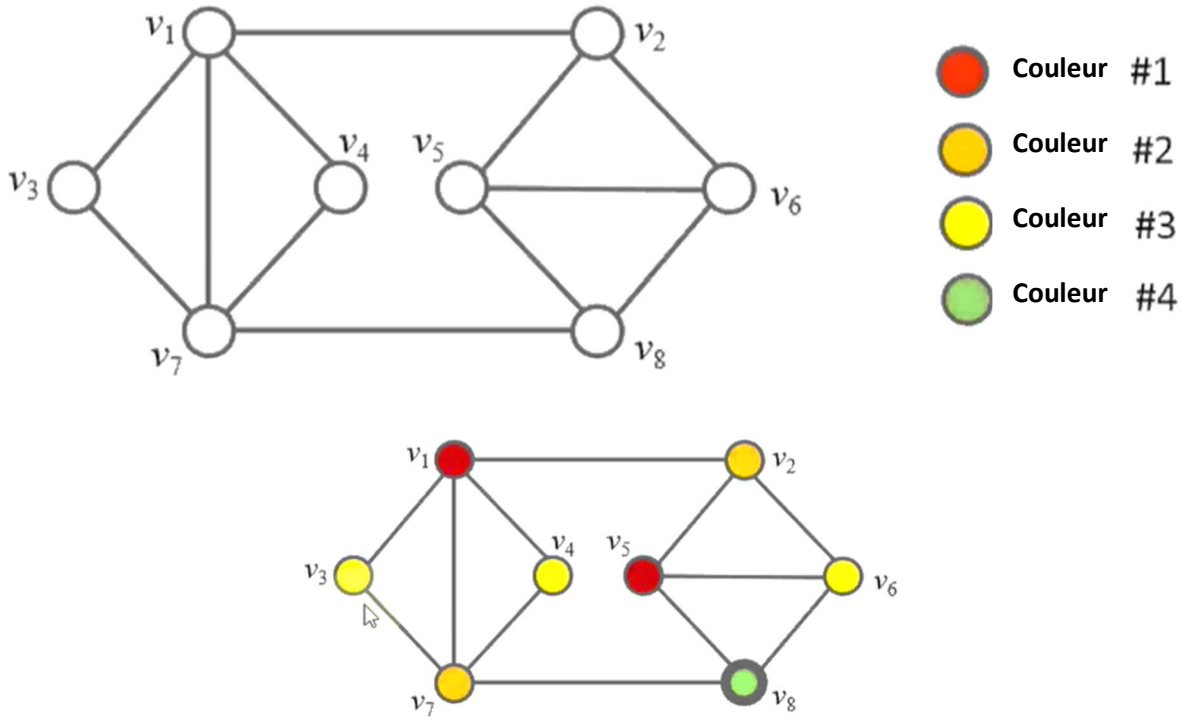
Le nombre minimum de couleurs nécessaires pour colorier un graphe G est appelé le nombre chromatique de G, et noté X(G).

2. Coloration des sommets



L'algorithme de Brélaz (DSATUR)

Tant qu'il existe un sommet non colorié, on choisit à chaque itération le sommet non colorié ayant le plus grand nombre de voisins coloriés avec des couleurs différentes, les ex aequo étant départagés en choisissant le sommet de plus fort degré. Le sommet choisi est alors colorié par la plus petite couleur possible (en partant du principe que les couleurs sont triées selon un ordre donné), si toutes les couleurs existantes sont utilisées par au moins un voisin du sommet à colorier, alors une nouvelle couleur est ajoutée à l'ensemble des couleurs disponibles.



Conjecture des 4 couleurs : Le nombre chromatique d'un graphe planaire est inférieur ou égal à 4.

3. Coloration des arêtes

Une coloration des arêtes d'un graphe G est une affectation de couleurs aux arêtes telle que les arêtes ayant une extrémité en commun sont de couleur différente. On cherche généralement à déterminer une coloration utilisant aussi peu de couleurs que possible. Le plus petit nombre de couleurs nécessaires pour colorer les arêtes d'un graphe G s'appelle « l'indice chromatique » de G et est noté q(G).

4. Le théorème des "4 couleurs"

Le théorème des quatre couleurs indique qu'il est possible, en n'utilisant que quatre couleurs différentes, de colorier n'importe quelle carte découpée en régions connexes, de sorte que deux régions adjacentes (ou limitrophes), c'est-à-dire ayant toute une frontière (et non simplement

un point) en commun reçoivent toujours deux couleurs distinctes. L'énoncé peut varier et concerner, de manière tout à fait équivalente, la coloration des faces d'un polyèdre, ou celle des sommets d'un graphe planaire. Trivialement, chacune des régions doit recevoir une couleur différente si les régions sont deux à deux adjacentes ; c'est le cas par exemple de la Belgique, du Luxembourg, de l'Allemagne et de la France dans une carte politique de l'Europe, d'où la nécessité des quatre couleurs dans le cas général. Par ailleurs, il ne peut exister cinq régions connexes deux à deux adjacentes (c'est la partie facile du théorème de Kuratowski). Lorsqu'on généralise le problème à un graphe quelconque, il devient NP-complet de déterminer s'il est coloriable avec seulement quatre couleurs (ou même trois).

5. Graphe parfait

Un ***ordre*** des sommets de G est dit ***parfait*** si aucune chaîne induite sur 4 sommets a,b,c,d avec les arêtes $[a,b]$, $[b,c]$ et $[c,d]$ est telle que $a<b$ et $d<c$. Si G possède un tel ordre, G est dit ***parfaitement ordonnable***, et l'algorithme séquentiel de coloration basé sur cet ordre donnera une coloration en $\chi(G)$ couleurs.

Il est clair que $\omega(G)\leq\chi(G)$ pour tout graphe G puisque les sommets d'une clique doivent avoir des couleurs différentes. Un ***graphe*** G est ***parfait*** si $\omega(H)=\chi(H)$ pour tout sous-graphe induit H de G.

Un graphe est parfait si et seulement si son complémentaire est parfait. En notant $\theta(G)$ le nombre minimum de cliques nécessaires pour recouvrir tous les sommets de G, on a donc que G est parfait si et seulement si $\alpha(H)=\theta(H)$ pour tout sous-graphe induit H.

Un graphe G est parfait si et seulement si ni G ni son complémentaire ne contient un cycle impair induit de longueur au moins 5.

Exercices Corrigés

Exercice 1 :

Trois enseignants E1, E2, E3 devront présenter ce lundi quelques cours pour trois classes C1, C2, C3:

E1 doit enseigner C1 de 2 heures et C2 de 1 heure ;

E2 doit enseigner C1 de 1 heure, C2 de 1 heure et C3 de 1 heure ;

E3 doit enseigner C1 de 1 heure, C2 de 1 heure et C3 de 2 heures.

Modéliser ce problème sous forme un graphe ? donner son type.

Quel est le nombre des créneaux nécessaire ?

En utilisant ce graphe proposer un emploi du temps

Exercice 2 :

Soit le graphe orienté G = (S, A) tel que

S = {1, 2, 3, 4, 5} A = {(1, 2),(1, 4),(2, 2),(2, 3),(2, 4),(3, 5),(4, 3),(5, 3)}

1. Dessiner le graphe G.

2. Donner le demi-degré extérieur de 2 et le demi-degré intérieur de 4,

3. Donner les sommets prédécesseurs de 4 et les sommets successeurs de 2,

4. Proposer un graphe partiel et un sous-graphe de ce graphe.

Exercice 3 :

Dessiner un graphe non orienté complet à 4 sommets.

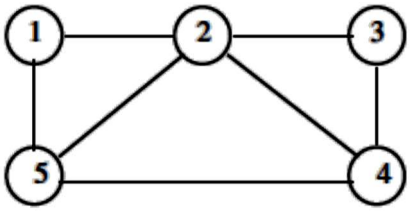
Quel est le degré des sommets de ce graphe ?

Combien d'arêtes possède-t-il ?

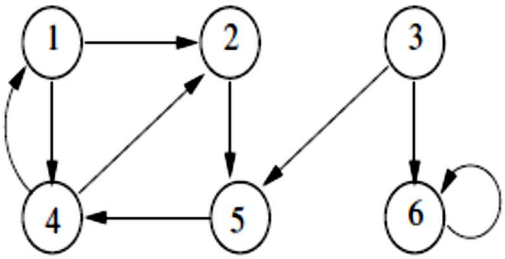
Généralisez ces résultats à un graphe simple complet ayant n sommets.

Exercice 4 :

Représenter par une matrice d'adjacence et des listes d'adjacence le graphe non orienté suivant :

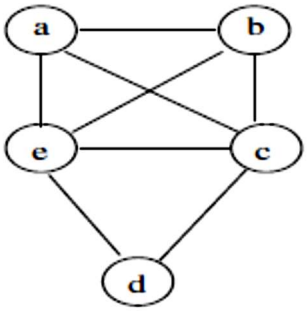


Donnez les représentations par une matrice d'adjacence et des listes d'adjacence du graphe orienté suivant :



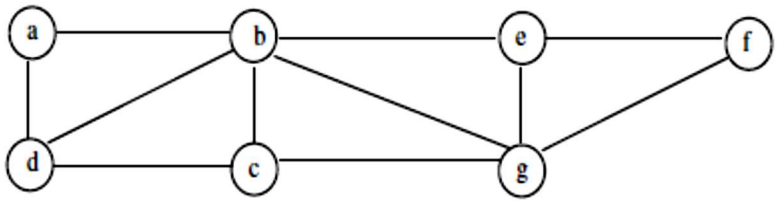
Exercice 5 :

Montrer que le graphe suivant est eulérien



Exercice 6 :

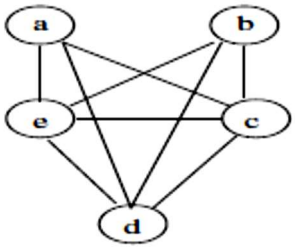
On considère le graphe non orienté suivant :



Combien faut-il enlever d'arêtes à ce graphe pour le transformer en arbre ? Donnez un graphe partiel de ce graphe qui soit un arbre.

Exercice 7 :

Montrer que le graphe suivant est planaire :

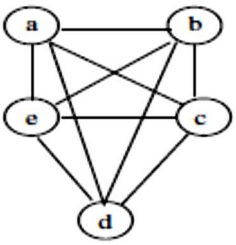


Exercice 8 :

Vérifiez la formule d'Euler dans le cas d'un arbre.

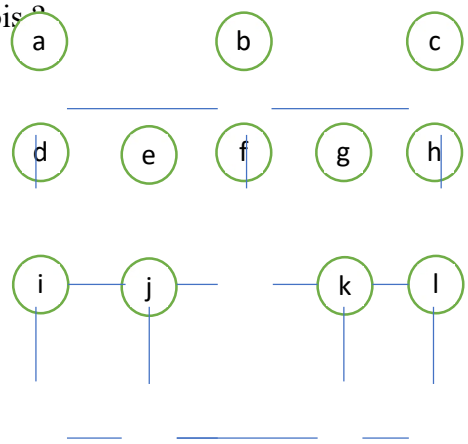
Exercice 9 :

Montrez, en utilisant la formule d'Euler que le graphe suivant n'est pas planaire.



Exercice 10 :

Est-il possible de dessiner sans lever la main un lacet qui traverse chaque arête de ce graphe planaire une et une seule fois ?



Exercice 11 :

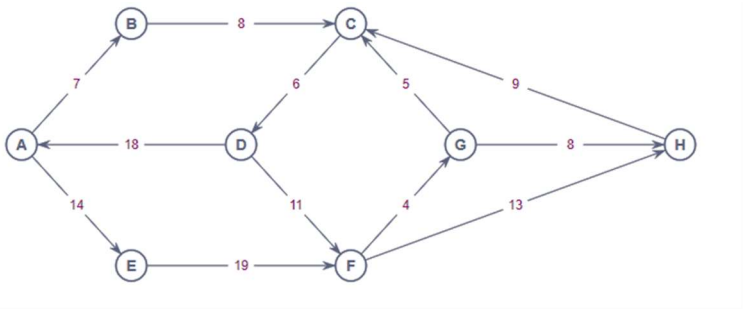
Cinq élèves (mohamed, ali, taha, samir et omar) doivent passer certains examens. Les examens que doivent passer chaque élève sont récapitulés dans le tableau suivant :

Mohamed	Français, Anglais, Mécanique
Ali	Dessin, Couture
Taha	Anglais, Solfège
Samir	Dessin, Couture, Mécanique
Omar	Dessin, Solfège

On désire que tous les élèves devant subir un même examen le fassent en même temps. Chaque étudiant ne peut se présenter qu'à une épreuve au plus par jour. Quel est le nombre minimal de jours nécessaires à l'organisation de toutes les épreuves ?

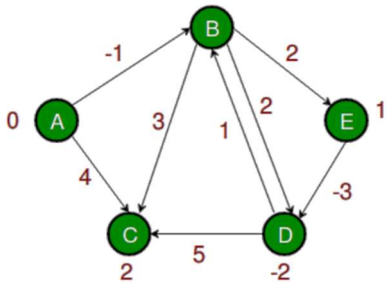
Exercice 12 :

Trouver les plus courts chemins à partir de A (Algorithme de Dijkstra)



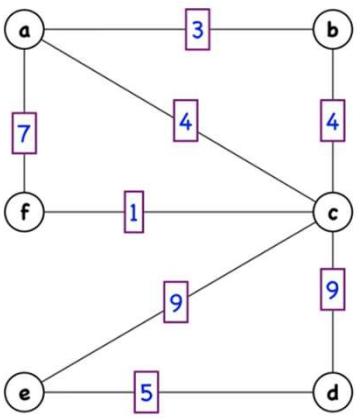
Exercice 13 :

Trouver les plus courts chemins à partir de A (Algorithme de Bellman-Ford)



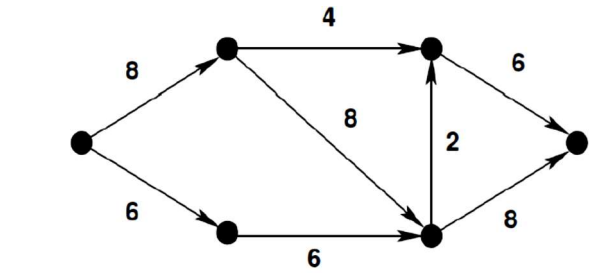
Exercice 14 :

Appliquer l'algorithme de Kruskal pour construire un arbre couvrant de poids minimal



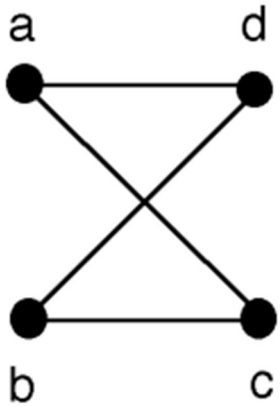
Exercice 15 :

Déterminer le flot maximal à partir du graphe de flot ci-dessous (Algorithme de Ford-Fulkerson)



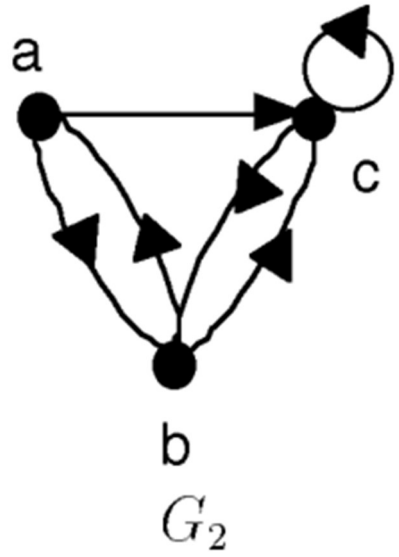
Exercice 16 :

Déterminer le nombre de chemins de longueur 4 allant de a à b dans le graphe G1



G_1

Déterminer le nombre de circuits de longueur 4 dans le graphe G2 :



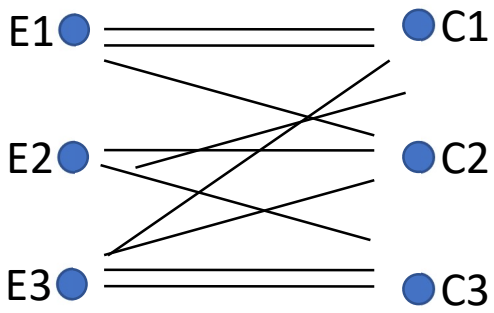
G_2

Correction

Exercice 1 :

On remarque qu'on a deux sous-ensembles de sommets les enseignants E et les groupes C. Toutes les relations (arêtes) relient un sommet de E avec un sommet de C. Dans ce cas, on a un graphe biparti.

On commence par relier les éléments de E avec les éléments de C avec des arêtes et on obtient le graphe suivant :

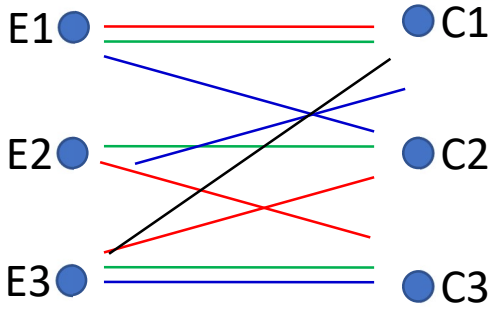


Le nombre max de créneaux horaires correspond au nombre max d'heures que peut dispenser un enseignant qui est le degré max des éléments de E

Max(d(Ei))=d(E3)=4; donc il nous faut quatre heures dans notre emploi du temps.

On adopte la convention suivante :

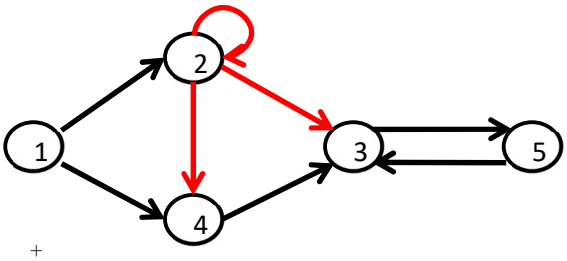
1ère heure en rouge, 2ème en vert, 3ème en bleu et la 4ème en noir, on commence par affecter la 1ère heure (arbitrairement) puis la 2ème, la 3ème et enfin la 4ème comme dans ce graphe :



On obtient le planning suivant :

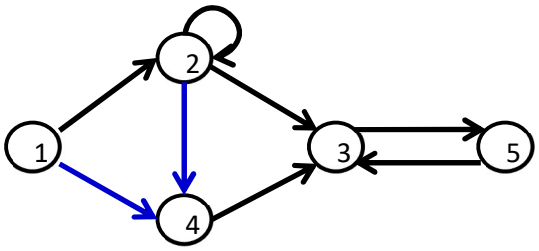
	E1	E2	E3
1ère heure(rouge)	C1	C3	C2
2ème heure(vert)	C1	C2	C3
3ème heure (bleu)	C2	C1	C3
4ème heure (noir)			C1

Exercice 2 :



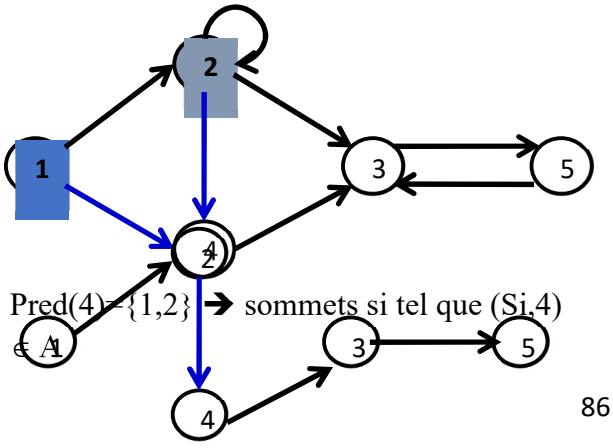
$d^+(2)=3$

- arcs en rouge partants du sommet 2

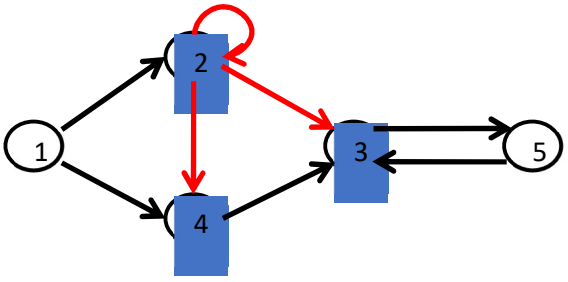


d -(4)=2

arcs en bleu arrivants au sommet 4



Pred(4)={1,2} ➔ sommets si tel que (Si,4) ∈ A



Succ(2)= {2,3,4} ➔ sommets sj tel que (2,Sj) ∈ A

Exercice 3

Le degré de chacun des 4 sommets du graphe est 3

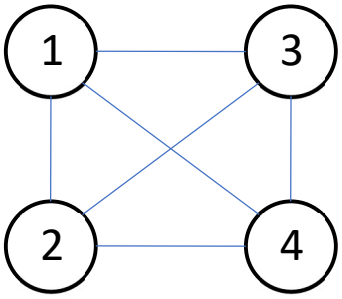
Le graphe comporte 6 arêtes.

Si on a n sommets d'un graphe complet, le degré de chaque sommet est de n-1, chaque arête participe au degré de ses extrémités et donc génère 2 degrés supplémentaires.

La somme des degrés de tous les sommets est de n(n-1).

Le nombre d'arêtes est déduit par |A|=n(n-1)/2 (puisque chaque arête augmente la somme des degrés par 2.

N.B: la propriété A=n(n-1)/2 des graphes simples complets peut être prouvée par récurrence.



Exercice 4 :

Pour les graphes non orientés, la matrice d'adjacence doit être forcément une matrice carrée symétrique ($a_{ij} = a_{ji}$). Son rang est de NxN ou N est le nombre de sommets du graphe. M(i,j)=M(j,i)=1 si $a_{ij} \in A$, M(i,j)=M(j,i)=0 si $a_{ij} \notin A$. Pour la représentation par liste d'adjacence on donne pour chaque sommet Si la liste des nœuds S_j tel que $a_{ij} \in A$.

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

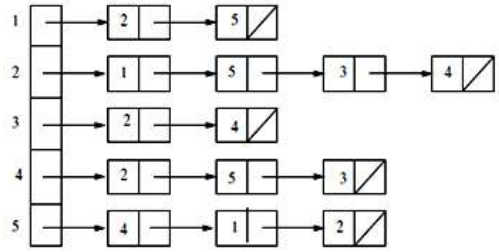
L1={2,5}

L2={1,3,4,5}

L3={2,4}

L4={2,3,5}

L5={1,2,4}



Pour les graphes orientés (comme B), la matrice d'adjacence n'est pas forcément symétrique.

Son rang est NxN ou N est le nombre de sommets du graphe.

M(i,j)=1 si $a_{ij} \in A$, M(i,j)=0 si $a_{ij} \notin A$.

M	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	1	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	2

La représentation en listes d'adjacence se fait de la même manière (avec des arcs).

L(1)={2,4}

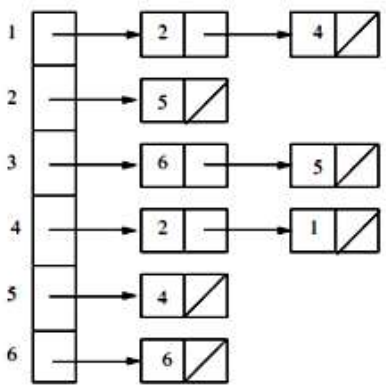
L(2)={5}

L(3)={5,6}

L(4)={1,2}

L(5)={4}

L(6)={6}



Exercice 5 :

Le degré de chacun des 4 sommets du graphe est 3

Le graphe comporte 6 arêtes.

Si on a n sommet d'un graphe complet, le degré de chaque sommet est de n-1, chaque arête participe au degré de ses extrémités et donc génère 2 degrés supplémentaires.

La somme des degrés de tous les sommets est de n(n-1).

Le nombre d'arêtes est déduit par |A|=n(n-1) /2 (puisque chaque arête augmente la somme des degrés par 2.

N.B: la propriété |A=n(n-1) /2 des graphes simples complets peut être prouvée par récurrence.

Exercice 6 :

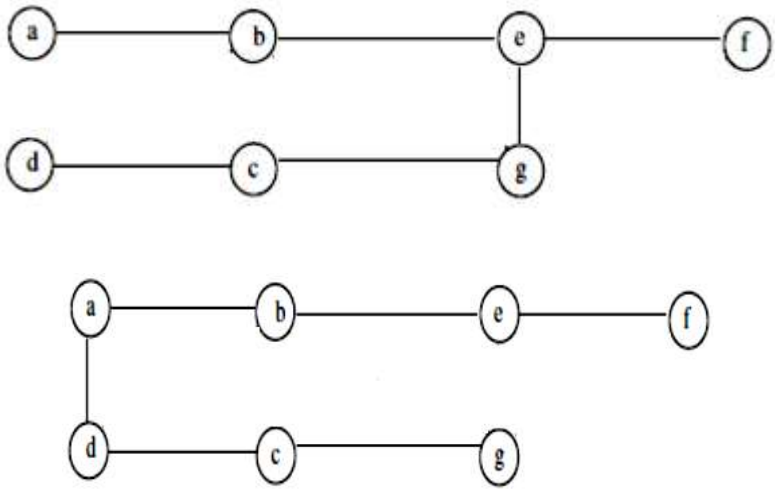
Soient |A| et |S| respectivement, le nombre d'arêtes et de sommets d'un graphe G

G est un arbre ➔ |A|=|S|-1

Dans notre cas on a |S|=7, pour que G soit un arbre il faut que |A|=7-1=6

Et comme |A|=11 donc il faut enlever au minimum 11-6=5 arêtes.

Les deux graphes ci-dessous représentent deux graphes partiaux de G et qui sont des arbres.



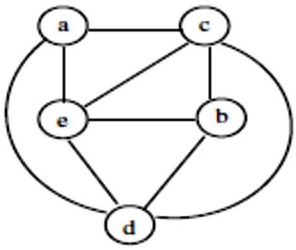
Exercice 7 :

Le graphe est planaire s'il n'a pas de mineurs (sous graphe obtenu par fusion ou suppression de sommets) isomorphe à K3,3 ou K5

Le graphe ci-dessous n'a pas de sous graphe de K3,3 du fait que son nombre de sommet est 5 (un K3,3 est un graphe de 6 sommets),

Le graphe n'est pas isomorphe avec K5 du fait que l'arête ab est inexistante.

Le graphe ci-dessus est donc planaire, en effet il peut être redessiner comme suit :



Exercice 8 :

Nombre de faces d'un arbre=1 (sans cycle) et le nombre d'arêtes= s-1, f+s=1+(a+1)=a+2.

Exercice 9 :

f: nombre de faces, a: nombre d'arêtes et s: nombre de sommets

G est simple (pas d'arêtes multiples ni de boucles). Chaque face est délimitée au moins de trois arêtes.

Si on compte f faces on va compter au moins 3f arêtes, certaines arêtes étant communes, mais pas toutes (les isthmes). D'où l'inégalité :

$3f <= 2a$ (1)

et comme f+s=a+2 (formule d'euler) on a $f=a+2-s$ ➔ $3f=3a-3s+6$ (2)

De (1) et (2) on a

$3a-3s+6 <= 2a$ ➔ $a+6<=3s$ (3)

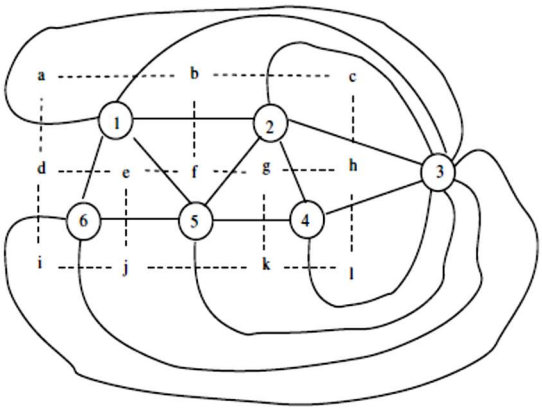
Pour le graphe (K5) de l'exercice on a=10 et s=5 ; en remplaçant dans (3) on trouve 16<=15 (contradiction) donc K5 n'est pas planaire.

Exercice 10 :

Considérons le graphe G' obtenu comme suit :

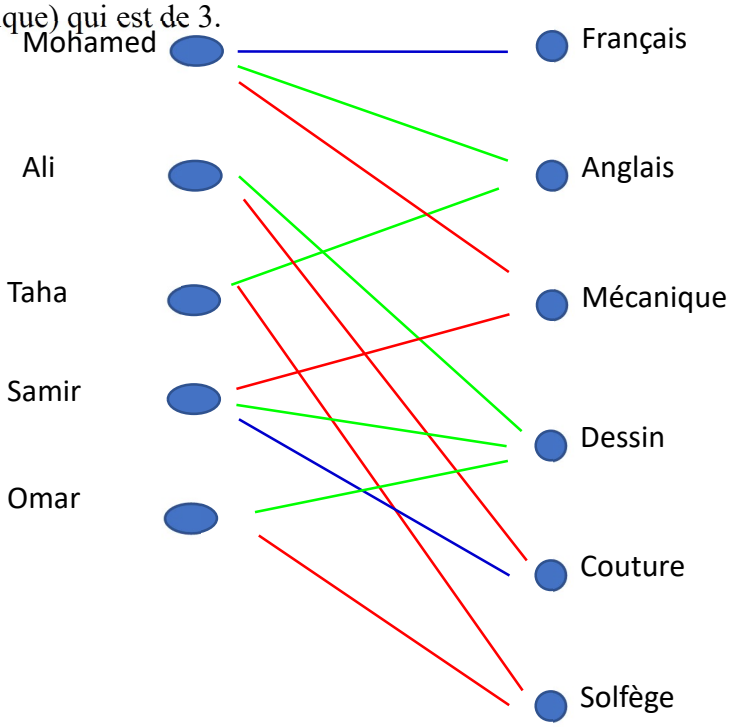
- À chaque face f_i de G correspond un sommet si dans G'
- À chaque arête de G frontière entre deux faces f_i et f_k correspond une arête a_{ik} reliant les sommets s_i et s_k

Traverser toutes les arêtes de G revient à trouver une chaine Eulérienne dans G' or on a 4 sommets de G' de degré impair (1, 2, 3 et 5) donc pas de chaine Eulérienne et le problème n'admet pas de solution.



Exercice 11

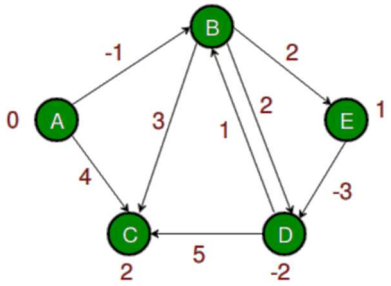
Après représentation par graphe, le problème revient au coloriage de ce graphe (trouver le nombre chromatique) qui est de 3.



Exercice 12

Etape	N	D(A), P(A)	D(B), P(B)	D(C), P(C)	D(D), P(D)	D(E), P(E)	D(F), P(F)	D(G), P(G)	D(H), P(H)
Initialisation	{}	0,-	∞,-	∞,-	∞,-	∞,-	∞,-	∞,-	∞,-
Etape 1	{A}		7,A	∞,-	∞,-	14,A	∞,-	∞,-	∞,-
Etape 2	{A,B}			15,B	∞,-	14,A	∞,-	∞,-	∞,-
Etape 3	{A,B,E}			15,B	∞,-		33,E	∞,-	∞,-
Etape 4	{A,B,E,C}				21,C		33,E	∞,-	∞,-
Etape 5	{A,B,E,C,D}						~~33,E~~ 32,D	∞,-	∞,-
Etape 6	{A,B,E,C,D,F}							36,F	45,F
Etape 7	{A,B,E,C,D,F,G}								~~45,F~~ 44,G

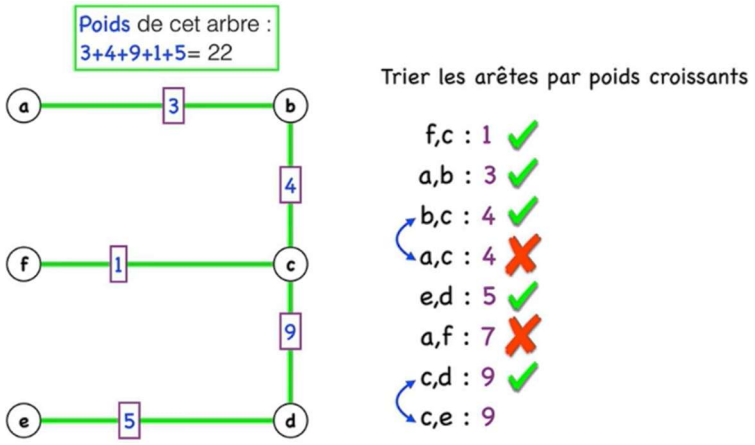
Exercice 13 :



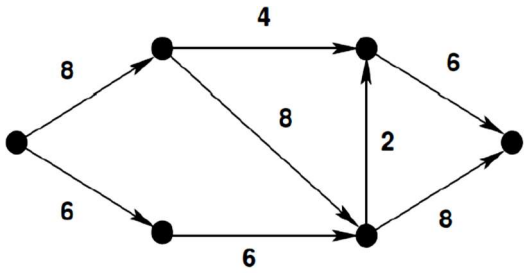
	A	B	C	D	E
V0	0	∞	∞	∞	∞
V1	0	**-1**	4	∞	∞
V2		-1	4	1	**1**

			2		
V3		-1	2	1 -2	**1**
V4		-1	**2**	**-2**	1

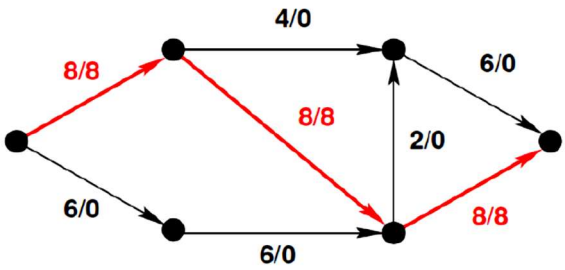
Exercice 14 :



Exercice 15 :



S P



Exercice 16 :

La matrice d'adjacence de G_1 est :

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

Le nombre de chemins cherché est le terme $(1, 2)$ de la matrice

$$M^4 = \begin{pmatrix} 8 & 8 & 0 & 0 \\ 8 & 8 & 0 & 0 \\ 0 & 0 & 8 & 8 \\ 0 & 0 & 8 & 8 \end{pmatrix}$$

La matrice d'adjacence de G_2 est :

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

Le nombre de circuits de longueur 4 dans G_2 est égale à la trace de M^4 :

$$M^4 = \begin{pmatrix} 3 & 5 & 8 \\ 2 & 6 & 8 \\ 3 & 5 & 8 \end{pmatrix}$$

Exemples d'examens

Exemple 1

Exercice 1 (9 pts)

A) Ce tableau est la première itération de l'algorithme de Dijkstra

A	B	C	D	E	F
0	5/A	2/A	∞	∞	6/A

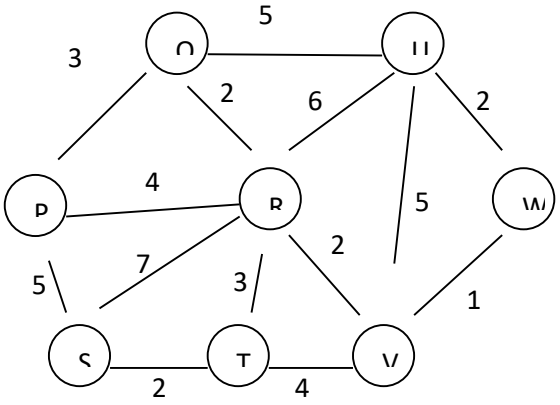
1. Quel est le sommet de départ ? …
2. Quels sont les deux sommets qui ne sont pas directement connectés au sommet de départ ? ……
3. Quel sommet sera le prochain sommet de l'algorithme ? …
4. Quelle est la valeur de l'arc (A, F) ? …

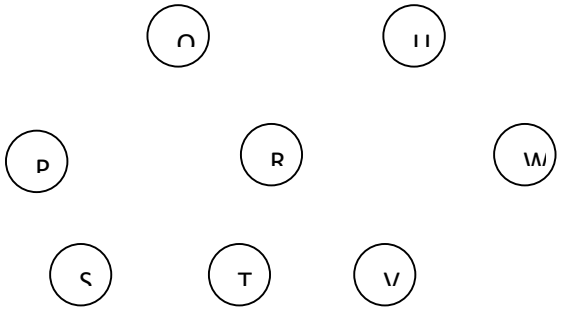
B) Ce tableau est le résultat final de l'application de l'algorithme de Dijkstra sur un autre problème

A	B	C	D	E	F	G	H	I
0	2/A	1/A	∞	5/A	∞	∞	∞	∞
0	2/A	1/A	∞	5/A	4/C	∞	∞	∞
0	2/A	1/A	6/C	5/A	4/C	∞	∞	∞
0	2/A	1/A	6/C	5/A	4/C	∞	8/F	∞
0	2/A	1/A	6/C	5/A	4/C	7/E	7/E	∞
0	2/A	1/A	6/C	5/A	4/C	7/E	7/E	∞
0	2/A	1/A	6/C	5/A	4/C	7/E	7/E	8/G
0	2/A	1/A	6/C	5/A	4/C	7/E	7/E	8/G

1. Quelle est la longueur du chemin le plus court de A à G ? …
2. Quel est le chemin le plus court de A à G ? …….
3. Quelle est la longueur du chemin le plus court de A à I ? …
4. Quel est le chemin le plus court de A à I ? ……

C) Appliquer l'algorithme de Kruskal pour construire un arbre couvrant de poids minimal

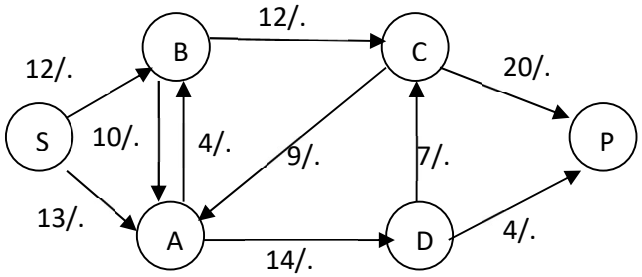




Exercice 2 (5 pts)

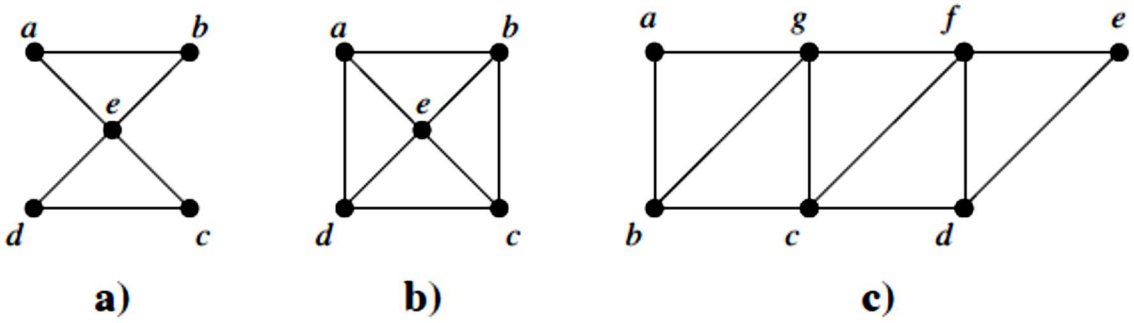
Déterminer le flot maximal à partir du graphe de flot ci-dessous

(Algorithme de Ford-Fulkerson)



Exercice 3 (6 pts)

A) Parmi ceux-ci, lesquels sont des graphes eulériens ou graphes hamiltoniens.



Graphes eulériens : ……

Graphes hamiltoniens : …….

QCM) Il y a toujours une seule réponse correcte

- Un circuit absorbant est un circuit de coût :

1- Négatif
2- Positif
3- Nul

- Le nombre chromatique d'un graphe planaire est

1- (≤ 4) 2- (= 4) 3- (= 2)

- Le parcours en largeur se base sur une file d'attente

1- FIFO 2- LIFO

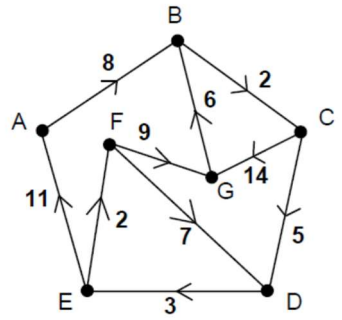
- La matrice $M+M^2+M^3$ permet de connaître le nombre de chemins de longueurs

1- (= 3) 2- (≤ 3) 3- (< 3)

Exemple 2

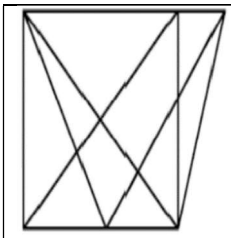
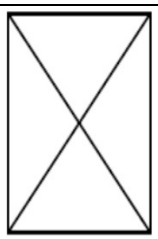
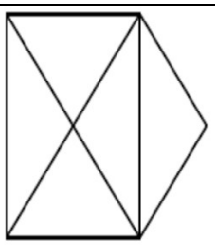
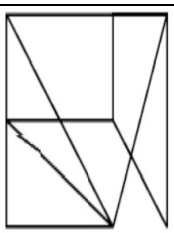
Exercice 1 (9 pts)

Donner le plus court chemin du sommet A aux autres sommets



Exercice 2 (5 pts)

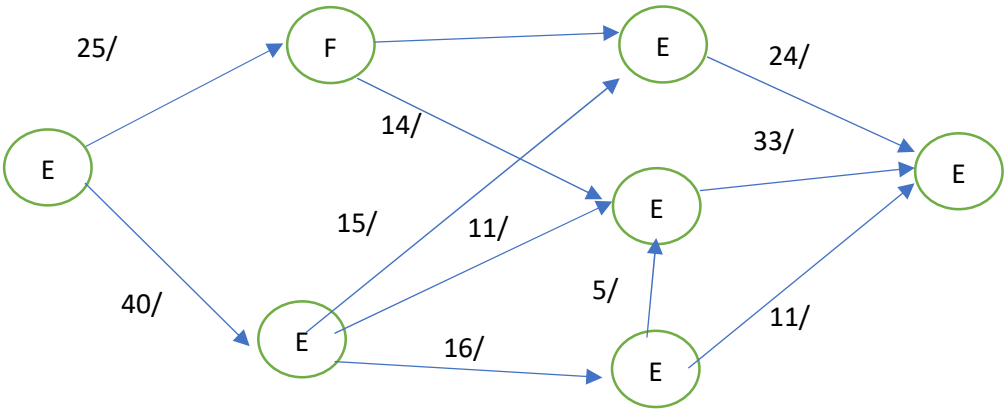
Est-il possible de tracer les figures suivantes sans lever le crayon et sans passer deux fois sur le même trait. Justifiez votre réponse.

			
A	B	C	D

Exercice 3 (6 pts)

On considère le réseau de transport suivant :

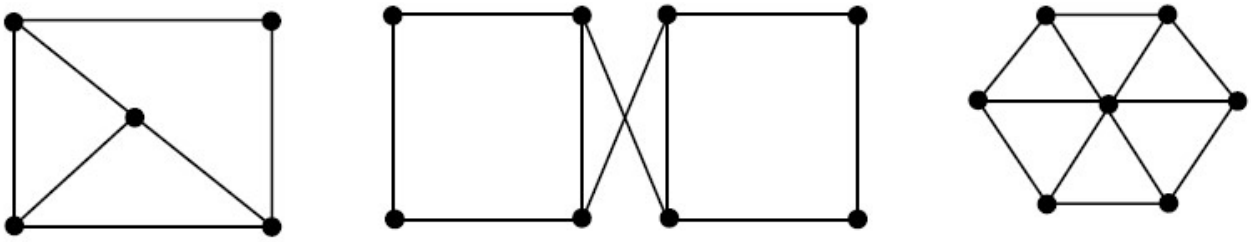
1) Construire un flot initial admissible et complet.
2) Appliquer l'algorithme de Ford-Fulkerson à ce flot initial pour déterminer le flot maximum.



Exemple 3

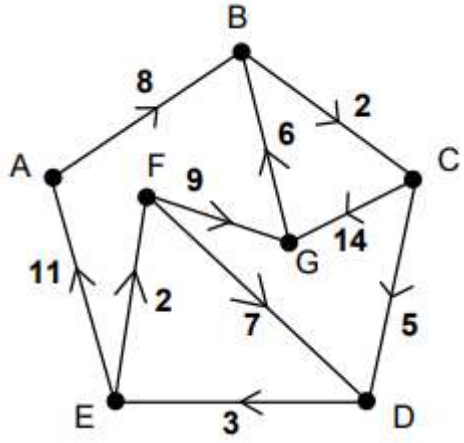
Exercice 1 (5 pts)

Déterminer le nombre chromatique des graphes suivants :



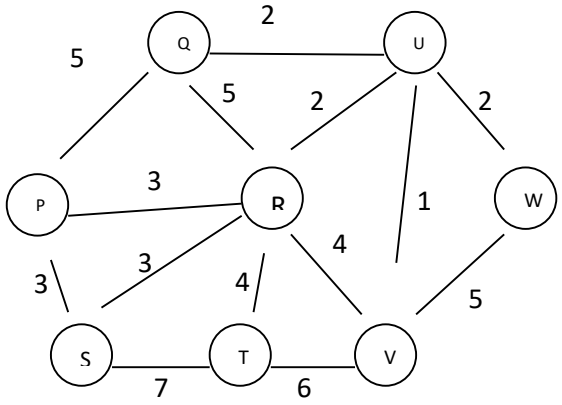
Exercice 2 (6 pts)

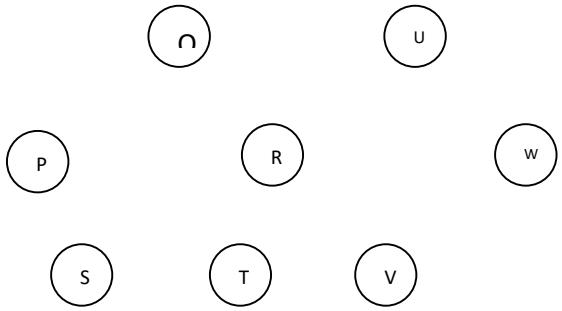
Exécutez l'algorithme de Dijkstra sur le graphe suivant, à partir du sommet C, puis à partir du sommet F.



Exercice 3 (4 pts)

Appliquer l'algorithme de Kruskal pour construire un arbre couvrant de poids minimal

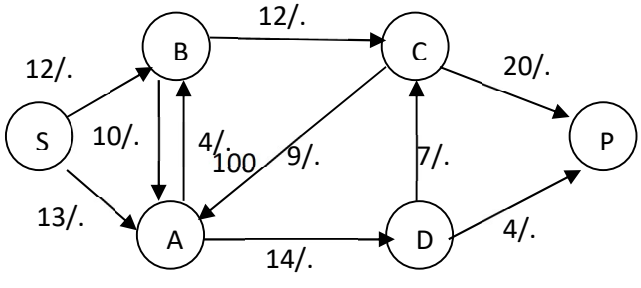




Exercice 4 (5 pts)

Déterminer le flot maximal à partir du graphe de flot ci-dessous

(Algorithme de Ford-Fulkerson)



QCM

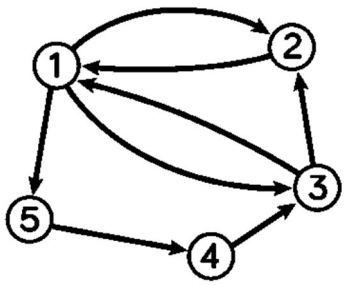
1. Quels sont les éléments indispensables pour définir un graphe orienté :

☐ Arête ☐ Arc ☐ Sommet ☐ Boucle

2. Quels sont les éléments indispensables pour définir un graphe non orienté :

☐ Arête ☐ Arc ☐ Sommet ☐ Boucle

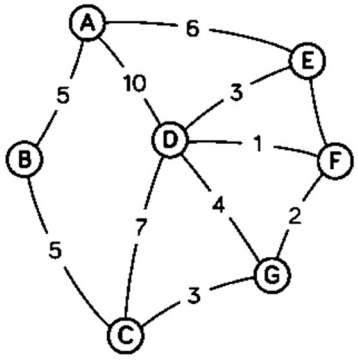
3. On considère le graphe suivant :



Il s'agit d'un graphe

☐ Pondéré ☐ Non orienté ☐ Orienté ☐ Complet

4. On considère le graphe suivant :



La chaîne du sommet A au sommet G de poids minimal est :

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

5. Soit la matrice d'adjacence suivante :

0	1	0	0	0
1	0	1	1	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0

Le nombre de chaînes de longueur 4 reliant le sommet 1 au sommet 4 est :

☐ 2 ☐ 4 ☐ 6 ☐ 8

6. Le tableau suivant donne les degrés des sommets d'un graphe d'ordre 6 :

X	1	2	3	4	5	6
D(x)	2	3	5	4	3	3

Le nombre d'arêtes de ce graphe est :

☐ 2 ☐ 6 ☐ 10 ☐ 20

7. Existe-t-il un graphe non orienté dont la liste des degrés des sommets est 4,2,2,2 ?

☐ Oui ☐ Non

8. Une chaîne eulérienne doit passer par tous les chemins une seule fois.

☐ Oui ☐ Non

9. Qu'est-ce que l'ordre d'un graphe ?

☐ Nombre d'arcs ☐ Nombre d'arêtes ☐ Nombre de sommets ☐ Nombre de tous les éléments

Questions

10. Qu'est qu'un sous graphe partiel ?

11. Qu'est-ce qu'un arbre couvrant ?

12. Qu'est-ce qu'un graphe planaire ?

Références Bibliographiques

SOLNON, Christine. Théorie des graphes et optimisation dans les graphes. *INSA de Lyon*.

FLAMENT, Claude. *Théorie des graphes et structures sociales*. Walter de Gruyter GmbH & Co KG, 2017.

MAQUIN, Didier. Eléments de Théorie des graphes. *Ecole Nationale Supérieur d'Electricité et de Mécanique*, 2003.

RIGO, Michel. Théorie des graphes. *Université de Liège*, 2009.

BRETTO, Alain, FAISANT, Alain, et HENNECART, François. *Éléments de théorie des graphes*. Springer, 2012.

COLIN, Fabrice. EN THÉORIE DES GRAPHES. 2003.

FOURNIER, Jean-Claude. *Théorie des graphes et applications: Avec exercices et problèmes*. Lavoisier, 2011.

BIGGS, Norman, LLOYD, E. Keith, et WILSON, Robin J.

Graph Theory, 1736-1936. Oxford University Press, 1986.

WEST, Douglas Brent, *et al. Introduction to graph theory*. Upper Saddle River : Prentice hall, 2001.

GROSS, Jonathan L. et YELLEN, Jay (ed.). *Handbook of graph theory*. CRC press, 2003.

CHEN, Wai-Kai. *Applied graph theory*. Elsevier, 2012.

GOLUMBIC, Martin Charles. *Algorithmic graph theory and perfect graphs*. Elsevier, 2004.

GIBBONS, Alan. *Algorithmic graph theory*. Cambridge university press, 1985.

CHARTRAND, Gary. *Introductory graph theory*. Courier Corporation, 1977.