Ministry of Higher Education and Scientific Research

Badji Mokhtar Annaba University

Faculty of Technology

وزارة التعليم العالي و البحث العلمي

جـامعة باجـي مختـار- عنابـة

كليـة الــتكنـولوجيا

Département d'informatique

Polycopié pédagogique

Titre

Théorie des Graphes

par Dr Guessoum Souad

Cours destiné aux étudiants de

2ème année ingénieur en informatique

Année : **2025/2026**

Table des matières

Liste des figures

Liste des tables

Introduction générale

La théorie des graphes constitue aujourd'hui l'un des domaines fondamentaux des mathématiques discrètes et de l'informatique théorique et appliquée. Elle offre un cadre mathématique puissant permettant de modéliser et d'analyser de nombreux systèmes complexes composés d'éléments interconnectés.

La théorie des graphes s'est progressivement imposée comme un outil incontournable dans de nombreux domaines scientifiques et technologiques. Elle intervient notamment dans l'étude des réseaux de communication, des réseaux de transport, des circuits électroniques, de l'optimisation des flux, de l'analyse des réseaux sociaux, ainsi que dans de nombreuses applications en informatique, en recherche opérationnelle et en intelligence artificielle.

L'objectif principal de ce module est d'introduire les concepts fondamentaux de la théorie des graphes et de fournir aux étudiants les outils nécessaires pour modéliser et résoudre des problèmes concrets à l'aide de structures de graphes. Le cours mettra particulièrement l'accent sur la compréhension des propriétés structurelles des graphes, l'étude des graphes orientés et non orientés, ainsi que sur l'analyse des principaux problèmes algorithmiques liés aux graphes.

Au cours de ce module, les étudiants seront amenés à se familiariser avec les notions essentielles telles que les chemins, les circuits, les composantes connexes, les parcours de graphes, ainsi que les problèmes d'optimisation sur les réseaux, notamment le problème du flot maximal et le problème des plus courts chemins. Une attention particulière sera également portée aux méthodes algorithmiques permettant de traiter efficacement ces problèmes.

À l'issue de ce module, les étudiants devront être capables de :

- comprendre et maîtriser les notions fondamentales de la théorie des graphes ;
- modéliser des situations réelles à l'aide de graphes ;
- analyser les propriétés structurelles d'un graphe ;
- appliquer des algorithmes classiques pour résoudre des problèmes de parcours et d'optimisation sur les graphes ;
- interpréter les résultats obtenus dans le contexte du problème étudié.

Ce polycopié a pour objectif de servir de support pédagogique structuré pour accompagner l'apprentissage de ces concepts. Il présente de manière progressive les notions théoriques essentielles, illustrées par des exemples et complétées par des exercices permettant de consolider la compréhension des différentes méthodes étudiées.

Etant donné le volume important du chapitre 3, à savoir « les cheminements dans les graphes », il a été scindé en 2 chapitres : « Cheminement dans les graphes non orientés » et « cheminement dans les graphes orientés » pour une assimilation plus progressive et plus fluide de la part de l'étudiant, ainsi qu'une pause pour entamer la série d'exercices du premier avant d'entamer le deuxième.

L'organisation de ce module est faite en sorte qu'à la fin de chaque chapitre, une série de travaux dirigés adaptés aux notions du chapitre est effectuée pour la maîtrise de son contenu. Les exercices sont ordonnés du plus simple aux plus compliqués pour susciter progressivement la concentration de l'étudiant. Les solutions des exercices sont placées en annexe de ce polycopié.

Chapitre 1
Introduction à la théorie des graphes

1.1 Introduction

La théorie des graphes constitue aujourd'hui l'un des domaines fondamentaux des mathématiques discrètes et de l'informatique théorique. Elle offre un cadre mathématique puissant pour la modélisation et l'analyse de systèmes composés d'entités interconnectées. Dans cette approche, les éléments d'un système sont représentés par des sommets, tandis que les relations entre ces éléments sont modélisées par des arêtes ou des arcs dans le cas des graphes orientés. L'intérêt de cette représentation réside dans sa capacité à abstraire la structure d'un problème indépendamment de sa nature physique ou conceptuelle. De nombreux systèmes complexes, tels que les réseaux de transport, les réseaux informatiques, les réseaux sociaux ou encore les systèmes biologiques, peuvent être décrits à l'aide de graphes. Cette modélisation permet ensuite d'analyser leurs propriétés structurelles et d'appliquer des méthodes algorithmiques afin de résoudre des problèmes d'optimisation ou de décision.

La théorie des graphes s'inscrit dans le cadre plus large des mathématiques combinatoires et des structures discrètes, dont l'objectif est l'étude de configurations finies d'objets et des relations qui les lient. Elle occupe aujourd'hui une position centrale dans plusieurs domaines scientifiques, notamment la recherche opérationnelle, l'intelligence artificielle, l'analyse des réseaux et la science des données.

Ce chapitre présente tout d'abord l'origine historique de la théorie des graphes, puis introduit la notion de problèmes combinatoires discrets, avant de mettre en évidence les principaux domaines d'application de cette discipline.

1.2 Origine de la théorie des graphes

La théorie des graphes trouve son origine dans les travaux du mathématicien suisse Leonhard Euler, qui s'intéressa en 1736 à un problème devenu célèbre : le problème des ponts de Königsberg. La ville de Königsberg, traversée par la rivière Pregel, comportait deux îles reliées entre elles et aux rives par sept ponts. La question posée consistait à déterminer s'il existait un parcours permettant de traverser chacun de ces ponts une et une seule fois.

Euler aborda ce problème en adoptant une démarche novatrice consistant à abstraire la structure géographique de la ville. Il représenta les différentes zones de terre par des points et les ponts par des lignes reliant ces points. Cette représentation constitue la première formalisation d'un graphe. En analysant les propriétés de cette structure, Euler démontra qu'un tel parcours était impossible.

Cette approche marque la naissance de la théorie des graphes et constitue l'un des premiers exemples d'utilisation d'une structure mathématique abstraite pour résoudre un problème concret. L'étude d'Euler introduit également des concepts fondamentaux tels que la notion de degré d'un sommet et les conditions d'existence de parcours eulériens.

Au cours du XIX^{e} et du XX^{e} siècle, la théorie des graphes s'est progressivement développée grâce aux contributions de nombreux mathématiciens. Parmi les avancées importantes figurent notamment l'étude des arbres par Arthur Cayley, les travaux de Dénes König sur la structure des graphes, ainsi que les contributions de nombreux chercheurs à l'étude des propriétés combinatoires des graphes.

À partir de la seconde moitié du XX^{e} siècle, l'essor de l'informatique et de la recherche opérationnelle a considérablement renforcé l'importance de la théorie des graphes. Les graphes sont devenus un outil fondamental pour la représentation et l'analyse des réseaux complexes, ainsi que pour la conception d'algorithmes efficaces permettant de résoudre des problèmes d'optimisation sur les réseaux.

1.3 Problèmes combinatoires discrets

La théorie des graphes appartient au domaine des mathématiques discrètes, qui s'intéressent à l'étude de structures finies ou dénombrables. Contrairement aux mathématiques continues, où les variables peuvent prendre une infinité de valeurs dans un intervalle, les mathématiques discrètes analysent des objets distincts et bien séparés.

Les problèmes combinatoires discrets consistent à étudier les différentes configurations possibles d'un ensemble fini d'éléments et à déterminer celles qui satisfont certaines contraintes ou qui optimisent un critère donné. Ces problèmes apparaissent dans de nombreux domaines scientifiques et technologiques et sont souvent caractérisés par une croissance rapide du nombre de configurations possibles lorsque la taille du problème augmente.

Dans le contexte de la théorie des graphes, les problèmes combinatoires prennent différentes formes. Parmi les plus importants, on peut citer :

- la recherche de chemins ou de circuits possédant certaines propriétés ;
- l'étude de la connexité d'un graphe et de ses composantes connexes ;
- la détermination de sous-graphes particuliers tels que les arbres couvrants ;
- les problèmes de coloration de graphes ;
- les problèmes d'optimisation de flux dans les réseaux ;
- les problèmes d'ordonnancement et de planification de tâches.

La résolution de ces problèmes repose souvent sur le développement d'algorithmes spécifiques exploitant la structure du graphe. Dans certains cas, des solutions efficaces peuvent être obtenues à l'aide d'algorithmes polynomiaux. Dans d'autres situations, les problèmes deviennent particulièrement complexes et appartiennent à des classes de complexité élevées, nécessitant le recours à des méthodes heuristiques ou approximatives.
Ainsi, la théorie des graphes joue un rôle essentiel dans l'étude et la résolution de nombreux problèmes combinatoires qui apparaissent dans les sciences de l'ingénieur et l'informatique.

1.4. Domaines d'application de la théorie des graphes

Grâce à sa capacité à représenter des systèmes composés d'éléments interconnectés, la théorie des graphes possède un très large champ d'applications dans de nombreux domaines scientifiques et technologiques. Le domaine des réseaux de transport, où les villes ou les

stations sont représentées par des sommets, tandis que les liaisons entre ces points sont représentées par des arêtes. Cette modélisation facilite l'étude de problèmes tels que la recherche de chemins optimaux, la planification des itinéraires ou l'optimisation des flux de circulation. Dans les réseaux informatiques et les télécommunications, les graphes servent à représenter les équipements et les connexions entre eux. Les algorithmes de graphes jouent un rôle central dans la conception des protocoles de routage, la gestion du trafic et l'analyse de la fiabilité des réseaux.

La théorie des graphes est également largement utilisée en recherche opérationnelle, notamment dans les problèmes d'optimisation liés à la gestion des ressources, à la planification de projets ou à la logistique. Les modèles de réseaux permettent par exemple d'analyser les flux de marchandises ou d'informations dans des systèmes complexes. Dans le domaine de l'informatique, les graphes interviennent dans de nombreuses structures de données et dans l'analyse des relations entre objets. Ils sont utilisés dans les compilateurs, les systèmes d'exploitation, les bases de données, ainsi que dans les moteurs de recherche pour l'analyse des liens entre pages web.

Plus récemment, la théorie des graphes joue un rôle majeur dans l'étude des réseaux sociaux, où elle permet d'analyser les interactions entre individus et la structure des communautés. Elle intervient également dans des domaines tels que la biologie des réseaux, où elle est utilisée pour étudier les interactions entre protéines ou les réseaux métaboliques. Enfin, les graphes sont devenus un outil essentiel dans des domaines émergents tels que la science des données, l'apprentissage automatique et l'intelligence artificielle, où ils permettent de représenter et d'exploiter des structures relationnelles complexes.

1.5 Conclusion

Dans ce chapitre, les origines historiques de la théorie des graphes ainsi que son lien étroit avec les problèmes combinatoires discrets ont été mis en évidence. Nous avons montré l'importance de ces derniers dans la résolution de certains problèmes réels. Également, les domaines d'application des graphes ont été soulignés allant des réseaux de transport et de télécommunication à la recherche opérationnelle, aux réseaux sociaux, à la science des données et à l'intelligence artificielle.

Chapitre 2
Notions Fondamentales de la Théorie des Graphes

2.1 Introduction

Dans ce chapitre, nous définissons les éléments fondamentaux de la théorie des graphes. Nous commençons par la notion et la définition mathématique de graphe, ensuite les concepts essentiels tels que les graphes orientés et non orientés, les sommets, les arêtes, les arcs, le voisinage et les degrés. Les principaux graphes particuliers sont également présentés. Enfin, nous étudions les différentes représentations non graphiques des graphes, notamment les matrices d'adjacence et d'incidence et les listes d'adjacence, qui jouent un rôle essentiel dans le traitement informatique.

2.2 Qu'est-ce que les graphes ?

Les graphes sont des diagrammes faits de lignes reliant certaines paires de points. Ils sont des représentations de situations réelles. Ils peuvent refléter des modèles sur lesquels différents types de problèmes peuvent être étudiés et résolus. Sur de tels graphes, nous nous intéressons principalement au fait que deux entités sont reliées ou non. Et il est impératif dans de telles représentations que les entités soient de la même nature, et que les liaisons le soient également.

2.3 Que peut-on modéliser avec les graphes ?

Les graphes peuvent modéliser toute situation où nous avons besoin de représenter des entités interconnectées entre-elles. Il peut s'agir par exemple :

- *des relations d'amitiés où les entités sont les personnes et les liaisons représentent l'existence d'une relation d'amitié.*
- *des réseaux routiers où les entités représentent les villes, et les liaisons représentent les routes.*
- *des réseaux informatiques où les machines sont représentées par les entités et les liaisons représentent les connexions physiques.*
- *des pages web où les sommets sont les pages et les liaisons sont les hyperliens d'une page à une autre.*

2.4 Définition mathématique des graphes

Un graphe G est constitué :

1 – d'un ensemble fini V de points appelés sommets ou nœuds, tel que :

$V = \{v_1, v_2, \ldots, v_n\}$, $v_1, v_2, \ldots, v_n$ *étant les nœuds du graphe G*

2 – d'un ensemble fini E de liaisons ; où chaque liaison relie deux sommets appelés ses extrémités, tel que : $E = \{e_1, e_2, \ldots, e_m\}$, $e_1, e_2, \ldots, e_m$ *étant les liaisons du graphe G*

2.5 Notions de base sur les graphes

2.5.1 Les graphes non orientés

Lorsque la direction de la liaison est sans importance, elle n'est pas mentionnée sur la ligne/courbe qui est appelée dans ce cas ***arête***, et le graphe est dit ***non-orienté.***

Une arête u définie par ses 2 extrémités xi et xj est représentée par la paire : **u = {xi,xj}**, et graphiquement par la figure suivante (ligne/courbe) :

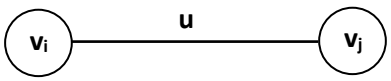


Figure 2.1 : Illustration d'une arête entre 2 sommets

Une boucle dans un graphe non-orienté est une arête où les 2 extrémités coïncident. Il s'agit d'une arête {xi, xi}. La Figure 2.2 illustre une arête en boucle.

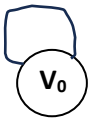


Figure 2.2 : une arête en boucle

Un graphe non-orienté est un diagramme défini par un ensemble fini de sommets et un ensemble fini d'arêtes. Le graphe de la figure 2.3 est défini par G= (V, E) où V est l'ensemble des sommets de G, et E l'ensemble des arêtes de G.

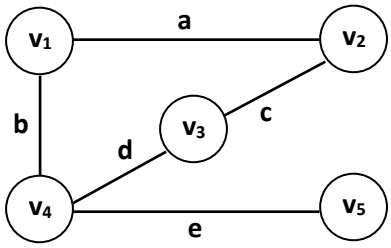


Figure 2.3 : un graphe non orienté

Nous avons alors :

- V= {v_1,v_2,v_3,v_4,v_5},
- E = {**a**={v_1,v_2}, **b**={ v_1,v_4}, **c**={ v_2,v_3}, **d**={ v_3,v_4}, **e**={ v_4,v_5}}.

Remarque : dans un graphe non orienté, une arête est représentée par ses 2 extrémités sans considération de l'ordre. Ainsi, dans la figure ci-dessus, nous avons : **a** = {v_1,v_2} = {v_2,v_1}.

Dans le dessin d'un graphe, la disposition des sommets n'est pas unique. Plusieurs illustrations sont possibles pour apporter la même information sur les sommets et leurs interconnexions. Ainsi, les graphes G et G' de la figure 2.4 sont les mêmes.

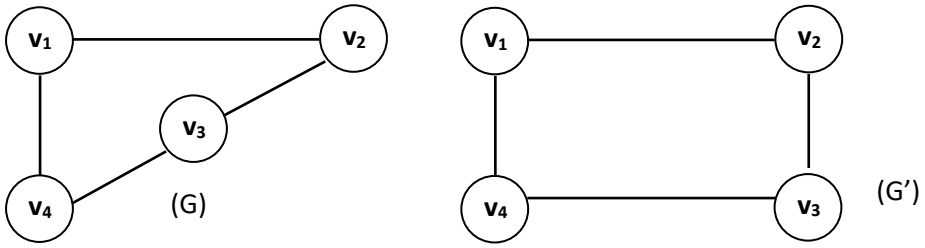


Figure 2.4 : Deux Graphes équivalents

2.5.2 Les graphes orientés

Dans beaucoup d'applications, les relations entre éléments respectent une ***direction***. Un élément x peut être en relation 'R' avec un autre élément y ***sans que*** y soit nécessairement en relation (la même 'R') avec x. En représentant cette notion dans les graphes, nous parlons de: ***graphe orienté.***

Un graphe orienté G(V, A) est un graphe où chaque liaison **est orientée** par une flèche et est appelée «arc ». '***V***' (pour vertex) étant l'ensemble des sommets et '***A***' (pour arc) étant l'ensemble de ses arcs. Un arc orienté va d'une extrémité ***initiale*** à une extrémité ***finale***, ou encore d'un sommet de ***départ*** vers un sommet ***d'arrivée.***

Un arc u qui part de v_i à v_j est représenté par le couple : **$u = (v_i, v_j)$** et graphiquement par la figure 2.5 (ligne/courbe) :

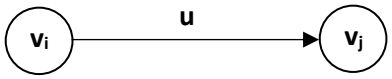


Figure 2.5 : Illustration d'un arc entre 2 sommets

Une Boucle dans un graphe orienté est un arc qui relie un sommet à lui-même (les 2 extrémités coïncident). Il s'agit d'un arc (xi, xi). La Figure 2.6 illustre un arc en boucle.

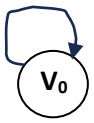


Figure 2.6 : un arc en boucle

2.5.3 Voisinage, Prédécesseurs, Successeurs

Nous adopterons le vocabulaire suivant :

- Si une arête/arc *e* part d'un sommet *v*, ou y arrive, on dit que *e* est ***incidente*** à *v*,
- Deux sommets sont dits ***adjacents*** s'ils sont reliés par une arête ou un arc,
- Deux arcs/arêtes sont dites ***adjacents*** s'ils ont un sommet en commun,
- Parfois, une information (chiffrée) est associée à l'arc/arête, elle est dite : la valeur, l'étiquette ou le poids de l'arc/arête.
- Un graphe possédant des valeurs associées à ses arcs/arêtes est dit graphe ***valué*** ou ***pondéré***.
- Les voisins d'un sommet sont l'ensemble des sommets qui lui sont adjacents.

Dans les graphes orientés, la notion de successeurs et de prédécesseurs est utilisée. Soit un graphe orienté G(V, A). Les successeurs, les prédécesseurs et les voisins d'un sommet sont définis dans la table 2.1.

Table 2.1 : successeurs, prédécesseurs et voisins

Successeur	**Prédécesseur**
v_j est **successeur** de v_i si $(v_i, v_j) \in A$	v_j est **prédécesseur** de v_i si $(v_j, v_i) \in A$
l'ensemble des successeurs de v_i est noté $\mathbf{\Gamma^+(v_i)}$. (gamma+ de v_i)	l'ensemble des prédécesseurs de v_i est noté $\mathbf{\Gamma^-(v_i)}$
$\mathbf{\Gamma^+(v_i) = \{v_j \in V /(v_i, v_j) \in A\}}$	$\mathbf{\Gamma^-(v_i) = \{v_j \in V /(v_j, v_i) \in A\}}$
Les **voisins** d'un sommet x sont **l'union** de ses prédécesseurs et ses successeurs : $\mathbf{\Gamma(v_i) = \Gamma^+(v_i) \cup \Gamma^-(v_i)}$	

Remarque : dans certains ouvrages, nous trouvons la notation suivante :

- Succ(v_i) pour $\Gamma^+(v_i)$
- Pred(v_i) pour $\Gamma^-(v_i)$

Exemple : Soit V l'ensemble des sommets suivants : V = {v_1, v_2, v_3, v_4, v_5};
et soit les ensembles de successeurs suivants :

- $\Gamma^+(v_1) = \{v_2\}$;
- $\Gamma^+(v_2) = \{v_1, v_3\}$;
- $\Gamma^+(v_3) = \{v_4, v_5\}$;
- $\Gamma^+(v_4) = \{v_5\}$;
- $\Gamma^+(v_5) = \{v_1\}$.

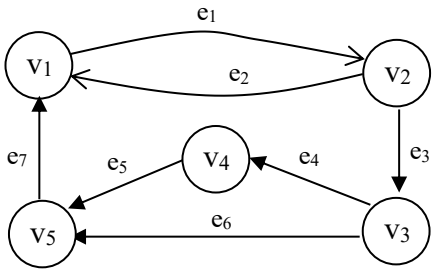


Figure *2.7* : graphe dessiné à partir de (V, Γ^+)

La figure 2.7 montre le graphe déterminé par (V, Γ^+).

2.6 Ordre, taille et degré dans les graphes

Nous considérons les concepts suivants dans un graphe G (V, E) :

a) L'***Ordre*** de G est le nombre de ses sommets, il est noté ***ordre(G)***, i.e Card(V) ou |V|.

b) La ***Taille*** de G est le nombre de ses arêtes/arcs, notée ***taille(G)***, i.e Card(E) ou |E|. Dans un graphe orienté, un arc (u,v) est différent de (v,u). Donc si les deux existent, ils comptent comme **2 arcs**, et ils augmentent la taille de 2.

c) Le ***Degré*** d'un sommet *v* est le nombre d'arêtes/arcs qui lui sont incidents. Il est noté ***deg(v)***. Une boucle est comptée 2 fois pour le sommet en question. Dans un graphe orienté, le concept de degrés est partagé en 2 types :

  • Demi degré intérieur noté $\boldsymbol{deg^-(v)}$: c'est le nombre d'arcs entrants vers ***v***

  • Demi degré extérieur noté $\boldsymbol{deg^+(v)}$: c'est le nombre d'arcs sortants de ***v***.

• Degré global : $\boldsymbol{deg\,(v) = deg^{+}(v) + deg^{-}(v)}$

d) Le ***Degré*** de G est le degré maximum de ses sommets, il est noté $\boldsymbol{\Delta(G)}$.

e) Un sommet dont le degré est égal à 1 est dit sommet ***pendant***

f) Un sommet dont le degré est égal à 0 est dit sommet ***isolé***

Illustration : Dans le graphe non orienté (G) de la figure 2.8, nous avons :
Ordre(G) = 6 ; *taille*(G) = 5 ; le sommet v_6 est *isolé*, le sommet v_5 est *pendant*.

Table 2.2 Degrés de sommets

Sommet v	v_1	v_2	v_3	v_4	v_5	v_6
Degré (v)	**2**	**2**	**2**	**3**	**1**	**0**

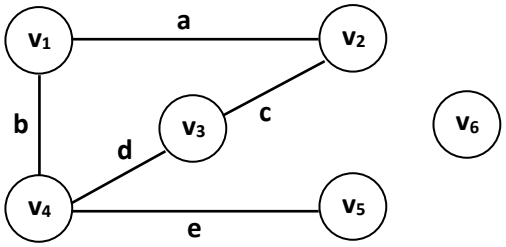


Figure 2.8 : sommet pendant, sommet isolé

La table 2.3 donne les demi-degrés des sommets du graphe orienté de la figure 2.7

Table 2.3 : demi-degrés d'un graphe orienté

	v_1	v_2	v_3	v_4	v_5	**total**
$\boldsymbol{d^{+}(v_i)}$	1	2	2	1	1	7
$\boldsymbol{d^{-}(v_i)}$	2	1	1	1	2	7
$\boldsymbol{d(v_i)}$	3	3	3	2	3	14

Remarque 1 **:** dans un graphe orienté, nous avons toujours : $\sum_{v \in V} d^{-}(v) = \sum_{v \in V} d^{+}(v)$

Donc systématiquement : La somme des degrés des sommets dans un graphe orienté est toujours paire.

Remarque 2 : dans un graphe non orienté, chaque arête est comptée doublement dans les degrés de ses 2 extrémités, ce qui donne impérativement que **la somme totale des degrés** égale à 2 fois le nombre d'arêtes et donc elle est **toujours paire**. D'où le théorème suivant :

Théorème : Pour tout graphe G=(V, E) de n sommets :

$$\sum_{v_i \in V}^{i=1,n} deg(v_i) = 2 * taille(G)$$

Etant donné ce théorème, nous aurons alors :

$$\sum_{v_i \in V, d(v_i)\ pair}^{i=1,n} d(v_i) + \sum_{v_i \in V, d(v_i)\ impair}^{i=1,n} d(v_i) = 2 * taille(G)$$

Et comme la somme des degrés pairs est systématiquement paire, la somme des degrés impairs sera aussi paire, cela implique que :

Dans tout graphe**, le nombre des sommets** de degrés impairs est forcément pair

2.7 Quelques graphes particuliers

2.7.1 Graphes non orientés particuliers

Les graphes non orientés présentent parfois quelques caractéristiques particulières :

a) **Un graphe simple** est un graphe dans lequel il n'y pas de boucle et il y a au maximum une seule arête entre deux sommets. La figure 2.9 donne un exemple de graphe *simple*.

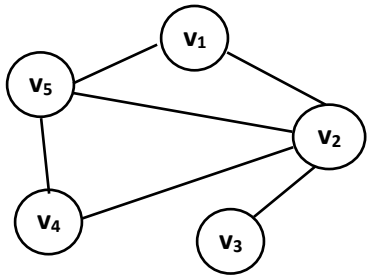


Figure 2.9 : un graphe non-orienté simple

Propriété : Dans un graphe **simple**, il existe toujours deux sommets ayant le même degré

Preuve : Il y a **n** valeurs prises (les valeurs de degrés) dans un ensemble de **n-1 valeurs** (de 1 à n -1). Donc forcément il y a au moins une valeur qui se répète !

b) **Un multi-graphe** est un graphe où il peut exister des boucles et/ou plusieurs arêtes entre 2 sommets. La figure 2.10 donne un exemple de *multigraphe*.

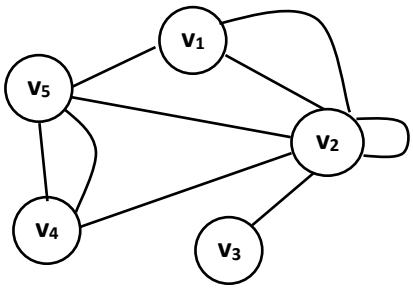


Figure 2.10 : un graphe multigraphe

c) Un graphe est dit **régulier** si tous ces sommets ont **le même degré**. Si ce degré est **k**, alors le graphe est dit **k-régulier.** La figure 2.11 donne un graphe *3-régulier*.

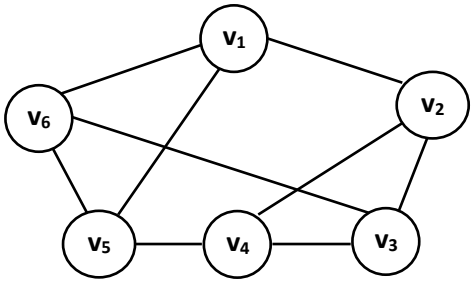


Figure 2.11 : un graphe 3-régulier

d) **Un graphe complet** est un graphe simple dans lequel chaque sommet est relié à tous les autres sommets. Le graphe complet contenant n sommets est noté K_n. La taille du graphe K_n est **n*(n - 1) / 2,** (la division par 2 a lieu car la même arête est considérée 2 fois). La figure 2.12 donne un graphe K_5.

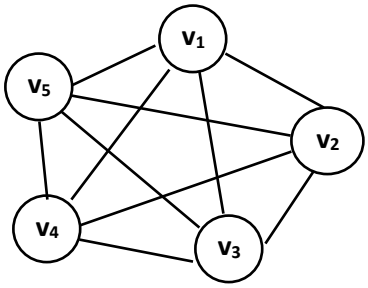


Figure 2.12 : un graphe K_5

e) **Un graphe biparti** est un graphe qu'on peut diviser en 2 ensembles de sommets V et V', tel que chaque arête relie un sommet de V à un sommet de V', et il n'y a pas d'arêtes à l'intérieur du même ensemble. Dans le graphe de la figure 2.13, nous remarquons que chaque arête relie un sommet de l'ensemble $V=\{v_1, v_3, v_4, v_6\}$ à un sommet de l'ensemble $V' =\{v_2,v_5\}$. Ce graphe est dit *biparti*. Le graphe de la figure 2.14 l'est également, avec $V=\{v_1, v_4\}$et $V'=\{v_2, v_3\}$.

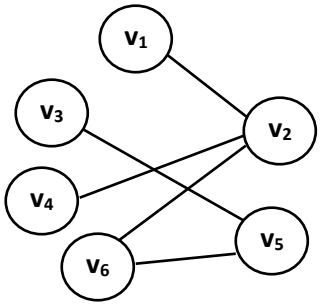


Figure 2.13 : un graphe biparti

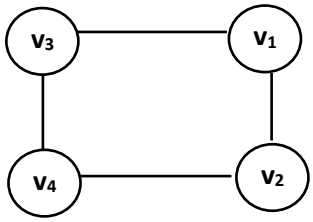


Figure 2.14 : un graphe biparti

f) **Un graphe biparti** est dit **complet,** si tout sommet de l'ensemble V est relié à tous les sommets de l'ensemble V'. Il est noté par $K_{m,n}$ où m est le nombre de sommets de V et n est le nombre de sommets de V'. Ce graphe est **régulier** si m=n, **bi-régulier** si tous les sommets de V ont le même degré et tous les sommets de V' ont le même degré. Le graphe de la figure 2.14 est **biparti complet et 2-régulier** alors que le graphe de la figure 2.15 est : **biparti complet $K_{4,2}$, et (2,4)-régulier.**

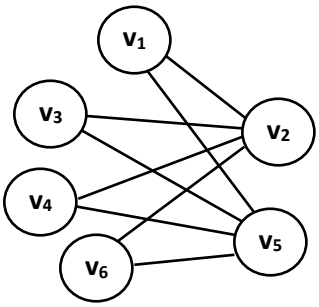


Figure 2.15 : un graphe biparti, complet $K_{4,2}$, (2,4)-régulier

g) **Un sous graphe** G'(V',E') du graphe G(V,E) est un graphe tel que : V'⊂ V et E' ⊂ E

h) **Un graphe partiel** G'(V,E') du graphe G(V,E) est un graphe tel que : E' ⊂ E

i) **Le graphe adjoint** L(G) (line graph) du graphe G : est le graphe obtenu de la manière suivante :

- Chaque sommet de $L(G)$ représente une arête de G,
- Deux sommets de $L(G)$ sont adjacents si et seulement si les arêtes correspondantes ont un sommet commun dans G

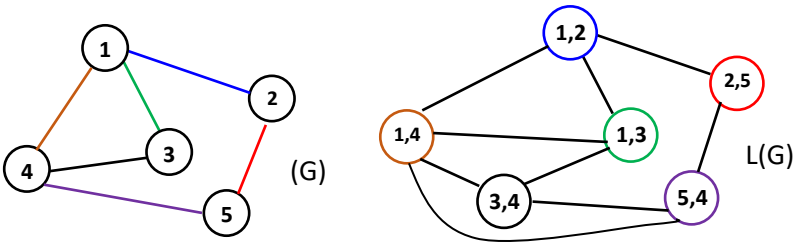


Figure 2.16 : graphe (G) et son graphe adjoint L(G)

2.7.2 Graphes orientés particuliers

Les graphes orientés peuvent aussi présenter parfois quelques caractéristiques particulières similaires à celles des graphes non orientés :

a) Un graphe élémentaire est un graphe orienté dans lequel il n'y pas de boucle et il y a au maximum un seul arc qui part d'un sommet v_i vers un sommet v_j. La figure 2.17 donne un exemple de graphe *élémentaire*.

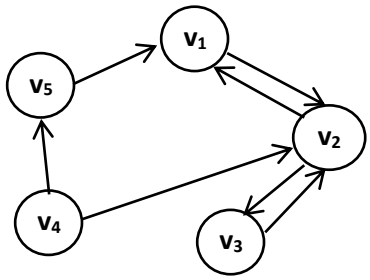


Figure 2.17 : un graphe orienté élémentaire

b) Un p-graphe est un graphe orienté où il peut exister au maximum p arcs ayant le même sommet de départ et le même sommet d'arrivée. C'est une **généralisation d'un multigraphe orienté**, avec une borne sur la multiplicité des arcs. Les figures 2.18, 2.19 et 2.20 illustrent des exemples de p-graphes.

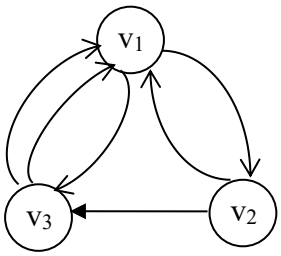


Figure *2.18* : un 2-graphe

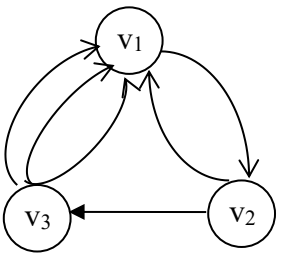


Figure *2.19* : un 3-graphe

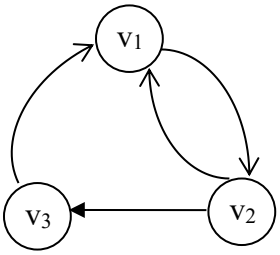


Figure *2.20* : un 1-graphe

c) Le graphe complet : un graphe orienté est dit complet (dans la version la plus courante), **si, pour *toute paire de sommets distincts* v_i et v_j, il existe *un arc dans chaque sens*** : (vi,vj) et (vj,vi) ; Cela signifie qu'entre deux sommets distincts, la relation est **bidirectionnelle**. Mais dans certains contextes particuliers, il peut être mentionné que le graphe est dit complet si au moins un arc existe entre chaque paire de sommets distincts. Cela veut dire que, pour chaque paire, au moins un arc les reliant est présent.

2.8 La représentation non graphique des graphes

Un certain nombre de représentations non graphiques existent pour décrire les graphes. Nous distinguons principalement :

- **La matrice d'adjacence** (sommet-sommet)
- **La matrice d'incidence** (sommets-arête)
- **La liste d'adjacence**

a) **La matrice d'adjacence**

La matrice d'adjacence *M* fait correspondre les sommets origine des arêtes/arcs placés en ligne dans la matrice aux sommets destination placés en colonne. C'est une matrice carrée *n x n* où n=ordre (G).

$$M_{ij} = \begin{cases} k, & \textbf{s'il y a k arêtes/arcs allant de i vers j} \\ 0, & \textbf{sinon} \end{cases}$$

Dans un graphe non-orienté, elle est symétrique. S'il y'a au maximum une arête entre deux sommets, elle devient binaire. Une boucle est représentée par un '1' sur la diagonale de la matrice. Dans les 2 exemples suivants : La matrice M1 est la matrice d'adjacence du graphe de la figure 2.21, et la matrice M2 est celle du graphe 2.22.

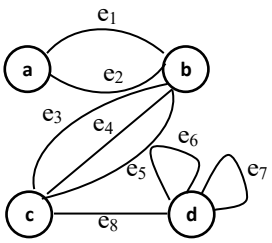


Figure 2.21 : multigraphe

$$M1 = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 2 & 0 & 3 & 0 \\ 0 & 3 & 0 & 1 \\ 0 & 0 & 1 & 2 \end{pmatrix}$$

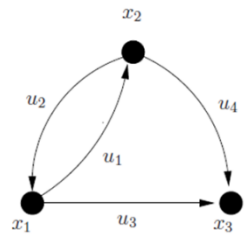


Figure 2.22 : graphe orienté

$$M2 = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Remarque1 : certains auteurs préfèrent *ignorer le nombre de liaisons* (dans les multigraphes) entre 2 sommets et se contentent de mentionner s'il y a liaison (1) ou pas (0). Et donc elle devient forcément binaire. Dans tous les cas, la décision du choix dépend toujours du besoin et du contexte. Dans notre cours, nous adoptons la formule donnée ci-dessus.

Remarque2 *:* Nous pouvons trouver au maximum n! matrices d'adjacence pour notre graphe, selon l'ordre dans lequel nous considérons les sommets.

b) Matrice d'adjacence pour les graphes pondérés

Lorsque le graphe est pondéré, donc ses arcs portent des valeurs qui peuvent représenter des distances, des coûts, du temps… Sa matrice d'adjacence portera ces pondérations à l'intersection des sommets concernés.

$$\boldsymbol{M_{ij}} = \begin{cases} \boldsymbol{valeur(i,j)}, & \textbf{si } (\mathbf{i},\mathbf{j}) \in \boldsymbol{A} \\ \infty, & \boldsymbol{sinon} \end{cases}$$

Exemple : La matrice M suivante est la matrice d'adjacence du graphe pondéré de la figure 2.23.

$$M = \begin{pmatrix} \infty & 6 & \infty & 2 & \infty & \infty \\ \infty & \infty & \infty & 2 & \infty & \infty \\ \infty & 4 & \infty & \infty & \infty & 1 \\ \infty & \infty & 3 & \infty & \infty & \infty \\ \infty & 2 & 8 & \infty & \infty & 9 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

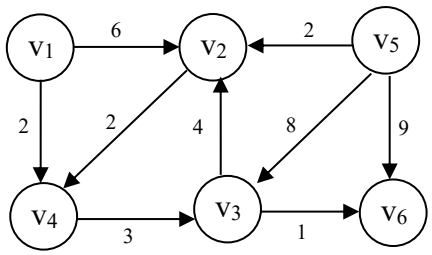


Figure *2.23* : graphe orienté pondéré

c) La matrice d'incidence

La matrice d'incidence *A* d'un graphe G est *une matrice n x p*, où n=ordre(G), et m= taille(G). Elle fait correspondre les sommets placés en ligne à des arêtes/arcs placés en colonne (il faudra les nommer). Dans le cas d'un graphe non orienté, son principe est le suivant :

- $\boldsymbol{si\ e = (i,j) \in E, alors} \begin{cases} \boldsymbol{A_{ie} = 1} \\ \boldsymbol{A_{je} = 1} \end{cases}$
- $\boldsymbol{si\ e = (i,i) \in E, alors\ A_{ie} = 2}$

Remarquons que dans une matrice d'incidence, nous mettons 2 pour une boucle.

La matrice d'incidence du graphe de la figure 2.21 peut être représentée de 2 façons :

$$A = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 2 & 2 & 1 \end{pmatrix}$$ ou bien

Table 2.4. Matrice d'incidence équivalente

	e_1	e_2	e_3	e_4	e_5	e_6	e_7	e_8
a	1	1	0	0	0	0	0	0
b	1	1	1	1	1	0	0	0
c	0	0	1	1	1	0	0	1
d	0	0	0	0	0	2	2	1

Quand le graphe est orienté, l'intersection de l'arc et le sommet de départ prend la valeur -1, alors que celle du sommet d'arrivée garde la valeur 1. La table 2.5 donne la matrice d'incidence du graphe 2.22.

$$\boldsymbol{si\ e = (i,j) \in E, alors} \begin{cases} \boldsymbol{A_{ie} = -1} \\ \boldsymbol{A_{je} = 1} \end{cases}$$

Table 2.5 Matrice d'incidence

	e_1	e_2	e_3	e_4
x1	-1	+1	-1	0
x2	+1	-1	0	-1
x3	0	0	+1	+1

d) La liste d'adjacence

La liste d'adjacence est une représentation par une liste chaînée de tous les adjacents de chacun des sommets. Nous aurons ainsi autant de chaines que de sommets. Le graphe de la figure 2.24 est représenté par la liste d'adjacence ci-dessous.

Table 2.6 : liste d'adjacence

a	a	b	c	d
b	a	c		
c	a	b	d	
d	a	c	d	

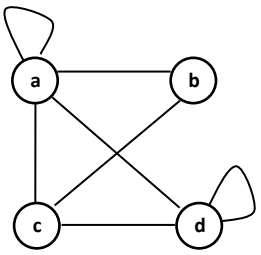


Figure 2.24 : multigraphe

2.9 Conclusion

Dans ce chapitre, nous avons établi les bases conceptuelles et mathématiques de la théorie des graphes. Nous avons présenté les notions essentielles relatives aux sommets, aux arêtes, aux arcs, aux relations d'adjacence ainsi qu'aux concepts d'ordre, de taille et de degré. Et tout cela après avoir défini auparavant ce qu'est un graphe et les différents contextes dans lesquels il peut être utilisé comme outil de modélisation. Les graphes orientés et non orientés ainsi que plusieurs familles de graphes particuliers, ont été présentés. Enfin, nous avons défini les principales représentations non graphiques des graphes, notamment les matrices d'adjacence, d'incidence et les listes d'adjacence, qui constituent les bases des implémentations informatiques et des algorithmes de traitement des graphes.

Série de TD n°1

Exercice 1

a) Démontrez que le nombre de sommets ayant un degré impair est pair.
b) peut-on trouver un groupe de 4 personnes tel que chaque personne est amie avec exactement trois autres personnes ?
c) peut-on trouver un groupe de cinq personnes tel que chaque personne est amie avec exactement trois autres personnes ?

Exercice 2

2.1. Étant donné un groupe de 10 personnes, le tableau suivant indique les paires de personnes qui ont une relation d'amitié.

a) Représentez cette situation par un graphe G.
b) Donnez un sommet pendant et un sommet isolé de G.
c) Vérifiez qu'il existe au moins deux personnes ayant le même nombre d'amis.

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

2.2. A un intervalle minime de temps, une application de chat, a enregistré les contacts suivants entre 5 clients. Illustrez cette situation par un graphe.

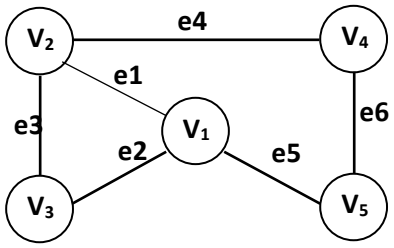
C_1 a contacté C_2 ; C_1 a contacté C_4 ; C_2 a contacté C_3 ; C_2 a contacté C_4 ; C_3 a contacté C_5 ; et C_4 a contacté C_3.

Exercice 3

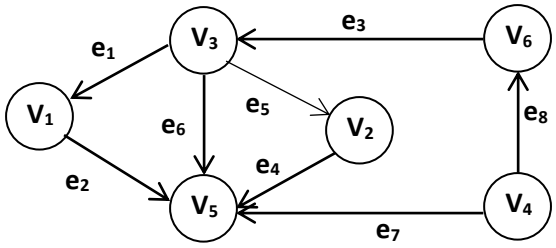
a) Construisez un graphe non orienté ayant au moins deux sommets et tel que tous les sommets ont des degrés distincts.
b) Qu'en déduisez-vous ?

Exercice 4

a) Donnez la matrice d'adjacence et la matrice d'incidence du graphe non orienté suivant :



b) Donnez la matrice d'adjacence et la matrice d'incidence du graphe orienté suivant :



c) Représentez graphiquement les graphes associés aux matrices suivantes :

-1	-1	0	0	0	0
1	0	-1	1	0	0
0	1	1	0	-1	0
0	0	0	-1	0	1
0	0	0	0	1	-1

A

1	1	-1	0	0	0	1
0	0	0	1	0	-1	0
-1	0	0	0	1	0	0
0	0	0	0	-1	1	-1
0	-1	1	-1	0	0	0

B

d) Représentez graphiquement le graphe associé à la matrice suivante :

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

Chapitre 3
Cheminements dans les graphes

3.1 Introduction

Dans ce chapitre, nous aborderons successivement les notions de chaînes, cycles et cocycles, les propriétés de connexité, les différents types de graphes particuliers ainsi que les méthodes de parcours en profondeur et en largeur. Nous étudierons également les propriétés spécifiques des graphes orientés, la matrice de fermeture transitive, les composantes connexes et fortement connexes, ainsi que les techniques de décomposition en niveaux.

3.2 Chaîne, Cycle, cocycle, base de cycle, base de cocycle

Dans un graphe non orienté, nous considérons le vocabulaire et les concepts suivants :

a) **Une chaîne** dans un graphe non-orienté et simple, est une suite des sommets consécutifs, : **$<v_1, v_2,..v_k>$**, tel qu'il existe une arête entre **v_i et v_{i+1} /i=1,k-1**. Puisque les arêtes ne sont pas orientées, alors les chaînes **$<v_1,v_2,\ldots,v_k>$** et **$<v_k,v_{k-1},\ldots,v_1>$** sont les mêmes.

   Si le graphe est multigraphe, il faudra préciser les arêtes concernées par le cheminement entre les sommets. Dans une chaîne, les sommets et les arêtes peuvent y apparaître plusieurs fois !

b) Une chaine **élémentaire** est une chaîne ne passant pas deux fois par le même sommet.
c) Une chaine **simple** est une chaîne ne passant pas deux fois par la même arête.
d) Une chaîne est dite **fermée**, si ses deux extrémités (sommet de départ et sommet d'arrivée) coïncident !
e) Un **cycle** est une chaine fermée et simple.
f) Un **cocycle** est un ensemble d'arêtes dont la suppression Sépare le graphe en deux parties. Un cocycle correspond à une **coupe**.
g) Une **base de cycles** est un ensemble minimal de cycles : Linéairement indépendants, Qui engendrent tous les cycles du graphe
h) Une **base de cocycles** est un ensemble minimal de cocycles : Linéairement indépendants Qui engendrent tous les cocycles
i) La **longueur** d'une chaine est définie par le nombre d'arêtes qui la composent, et c'est toujours le **nombre de ses sommets – 1**.
j) Une **distance** entre 2 sommets est la longueur de la plus courte chaîne entre eux.
k) Un **diamètre** entre 2 sommets est la longueur de la plus longue chaine entre eux.

Exemple : Dans le graphe de la figure 3.1, nous avons :

- La chaîne **$<v_1,v_3,v_2,v_5>$** est de **longueur** = 3
- La chaîne **$<v_1,v_3,v_4,v_1>$** est un **cycle**
- La chaîne **$<v_3,v_2,v_5,v_3>$** est un autre **cycle**
- La **distance** entre v_1 et v_5 = 2, en considérant la chaîne **$<v_1,v_3,v_5>$**
- Le **diamètre** entre v_1 et v_5 = 4, en considérant la chaîne **$<v_1,v_4,v_3,v_2,v_5>$**

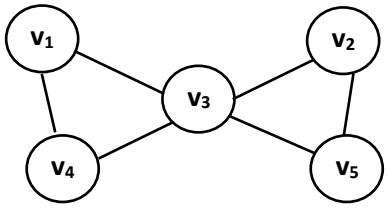


Figure 3.1 : graphe simple

3.3 La connexité dans les graphes non-orientés

Un graphe non orienté est dit **connexe** s'il comporte une chaîne entre toute paire de sommets qui le composent. Il est dit **non-connexe**, s'il comporte au moins deux sommets entre lesquels il n'existe pas de chaîne.

Un graphe qui n'est pas connexe est l'union de deux ou plusieurs ***sous-graphes connexes*** et ***disjoints***. Ces derniers sont les ***composantes connexes*** du graphe. Le graphe de la figure 3.2 n'est pas connexe, et il est composé de trois composantes connexes :

$C_1=\{v_1,v_3,v_4\}$; $C_2=\{v_2,v_5\}$; $C_3=\{v_6\}$;

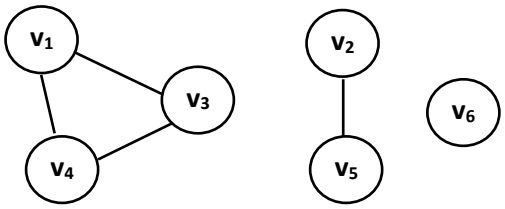


Figure 3.2 : graphe non-connexe composé de 3 composantes connexes

3.4 Les types particuliers des graphes non orientés

3.4.1 Les graphes planaires

Sachant que le dessin d'une arête peut se faire par n'importe quelle courbe continue, qu'elle soit droite ou non, beaucoup de graphes peuvent être dessinés ou redessinés en évitant que leurs arrêtes se croisent. Par exemple, étant donné :

- un ensemble de sommets $V=\{v_1,v_2,v_3,v_4,v_5\}$ et
- un ensemble d'arêtes $E=\{\{v_1,v_2\}, \{v_1,v_3\}, \{v_2,v_3\}, \{v_2,v_4\}, \{v_3,v_4\}, \{v_2,v_5\}\}$

Pour ces mêmes données, nous pouvons tracer les illustrations graphiques suivantes :

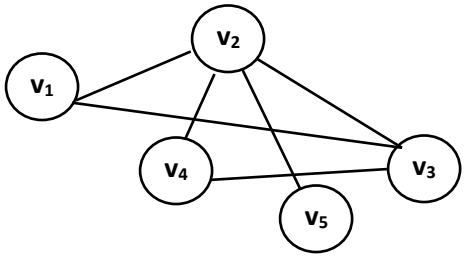


Figure 3.3 : représentation 1

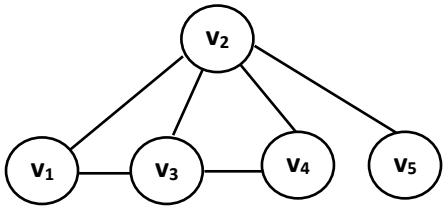


Figure 3.4 : représentation 2

Nous remarquons bien que le graphe a pu être redessiné de façon à ce que ses arêtes ne se croisent pas. Nous appelons un tel graphe « **planaire** ». Dans la 2ème représentation il est dit « **planaire topologique** ».

Soit le graphe complet K_4 de la figure 3.5. Est-il planaire ?

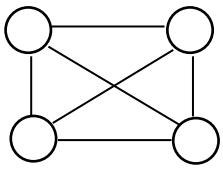


Figure 3.5 : graphe complet K_4

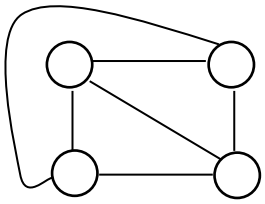


Figure 3.6 : graphe planaire

Réponse : Oui, puisque nous pouvons le retracer tel que dans la figure 3.6.

Définition : Dans une représentation planaire d'un graphe, les zones délimitées par des arêtes qui les entourent sont appelées des « faces ». La face extérieure (qui est infinie) est considérée aussi parmi les faces d'un graphe planaire. Dans le graphe de la figure 3.7, nous avons 4 faces. La 4ème étant la face infinie. Le degré d'une face est le nombre d'arêtes qui l'entourent.

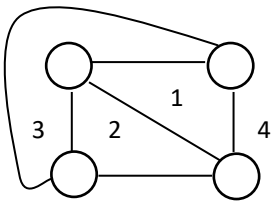


Figure 3.7 : les faces dans un graphe planaire topologique

> **Théorème** : La somme des degrés de toutes les faces d'un graphe planaire est égale au double de son ordre

.

Formule d'Euler (dite de planarité) : Dans un graphe planaire, la relation suivante est toujours vérifiée entre l'ordre du graphe (n), sa taille (m) et le nombre de ses faces (f).

$$n - m + f = 2 \quad donc \quad f = m - n + 2$$

3.4.2 Les graphes Eulériens

Dans le 18ème siècle, il existait en Russie, une ville qui s'appelait « konigsberg » construite autour de deux îles situées sur un fleuve. Six ponts reliaient les 2 rives du fleuve à travers les 2 îles, et un septième pont reliait les deux îles entre elles-mêmes. La figure suivante en est une illustration.

Figure 3.8 Les 7 ponts de Konigsberg

a) **Le problème des 7 ponts**

Ce problème avait vu le jour en **1735** lorsque les habitants de cette ville se demandaient « Existe-t-il **une promenade** dans les rues de Königsberg permettant :

- à partir **d'un quartier de départ au choix**,
- de passer **une et une seule fois par chaque pont**,
- et de revenir à **son quartier de départ**,

Les gens essayaient différents chemins et ne parvenaient pas à en trouver un qui les fasse passer exactement une fois sur chaque pont. Mais la multitude des chemins possibles, les gardait optimistes jusqu'à ce que Leonhard Euler mathématicien célèbre à l'époque résolve le problème dans un article écrit en 1736.

La remarque importante d'Euler Leonhard, était que :

- Pour traverser chaque pont une et une seule fois, il faut **autant de ponts pour quitter** un quartier **que pour y revenir** (sinon l'on ne reviendrait jamais au point de départ) ;
- il faut donc que chaque quartier comporte un **nombre pair** de ponts.

- Or, dans la ville de Königsberg, tous les quartiers sont reliés aux autres par un nombre **impair** de ponts.
- Par conséquent le problème de la promenade par les sept ponts n'a **pas de solution !**

Et si on veut faire **une promenade qui** passe **une et une seule fois par chaque pont**, mais sans revenir à **son quartier de départ** ? Cela sous entends que les quartiers de départ et d'arrivée soient différents ! Une telle promenade n'existe pas aussi car cela suppose que seuls les quartiers de départ et d'arrivée soient reliés à un nombre impair de ponts, or ce n'est pas le cas !!!

b) Chaines, cycles et graphes eulériens

- Une **chaine eulérienne** d'un graphe G est une chaine simple qui :
  - Contient **toutes les arêtes** de G et,
  - Qui passe **exactement une seule fois** par **chacune des arêtes**
- Un **cycle eulérien** de G est une chaine eulérienne **fermée**
- Un graphe est dit **eulérien** s'il possède **un cycle eulérien**
- Un graphe est dit **semi-eulérien** s'il ne possède que **des chaines eulériennes**

Plus simplement, on peut dire qu'un graphe est eulérien si on peut **le dessiner sans lever la main** et en passant une seule fois par chaque arête.

Théorème : Un graphe **connexe G** tel que **m ≥ 1** est **eulérien si et seulement si tous ses sommets sont de degrés pairs**

Un graphe connexe G est **semi-eulérien si et seulement si tous ses sommets ont des degrés pairs sauf 2 éventuellement**

Application : Les graphes suivants sont-ils eulériens ou semi-eulériens ?

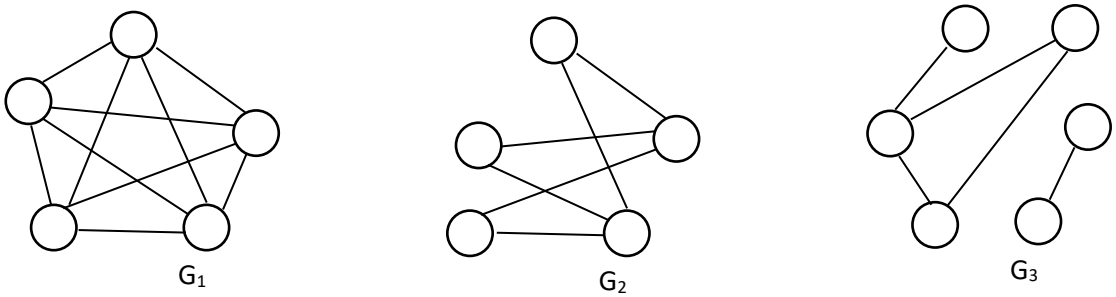


Figure 3.9 : Graphes non orientés

- G1 : Tous les degrés de sommets sont pairs, donc il est eulérien
- G2 : Tous les degrés de sommets sont pairs, sauf 2, donc il est semi-eulérien
- G3 : Le graphe est non-connexe, donc il ne peut pas être eulérien ni semi-eulérien

c) La construction algorithmique d'une chaine eulérienne

Définition : **Un pont** est une arête telle que si elle est supprimée, certaines parties du graphe ne seront plus accessibles. La fonction suivante détermine si une arête passée en paramètre est un pont ou non.

```
Fonction Pont (u,v),
S'il y a un seul sommet adjacent à u (qui est v) alors Pont = faux
Sinon
    Calculer le nombre de sommets accessibles depuis u, soit le nombre a
    Enlever l'arête (u,v) puis calculer le nombre de sommets accessibles depuis u,
    soit le nombre b
    Si a > b alors Pont = vrai
    Sinon Pont = faux
```

Lorsque le graphe est au moins semi eulérien, nous pouvons utiliser **l'algorithme de Fleury** est utilisé pour la construction d'une chaîne eulérienne.

```
si le graphe est eulérien alors
    commencer par n'importe quel sommet
sinon
    commencer par un sommet dont le degré est impair,
    choisir le prochain sommet en fonction des arêtes incidentes non encore visitées.
    Si l'on doit choisir entre un pont et un non-pont alors on choisit un non-pont.
    Lorsqu'une arête est sélectionnée, alors elle est enlevée du graphe.
s'arrêter lorsqu'il n'y a plus d'arêtes.
```

3.4.3 Les graphes Hamiltoniens

a) **Définitions** : Dans un graphe G,

- Une chaîne est dite **Hamiltonienne** si elle permet de **passer une seule fois par chacun des sommets** de G.
- Un **cycle hamiltonien** est une chaîne hamiltonienne fermée
- **Un graphe hamiltonien** est un graphe qui contient **un cycle hamiltonien**

- Un graphe **non-hamiltonien** contenant **une chaîne hamiltonienne**, est dit **semi-hamiltonien**

Propriétés

- Un graphe possédant un sommet de degré 1 ne peut pas être hamiltonien
- Si, dans un graphe, un sommet est de degré 2, alors les deux arêtes ; incidentes à ce sommet doivent faire partie du cycle hamiltonien ;
- Les graphes complets K_n sont hamiltoniens.

Le **théorème** suivant définit **une condition suffisante mais pas nécessaire** pour un graphe hamiltonien.

Théorème

Pour un graphe G simple et d'ordre n > 3, si nous avons : **d(x) + d(y) ≥ n** pour toute paire (x, y) de sommets non adjacents, alors G est hamiltonien.

On peut dériver une autre caractéristique (plus forte mais plus simple à vérifier) pour un graphe hamiltonien :

Pour n > 3, Si pour tout sommet x on a **d(x) ≥ n/2,** alors le graphe est **hamiltonien.**

Exemple : Soit le graphe enveloppe de la figure 3.10 Est-il hamiltonien ?

Nous allons le tester selon le théorème ci-dessus. Donc nous vérifions la somme des degrés de toute paire de sommets non-adjacents.

(e,d) : d(e) + d(d) = 2 + 3 = 5 ≥ **n =5**

(e,c) : d(e) + d(c) = 2 + 3 = 5 ≥ **n =5**

Donc le graphe est hamiltonien.

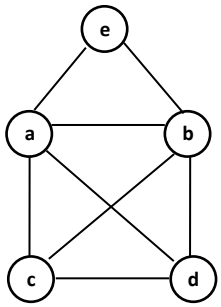


Figure 3.10 : graphe enveloppe

b) Application des graphes hamiltoniens

Les graphes hamiltoniens possèdent beaucoup d'applications, notamment dans le problème du voyageur de commerce où un commerçant doit visiter toutes les villes d'un certain réseau une seule fois tout en minimisant le coût des déplacements. De manière générale : dans toute situation où nous avons besoin d'atteindre des points spécifiques une et une seule fois, nous sommes dans le besoin de chercher un graphe hamiltonien.

3.5 Chemins, circuits, graphes sans circuits, source et puit

Nous étudions principalement dans cette section les **1-*graphes*** (sans boucle et sans arcs multiples entre les sommets).

Les notions des *chaînes* et *cycles* des graphes non-orientés correspondent ici aux notions respectives de ***chemins*** et ***circuits***. Nous adopterons le vocabulaire et les concepts suivants dans ce cours :

- Un chemin de v_1 à v_k est une séquence de sommets $<v_1,v_2,..v_k>$, tel qu'il existe un arc entre v_i et v_{i+1} /i=1,k-1. Puisque les arcs sont orientés, alors si $<v_1,v_2, .., v_k>$ est un chemin, alors $<v_k,v_{k-1},\ldots,v_1>$ ne l'est pas forcément.
- Un chemin est dit ***élémentaire*** si chaque sommet qui le compose y apparait une seule fois.
- La **longueur d'un chemin** c'est le nombre d'arcs qui le composent.
- Si le sommet de départ coïncide avec le sommet d'arrivée dans un chemin, nous parlons de ***circuit***.
- Une boucle est un circuit de longueur 1.
- Un sommet x est dit source si $deg^-(x)=0$; c'est-à-dire qu'il n'a aucun arc entrant vers lui. Une source représente un point de départ.
- Un sommet x est dit **puit** si $deg^+(x)=0$; c'est-à-dire qu'il n'a aucun arc sortant de lui. Un puit représente un point final.
- Un **graphe sans circuit** est un graphe orienté qui **ne contient aucun circuit**. On l'appelle aussi : **DAG** (Directed Acyclic Graph). Un DAG possède plusieurs propriétés fondamentales
    1. **Il admet un ordre topologique :** On peut classer les sommets dans un ordre linéaire tel que : Si $(u,v) \in E$ alors u apparaît avant v dans l'ordre, ce qui est impossible s'il existe un circuit.
    2. **Il possède au moins une source** : x est un sommet source

3.6 Quelques propriétés des graphes orientés

Soit G(V, A) un graphe orienté. Nous pouvons avoir les propriétés suivantes :

a) **Réflexivité :**

- G est ***réflexif***, si $\forall v_i \in V, (v_i, v_i) \in A$
- G est ***irréflexif***, si $\forall v_i \in V, (v_i, v_i) \notin A$

b) Symétrie

- G est ***symétrique***, si $\forall\ v_i, v_j \in V, (v_i, v_j) \in A \Rightarrow (v_j, v_i) \in A$
- G est ***asymétrique***, si $\forall\ v_i, v_j \in V, (v_i, v_j) \in A \Rightarrow (v_j, v_i) \notin A$
- G est ***antisymétrique***, si $\forall v_i, v_j \in V, (v_i, v_j) \in A$ et $(v_j, v_i) \in A \Rightarrow v_i = v_j$

c) Transitivité

- G est ***transitif***, si $\forall\ v_i, v_j, v_k \in V$,
  - Si $(v_i, v_j) \in A$ et $(v_j, v_k) \in A \Rightarrow (v_i, v_k) \in A$

Remarques :

- G est asymétrique $\Rightarrow$ G est irréflexif (ne contient pas de boucles)
- G est asymétrique $\Rightarrow$ G est aussi antisymétrique

3.7 Matrice de fermeture transitive

La fermeture transitive d'un graphe orienté G, notée G^+, est le graphe orienté ayant :

- le même ensemble de sommets
- un arc (u,v) si et seulement si il existe un chemin de u vers v dans G

Autrement dit ***: On ajoute toutes les arêtes correspondant à l'accessibilité.***

Soit A la matrice d'adjacence du graphe. On note : ***A^k le produit matriciel classique.*** L'entrée (i,j)) de A^k donne le nombre de chemins de longueur k. Ainsi :

$A+A^2+A^3+\cdots+A^n$ permet de détecter l'accessibilité

La méthode classique pour calculer la fermeture transitive est : *l'Algorithme de Warshall, qui* repose sur la programmation dynamique. Son principe :

On construit progressivement la matrice d'accessibilité :

R^k (i,j) = il existe un chemin de i à j n'utilisant que les sommets {1,…,k} comme intermédiaires.

La formule de récurrence : $R^k(i,j)=R^{k-1}(i,j)$ permet d'obtenir la matrice d'accessibilité complète.

3.8 Le parcours (exploration) des graphes

3.8.1 Exploration profondeur

L'exploration en profondeur (DFS) est un algorithme qui permet de parcourir tous les sommets d'un graphe en allant le plus loin possible dans une branche avant de revenir en arrière. Donc on explore un voisin, puis un voisin de ce voisin, etc., jusqu'à blocage, puis on revient en arrière (backtracking). Ce-dessous l'algorithme correspondant à cette politique.

```
Algorithme1 : Parcours en profondeur DFS(G)
Données : graphe G, marque des sommets (initialisé à Faux),
père π des sommets (initialisée à null)
début
    pour chaque s sommet de G faire
      début
          début
              si s non marqué alors
              Visiter(s) ;
          fin
      fin
fin
```

```
Algorithme 2 : fonction récursive Visiter(u)
début
marque[u] ← Vrai ;
pour chaque v voisin de u faire
      début
            début
                si v non marqué alors
                π[v ] ← u ;
                Visiter(v ) ;
            fin
      fin
fin
```

3.8.2 Exploration en largeur

Un parcours en largeur explore le graphe à partir d'un sommet donné. L'algorithme permet de calculer les distances de tous les sommets accessibles depuis le sommet source. Le principe est de simuler la transmission d'un message à partir d'un sommet source, tel que : Tout sommet qui reçoit le message, le transférera à tous ses voisins qui ne l'auront pas encore reçu. Pour modéliser le fait si un sommet a déjà reçu le message ou pas, on utilise le marquage. Pour stocker l'ensemble de sommets ayant reçu le message, mais qui ne l'ont pas encore propagé, on utilise une file (FIFO).

```
Algorithme : Parcours en largeur BFS(G, s)
Données : graphe G, sommet de départ s
File q (initialisée à vide), marque des sommets (initialisé à Faux)
début
marque[s] ← Vrai ;
enfiler s à la fin de q ;
tant que q non vide faire
            Début
                u ← tête(q) ;
                pour chaque v voisin de u faire
                            début
                                si v non marqué alors
                                            début
                                                marque[v ] ← Vrai ;
                                                enfiler v à la fin de q ;
                                            fin
                            fin
                défiler u de la tête de q ;
            fin
fin
```

3.9 Connexité, forte connexité, composantes connexes, composantes fortement connexes, graphe réduit

Un graphe orienté G(V, A) est dit ***fortement connexe*** si à partir de n'importe quel sommet, nous pouvons atteindre n'importe quel autre sommet. Si G ne l'est pas, nous pouvons être menés à chercher des sous-graphes de G ayant cette caractéristique. Ils sont appelés dans ce cas Composantes Fortement Connexes (CFC) de G. et nous utilisons l'algorithme décrit ci-dessous pour les trouver.

L'algorithme de marquage

L'algorithme dit 'de marquage' permet de trouver les composantes fortement connexes d'un graphe orienté G. Son principe est le suivant :

```
k = 0
Tant qu'il reste des sommets dans le graphe faire
    | choisir un sommet x et le marquer par (+) et (-), k = k + 1
    | marquer tous les successeurs directs et indirects de x par (+)
    | marquer tous les prédécesseurs directs et indirects par (-)
    | les sommets marqués avec (+) et (-) forment la composante connexe Ck
    | retirer tous les sommets de Ck, effacer toutes les marques
```

Le graphe réduit d'un graphe G, est un graphe où les sommets sont les composantes connexes de G. Les arcs du graphe réduit sont tracés de la façon suivante :

si $((v_i,v_j) \in A)$ **et** $(v_i \in C_k)$ **et** $(v_j \in C_{k'})$, alors un arc est tracé de C_k à $C_{k'}$.

Application :

Soit le graphe G de la figure 3.11.

Le graphe G est-il fortement connexe ? justifiez.

Appliquer l'algorithme de marquage sur G pour ressortir ses composantes fortement connexes, et tracez son graphe réduit.

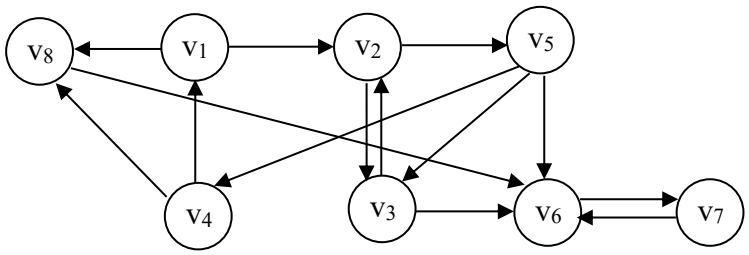


Figure *3.11* : graphe G

Réponse :

Non, le graphe n'est pas fortement connexe parce qu'il n'existe pas un chemin entre les sommets v6 et v8 par exemple.

Pour l'application de l'algorithme de marquage, nous allons utiliser un tableau où les colonnes représentent les sommets, et nous ajoutons une dernière colonne pour les composantes

fortement connexes que nous allons constituer. Chaque itération de l'algorithme correspond à une ligne dans le tableau.

On choisit de marquer le sommet v_1 dans l'itération 1, on le marque par (+,-), puis on marque tous ses successeurs directs et indirects par (+) et tous ses prédécesseurs directs et indirects par (-), nous obtenons la ligne suivante :

Table 3.1 : première itération de l'algorithme de marquage

itération	**k**	**v_1**	**v_2**	**v_3**	**v_4**	**v_5**	**v_6**	**v_7**	**v_8**	**Comp.fort.connexe**
1	**1**	+ -	+-	+-	+ -	+-	+	+	+	$C_1 = \{v_1, v_2, v_3, v_4, v_5\}$

Les sommets v_1, v_2, v_3,v_4,et v_5 sont tous marqués par (+,-) dans cette itération. Donc, ils forment ensemble la composante fortement connexe C_1 (k=1). On ajoute C_1 dans le tableau, on supprime ses sommets du traitement et on passe à l'itération 2.
On choisit maintenant de marquer le sommet v_6 par (+,-), puis ses successeurs par (+) et ses prédécesseurs par (-). On arrive ainsi à constituer la C_2 avec les sommets v_6 et v_7., on les supprime du traitement et on passe à l'itération 3.

Table 3.2 : Deuxième itération de l'algorithme de marquage

itération	**k**	**v_1**	**v_2**	**v_3**	**v_4**	**v_5**	**v_6**	**v_7**	**v_8**	**Comp.fort.connexe**
1	**1**	+ -	+-	+-	+ -	+-	+	+	+	$C_1 = \{v_1, v_2, v_3, v_4, v_5\}$
2							+-	+-	-	$C_2 = \{v_6, v_7\}$

On continue de la même façon pour obtenir le tableau suivant :

Table 3.3 : Troisième itération de l'algorithme de marquage

itération	**k**	**v_1**	**v_2**	**v_3**	**v_4**	**v_5**	**v_6**	**v_7**	**v_8**	**Comp.fort.connexe**
1	**1**	+ -	+-	+-	+ -	+-	+	+	+	$C_1 = \{v_1, v_2, v_3, v_4, v_5\}$
2	**2**						+-	+-	-	$C_2 = \{v_6, v_7\}$
3	**3**								+-	$C_3 = \{v_8\}$

Ainsi, le graphe G comporte 3 composantes connexes :

- $C_1 = \{v_1, v_2, v_3, v_4, v_5\}$
- $C_2 = \{v_6, v_7\}$
- $C_3 = \{v_8\}$

Le graphe réduit du graphe G est donné par la figure 3.12.

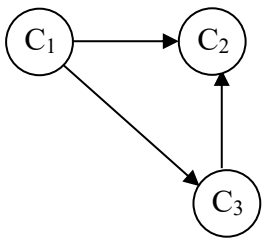


Figure *3.12* : Le graphe réduit de G

3.10 Décomposition en niveaux d'un graphe

L'ordonnancement d'un graphe consiste à ordonner ses sommets en niveaux (ou rang) tel que les arcs vont tous dans le même sens (toujours d'un niveau i à un niveau j > i). L'ordonnancement appelé aussi classement ou décomposition en niveaux, est basé sur les *prédécesseurs* des sommets.

a) Algorithme d'ordonnancement d'un graphe

```
Rang = 1
Tant qu'il existe des sommets non classés faire
        Déterminer les sommets non classés dont l'ensemble des prédécesseurs est vide
        Leur affecter le niveau rang
        Rang = Rang + 1
```

Appliquons l'algorithme sur le graphe de la figure 3.13.

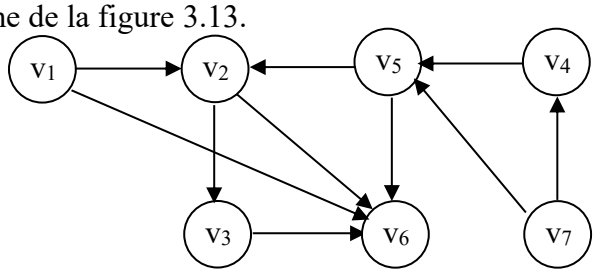


Figure 3.*13* : graphe G à décomposer en niveaux

Nous allons utiliser pour chaque itération un tableau de 3 colonnes, une pour les sommets, une pour les prédécesseurs de chaque sommet et la dernière pour le classement des sommets dans le niveau (le rang). Dans chaque itération, le sommet x dont $\Gamma^-(x)$ est vide est classé dans le niveau en cours puis retiré du traitement de l'itération suivante. Nous obtenons ainsi le tableau suivant :

Table 3.4 : Application de l'algorithme d'ordonnancement

x	$\Gamma^-(x)$	niveau (v_1, v_7) = **1**	x	$\Gamma^-(x)$	niveau (v_4) = **2**	x	$\Gamma^-(x)$	niveau (v_5) = **3**	x	$\Gamma^-(x)$	niveau (v_2) = **4**	x	$\Gamma^-(x)$	niveau (v_3) = **5**	x	$\Gamma^-(x)$	niveau (v_6) = **6**
v_1	Ø		v_1	-		v_1	-		v_1	-		v_1	-		v_1	-	
v_2	v_1,v_5		v_2	v_5		v_2	v_5		v_2	Ø		v_2	-		v_2	-	
v_3	v_2		v_3	v_2		v_3	v_2		v_3	v_2		v_3	Ø		v_3	-	
v_4	v_7		v_4	Ø		v_4	-		v_4	-		v_4	-		v_4	-	
v_5	v_4,v_7		v_5	v_4		v_5	Ø		v_5	-		v_5	-		v_5	-	
v_6	v_1,v_2, v_3,v_5		v_6	v_2,v_3, v_5		v_6	v_2,v_3, v_5		v_6	v_2,v_3		v_6	v_3		v_6	Ø	
v_7	Ø		v_7	-		v_7	-		v_7	-		v_7	-		v_7	-	

Itération 1 — Itération 6

Maintenant nous pouvons retracer le graphe selon les niveaux trouvés de façon à ce que tous les arcs vont dans un seul sens et donc toujours d'un niveau inférieur à un autre supérieur. La figure 3.14 illustre le classement des sommets du graphe G.

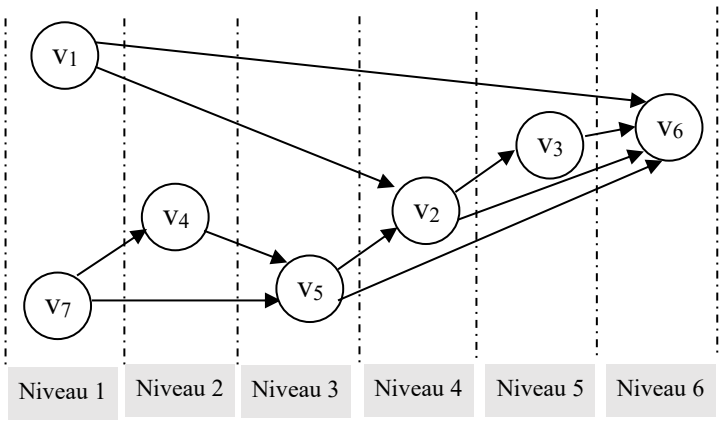


Figure 3.*14* : graphe G décomposé en niveaux

3.11 Conclusion

Dans ce chapitre, nous avons présenté les différentes notions de cheminement dans les graphes, qui constituent un élément central dans leur analyse et exploitation. Les concepts fondamentaux de chaînes, cycles, cocycles, chemins et circuits dans les graphes non orientés ont été introduits. La notion de la connexité a permis de comprendre comment identifier les différentes composantes d'un graphe et d'évaluer son niveau de cohésion. Également, plusieurs familles importantes de graphes particuliers, notamment les graphes planaires, eulériens et hamiltoniens ont été balayées en mettant en évidence leurs propriétés ainsi que leurs applications dans des problèmes réels de parcours et d'optimisation.

Dans le cas des graphes orientés, nous avons examiné les notions de chemins, de circuits, de sources et de puits, ainsi que plusieurs propriétés structurelles telles que la réflexivité, la symétrie et la transitivité. La fermeture transitive a ensuite été introduite comme un outil permettant d'étudier l'accessibilité entre les sommets. Les concepts du parcours en profondeur et du parcours en largeur, qui constituent la base de nombreuses méthodes de traitement et d'analyse des réseaux, ont été aussi abordé dans ce chapitre. Enfin, la décomposition en niveaux a montré comment organiser et simplifier la structure des graphes orientés afin de faciliter leur interprétation et leur exploitation.

Série de TD n°2

Exercice 1

Soit un graphe planaire connexe avec 12 sommets et 18 arêtes. Combien a-t-il de faces ?

Exercice 2

Prouvez qu'un graphe planaire G(V,E) de n sommets et m arêtes a au plus **3n – 6 arêtes**, et que s'il est **biparti** il a au plus **2n -4 arêtes**.

Exercice 3

Est-ce que le graphe complet K_5 et le graphe biparti complet $K_{3,3}$ sont planaires ? expliquez

Exercice 4

Trois maisons (M1, M2, M3) et trois usines (Eau, Gaz et Elec) sont considérées. La première usine fournit de l'eau, la deuxième du gaz et la troisième de l'électricité. On désire relier chacune des trois maisons aux trois usines pour qu'elles aient accès à l'eau, au gaz et à l'électricité. Une contrainte supplémentaire est que les conduites ne doivent jamais se croiser. Est-il possible de le faire. Dites pourquoi.

Exercice 5

Dessinez un graphe d'ordre au moins 5 qui soit :

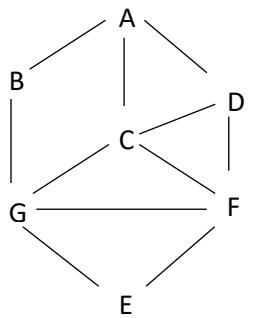
- Hamiltonien et eulérien
- Hamiltonien et non eulérien
- Non hamiltonien et eulérien
- Non hamiltonien et non eulérien

Exercice 6

Des touristes sont logés dans un hôtel nommé A. Un guide fait visiter six sites touristiques nommés B, C, D, E, F et G. Les tronçons de route qu'il peut emprunter sont représentés sur le graphe ci-dessous.

1) A partir de l'hôtel, le guide peut-il emprunter tous les tronçons de route en passant une et une seule fois sur chacun d'eux ?

2) Même question s'il doit obligatoirement terminer sa tournée à l'hôtel.



Exercice 7

Un club de 9 joueurs se réunit chaque jour autour d'une table ronde. Une règle du club interdit qu'un joueur ait deux fois la même personne à côté de lui.

1. Combien de jours au maximum pourront ils se réunir en satisfaisant cette règle ?
2. Donnez une organisation de la table pour chacun de ces jours
3. Même question que 1) mais avec 3 tables de places
4. Donnez une organisation des 3 tables pour chacun de ces jours

Exercice 8

Soit l'ensemble des pairs suivant : {(0,3), (2,7), (3,8), (4,5), (5,1), (6,0), (7,3), (8,7)}. On dit que (a,b) domine (c,d) si ($a \leq c$ et $b < d$) ou ($a < c$ et $b \leq d$). Par exemple : (0,3) domine (7,3)

1. Construisez le graphe correspondant à la relation de domination.
2. Le graphe est-il transitif ? justifiez.
3. Appliquez la décomposition en niveaux à ce graphe
4. Une paire est dite pareto si elle ne peut pas être dominée. Dites comment trouver les paires pareto.

Exercice 9

Un virus se propage dans une population. Une personne atteinte peut transmettre le virus. La table suivante montre les liaisons de transmission.

Personne	a contaminé
1	2,3
2	4
3	5,7
4	6
5	4
6	8
7	-
8	-

1. Représentez la table des contaminations par un graphe G
2. Est-ce que le graphe est fortement connexe ? justifiez
3. Donnez la fermeture transitive de G, appelons le G*
4. Quel est le plus long chemin de G* ? que représente la longueur de ce chemin par rapport au graphe G ?

Exercice 10

Afin de détecter la présence de spammeurs (une personne envoyant des messages électroniques à un grand nombre de personnes sans aucune sollicitation), une étude a permis d'identifier les flux importants de mails. Elle est représentée dans le tableau suivant :

Personne	**Envoie bcp de mails à :**
A	B E D
B	E H
C	D E
D	B A G
E	H G
F	D H G
G	H
H	A B

1. Modélisez cette situation à l'aide d'un graphe
2. Le graphe est-il fortement connexe ? Justifiez
3. Trouvez les composantes fortement connexes du graphe
4. D'après le graphe qui pourrait être un spammeur ? justifiez

Exercice 11

On considère le graphe non orienté suivant :

- Sommets : V={A,B,C,D,E,F,G,H}
- Arêtes: E={AB,AC,BD,BE,CF,EG,FH,GH}

On suppose que, lors des parcours, les voisins sont visités dans l'ordre alphabétique.

1. Représenter le graphe.
2. Effectuer un parcours en largeur (BFS) à partir du sommet **A**.
    - Donner l'ordre de visite.
    - Donner l'arbre BFS obtenu.
3. Effectuer un parcours en profondeur (DFS) à partir du sommet A.
    - Donner l'ordre de visite.
    - Donner l'arbre DFS obtenu.
4. Comparer les deux arbres obtenus et commenter.

Chapitre 4
Arbres et arborescences

4.1 Introduction

Dans ce chapitre, nous allons considérer uniquement les graphes simples, donc les graphes sans boucle et qui comportent au maximum une seule arête/arc entre 2 sommets. Des graphes que nous appellerons ici G = (V,E), et ayant *n sommets et m arêtes/arcs*. Deux notions importantes ont besoin d'être rappelées :

- *La notion de connexité* : G est connexe si pour toute paire de sommets, il existe une chaîne/chemin entre eux.
- *La notion de cycle* : G est cyclique s'il contient une chaîne fermée $x_0,x_1,x_2,\ldots x_0$. (il est dit acyclique dans le cas contraire).

Nous rappelons également que :

- G est connexe $\Rightarrow m \geq n-1$
- G est acyclique $\Rightarrow m \leq n-1$

4.2 Définitions et concepts de base

a) ***Arbre*** : est un graphe
   - *non-orienté*
   - *connexe et sans cycle*.
   - Si l'arbre est d'ordre n, alors m = n-1

Exemples : Les graphes de la figure 4.1 sont des arbres

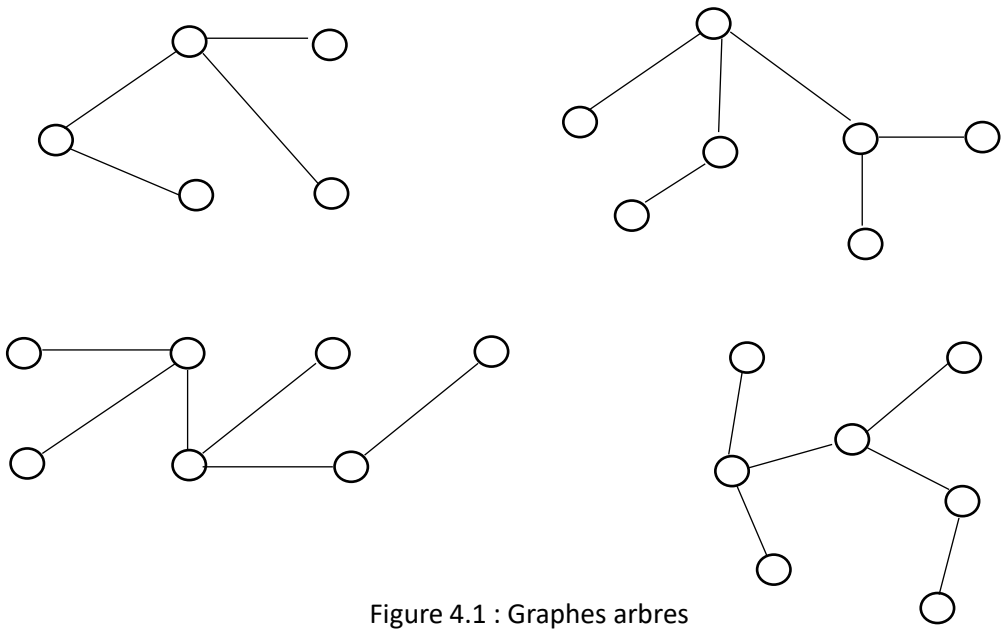


Figure 4.1 : Graphes arbres

La figure 4.2 présente 3 arbres particuliers :

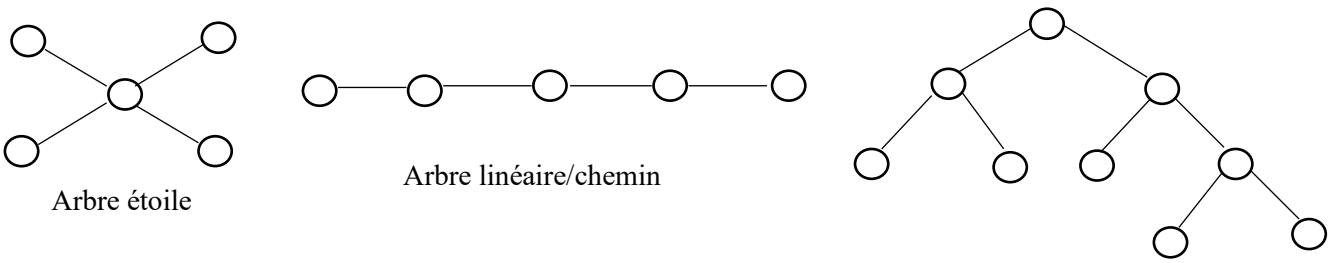


Figure 4.2 : Graphes arbres

Théorème très classique dans le monde des arbres (vous pouvez chercher la démonstration sur le net) :

Pour un arbre G non orienté d'ordre n (n > 2), les assertions suivantes sont équivalentes :

1. G est connexe et sans cycle (donc un arbre);
2. G est connexe et minimal : si on supprime une arête quelconque, il ne sera plus connexe ;
3. G est connexe et m = n − 1
4. G est sans cycle et maximal : si on lui rajoute encore une arête alors il contiendra un cycle ;
5. G est sans cycle et m = n − 1
6. Il existe une chaîne unique entre deux sommets quelconques de G.

Donc, il suffit de démontrer l'une de ces assertions pour démontrer que le graphe est un arbre !

Un arbre peut être colorié avec deux couleurs, il est donc biparti. Tout sommet pendant d'un arbre s'appelle feuille. Dans un arbre d'ordre $n \geq 2$, il existe au moins deux sommets pendants (donc deux feuilles). Dans le graphe de la figure 5.3 : C, D et E: sont des feuilles

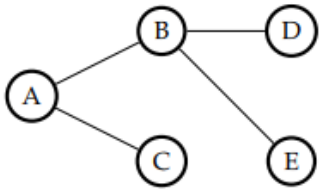


Figure 4.3 : arbre, feuilles

b) ***Forêt*** : Un graphe non connexe et dont chaque composante connexe est un arbre est appelé ***forêt*** Une forêt est un ensemble disjoint d'arbres. Le graphe de la figure 4.4 est une forêt d'arbres.

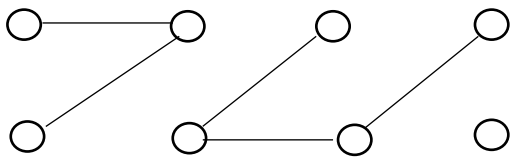


Figure 4.4 : Graphe forêt

c) ***Polyarbre :*** un polyarbre est un arbre orienté, arbre dirigé. Sa définition peut être plus simplifiée : si on remplace tous les arcs par des arêtes, nous obtenons un arbre (connexe et sans cycle). Le graphe de la figure 4.5 est un polyarbre.

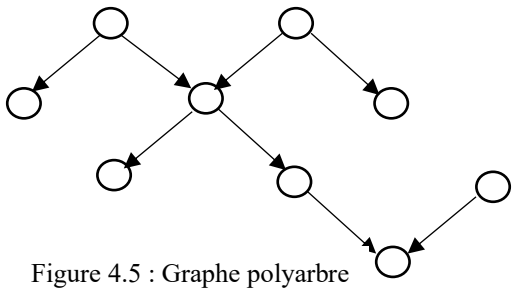


Figure 4.5 : Graphe polyarbre

d) ***Racine, anti-racine*** : dans un graphe orienté, on appelle une ***racine***, un sommet permettant de rejoindre n'importe quel autre sommet. Une ***anti-racine*** est le concept opposé de la racine. Dans le graphe de la figure 4.6, le sommet A est une racine. Si on ajoute un arc du sommet D au sommet F et un arc du sommet F au sommet C dans le même graphe, C devient une anti-racine comme le montre la figure 4.7.

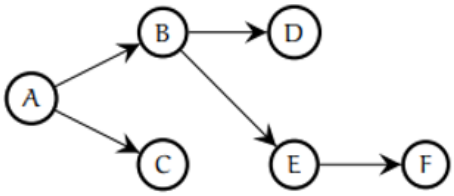


Figure 4.6 : racine

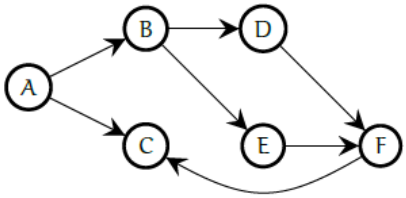


Figure 4.7 : anti-racine

e) ***.Arborescence :*** Une *arborescence* est un graphe orienté G sans circuit tel que :

1. G possède un sommet racine S ;
2. Pour tout sommet x différent de S, il existe un chemin unique de S vers x, donc $d^-(x) = 1$ (il a un seul prédécesseur)

Nous pouvons donner aussi une autre définition pour une arborescence : Une arborescence est un graphe orienté acyclique, comportant un sommet particulier qui permet d'atteindre tous les autres sommets et dont le graphe non orienté sous-jacent est un arbre. Notons que « toute arborescence est un polyarbre mais l'inverse est faux », et également : « si l'arborescence est d'ordre n, alors elle contient **m = n-1 »**. Le graphe de la figure 4.8 présente un exemple d'arborescence dont la racine est « a ».

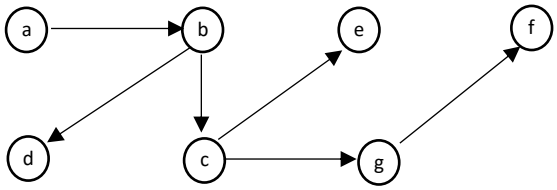


Figure 4.8 : Arborescence

f) Anti-Arborescence : Une anti-arborescence est un arbre orienté où tout point possède au plus un successeur. Si on inverse le sens des arcs d'une arborescence, on obtient une anti-arborescence. Un seul point ne possède pas de successeurs, il est appelé anti-racine

4.3 Les applications des arbres/arborescences

Parmi les applications des arborescences : Découpage hiérarchique comme :

- le découpage d'un livre en chapitres, sections, paragraphes, etc.
- Liens généalogiques, par exemple dans une famille, où le modèle a été repris en informatique sous la forme de l'héritage en conception orientée objet.
    - les sommets représentent les membres
    - et les arcs les liens de parenté.

Beaucoup d'algorithmes utilisent les arbres pour résoudre des problèmes de manière plus efficace

4.4 Arbre de couverture

4.4.1 Définitions

Soit G = (X, E) un graphe non-orienté connexe. Un ***arbre de couverture (ou arbre couvrant)*** est un graphe *partiel* de G qui est :

- un ***arbre*** donc, connexe et acyclique
- ***Partiel*** : touche tous les sommets de G, avec un sous-ensemble de ses arêtes.

Remarque importante : Dans un graphe, on peut construire **plusieurs arbres couvrants**, leur nombre (***AC***) dépend de la forme du graphe. Pour les graphes complets et biparti complet, nous donnons les formules suivantes pour calculer AC.

- Graphe complet K_n : $AC = n^{n-2}$
- Biparti complet $K_{m,n}$: $AC = m^{n-1} * n^{m-1}$

Théorème : G=(V,E) est connexe est équivalent à dire G contient un arbre couvrant

4.4.2 Arbre de couverture minimale

Dans cette section, nous allons considérer les graphes pondérés, où à chaque arête est associée un poids.

a) **Définition** :
L'arbre couvrant de poids minimal ou l'arbre couvrant minimal (ACM) est l'arbre couvrant dont la somme des poids des arêtes est minimale. Si le graphe *n'est pas connexe*, alors on cherche un ACM pour chaque composante connexe. Et donc une *forêt* de poids minimal.

b) **Les applications des ACM**

La construction des ACM a plusieurs applications.

- ***Exemple 1*** : Dans un système de communication par message, si un nœud souhaite diffuser un message, il peut construire un arbre de couverture de poids minimal pour faire parvenir le message à tous les autres nœuds avec un coût minimal.
- ***Exemple 2*** : Dans un réseau, par exemple un réseau d'ordinateurs, on utilise des arbres pour effectuer des connexions si on veut minimiser le nombre de lignes de connexion.

- ***Exemple 3*** : si on veut minimiser le coût d'installation téléphonique dans un quartier, le problème peut être modélisé sous forme de recherche d'un arbre de poids minimal. Idem pour un réseau électrique…

Il existe plusieurs algorithmes pour trouver l'arbre de couverture de poids minimal : les plus populaires sont Kruskal et Prim.

4.4.3 L'algorithme de Kruskal

L'algorithme de Kruskal permet de trouver l'ACM d'un graphe pondéré.

Soient :

- ***n*** = *ordre (G);*
- ***m*** = *taille(G); et*
- **T** = l'ensemble des arêtes qui appartiennent à l'arbre cherché

Trier les arêtes a de G dans l'ordre croissant des poids
T = ϕ ; k=1 ;

Tant que k ≤ m et card(T) < n-1 faire

Si l'ajout de a_k ne crée pas de cycle alors T = T ∪ { a_k } ;

k=k+1;

Appliquons cet algorithme sur le graphe G = (V, E) pondéré suivant pour trouver son ACM

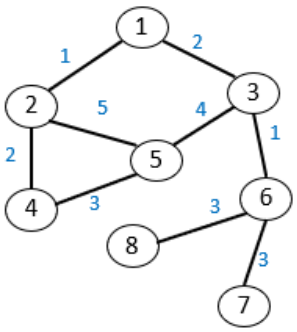


Figure 4.9 : graphe non orienté pondéré

En 1er lieu, les sommets doivent être triés dans l'ordre croissant de leurs degrés, ce qui donne le tableau suivant :

Table 5.1 : Ordre croissant des degrés des sommets

Arête	**(1,2)**	**(3,6)**	**(1,3)**	**(2,4)**	**(4,5)**	**(6,7)**	**(6,8)**	**(5,3)**	**(2,5)**
Poids	1	1	2	2	3	3	3	4	5
Indice k	1	2	3	4	5	6	7	8	9

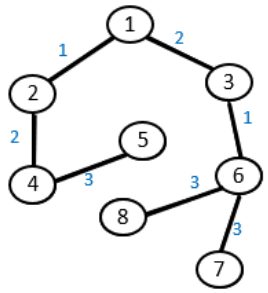
Au démarrage T = ϕ

Condition d'arrêt : k ≤ 9 et nombre d'arêtes de l'arbre = 8-1 = 7

Table 5.2 : Application de l'algorithme d'ordonnancement

k	L'ensemble F des arêtes de l'arbre	Action/remarque sur l'arête
1	T = {(1,2)}	L'arête (1,2) est ajoutée à l'arbre
2	T = {(1,2),(3,6)}	L'arête (3,6) est ajoutée à l'arbre
3	T = {(1,2),(3,6),(1,3)}	L'arête (1,3) est ajoutée à l'arbre
4	T = {(1,2),(3,6),(1,3),(2,4)}	L'arête (2,4) est ajoutée à l'arbre
5	T = {(1,2),(3,6),(1,3),(2,4),(4,5)}	L'arête (4,5) est ajoutée à l'arbre
6	T = {(1,2),(3,6),(1,3),(2,4),(4,5),(6,7)}	L'arête (6,7) est ajoutée à l'arbre
7	T = {(1,2),(3,6),(1,3),(2,4),(4,5),(6,7),(6,8)}	L'arête (6,8) est ajoutée à l'arbre
	A ce niveau, nous avons inséré 7 arêtes dans T et l'algorithme doit s'arrêter, car nous savons que Card (T) = n- 1 = 7 dans le cas de ce graphe. Et de toute façon, si on tente de continuer les itérations, nous obtenons :	
8	T = {(1,2),(3,6),(1,3),(2,4),(4,5),(6,7),(6,8)}	L'arête (5,3) est ignorée car elle forme un cycle
9	T = {(1,2),(3,6),(1,3),(2,4),(4,5),(6,7),(6,8)}	L'arête (2,5) est ignorée car elle forme un cycle

Donc le résultat de l'algorithme est l'ACM suivant de poids total égal à 15.



Remarque : L'algorithme de Kruskal s'applique également sur un graphe non-connexe, et produit dans ce cas une forêt d'ACM. Le nombre des ACMs trouvés est le même que les composantes connexes du graphe original.

4.4.4 L'algorithme de Prim

L'algorithme de Prim (proposé par le mathématicien américain Robert Clay Prim en 1957) est un algorithme glouton qui permet de construire un arbre couvrant minimal (ACM) d'un graphe pondéré connexe. Contrairement à l'algorithme de Kruskal qui sélectionne les arêtes dans l'ordre croissant de leurs poids, l'algorithme de Prim construit progressivement un arbre unique en partant d'un sommet quelconque et en ajoutant à chaque étape l'arête de poids minimal reliant l'arbre en construction à un nouveau sommet.

L'algorithme de Prim garantit l'obtention d'un arbre couvrant de poids minimal, son principe est simple :

1. Choisir un sommet de départ.
2. Considérer cet unique sommet comme l'arbre courant.
3. Rechercher l'arête de plus faible poids reliant un sommet de l'arbre à un sommet extérieur.
4. Ajouter cette arête et le nouveau sommet.
5. Répéter jusqu'à ce que tous les sommets soient inclus.

Il est bien évident que pour appliquer Prim,

- le graphe doit être non orienté ;
- le graphe doit être connexe ;
- les arêtes doivent être pondérées.

Si le graphe n'est pas connexe, l'algorithme construit un ACM pour chaque composante connexe (forêt minimale).

Soit :

- G=(V,E) un graphe pondéré ;
- T : ensemble des arêtes de l'arbre minimal.

Choisir un sommet de départ S
T = ∅
A = {S}

Tant que |A| < n faire

Choisir l'arête de poids minimum reliant un sommet de A à un sommet extérieur à A

Ajouter cette arête à T

Ajouter le nouveau sommet à A

Fin Tant que

Appliquons l'algorithme de Prim sur le même graphe 4.9.

En choisissant un sommet quelconque, disons le sommet 5. Initialement :

T = ∅
A = {5}

Itération	A	T	Poids de l'arbre
1	5, 4	(5,4)	3
2	5,4,2	(5,4), (4,2)	5
3	5,4,2,1	(5,4), (4,2) (2,1)	6
4	5,4,2,1,3	(5,4), (4,2) (2,1), (1,3)	8
5	5,4,2,1,3, 6	(5,4), (4,2) (2,1), (1,3), (3,6)	9
6	5,4,2,1,3, 6,8	(5,4), (4,2) (2,1), (1,3), (3,6), (6,8)	12
7	5,4,2,1,3, 6,8,7	(5,4), (4,2) (2,1), (1,3), (3,6), (6,8), (6,7)	15

Et le poids = 15, et il s'agit bien du même ACM trouvé par l'algorithme de Kruskal.

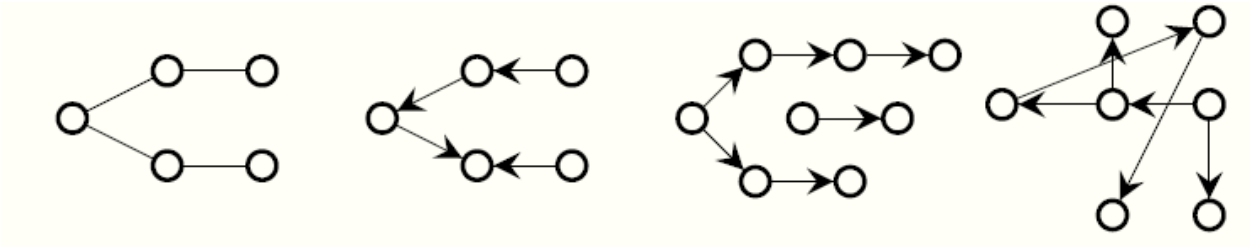
4.5 Conclusion

Dans ce chapitre, les notions fondamentales liées aux arbres et aux arborescences qui constituent des structures essentielles en théorie des graphes, ont été étudiées. Nous avons abordé la notion d'arbre couvrant en portant une attention particulière aux arbres couvrants minimaux, dont l'objectif est de minimiser le coût total des connexions dans un graphe pondéré. Deux algorithmes classiques de recherche d'arbres couvrants minimaux ont été présentés avec des exemples d'application, à savoir : l'algorithme de Kruskal et l'algorithme de Prim.

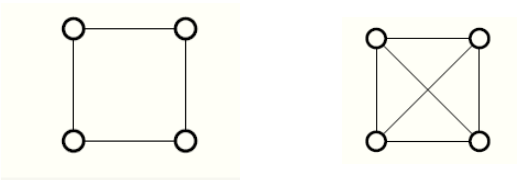
Série de TD n°3

Exercice 1

a) Les graphes représentés ci-dessous, sont-ils des arbres, des forêts, des arborescences, ou des anti-arborescences ?

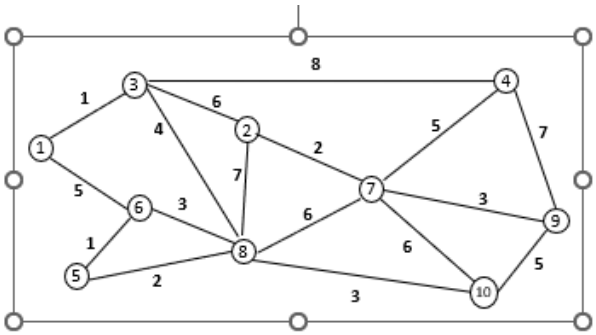


b) Pour les graphes suivants, donnez le nombre d'arbres de couverture possibles



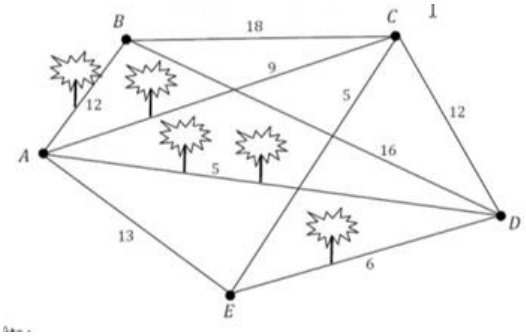
Exercice 2

Trouver l'arbre couvrant de poids minimum du graphe G suivant :



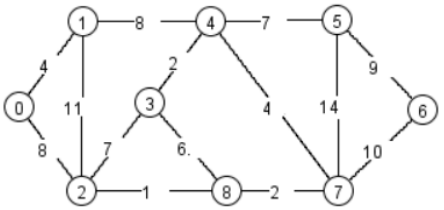
Exercice 3

On veut réaliser un réseau ferroviaire entre 5 villes (graphe ci-dessous) avec un cout minimum, sachant que le coût de revient unitaire au km est de 6 unités et celui de découpage d'un arbre est de 30 unités. Déterminer un réseau de coût minimum, donner son tracé et son coût.



Exercice 4

Trouver l'ACM du graphe suivant en appliquant l'algorithme de Prim.



Chapitre 5
Problème des plus courts chemins

5.1 Introduction et contexte

L'un des problèmes les plus fondamentaux en théorie des graphes et en optimisation des réseaux est celui du chemin le plus court. Ce problème se pose naturellement dans de nombreuses situations réelles où il est nécessaire de déterminer le moyen le plus efficace pour se déplacer d'un point à un autre au sein d'un réseau.

Dans sa forme la plus générale, un réseau peut être modélisé par un graphe pondéré, où les sommets représentent des entités telles que des villes, des routeurs ou des unités de traitement, et les arêtes représentent les connexions entre elles. Chaque arête se voit attribuer une valeur numérique, appelée poids, qui peut représenter une distance, un coût, un temps de trajet ou toute autre mesure quantitative associée à la traversée de cette connexion.

Le problème du chemin le plus court consiste à déterminer un chemin entre deux sommets de manière à minimiser la somme des poids des arêtes le long du chemin. En d'autres termes, l'objectif est de trouver le chemin dont le coût total est le plus faible en fonction des poids attribués aux arêtes. Ce problème joue un rôle central dans de nombreuses applications. Dans les réseaux de transport, il est utilisé pour déterminer l'itinéraire le plus court ou le plus rapide entre deux endroits. Dans les réseaux informatiques, les algorithmes de chemin le plus court sont utilisés dans les protocoles de routage pour déterminer le chemin le plus efficace pour la transmission de données. De même, dans la logistique et la recherche opérationnelle, les modèles de chemin le plus court sont utilisés pour optimiser le mouvement des marchandises et des ressources à travers des réseaux complexes.

D'un point de vue théorique, le problème du chemin le plus court a conduit au développement de plusieurs algorithmes importants qui calculent efficacement les chemins optimaux dans les graphes pondérés. Parmi les méthodes les plus connues, on trouve l'algorithme de Dijkstra, particulièrement efficace pour les graphes dont les arêtes ont des poids non négatifs, et l'algorithme de Bellman-Ford, qui peut traiter les graphes contenant des arêtes de poids négatifs et qui est également capable de détecter les cycles négatifs.

5.2 Définitions

a) ***Un réseau*** : est un graphe orienté et valué G = (X, E, d) où X est l'ensemble de sommets, E est l'ensemble des arcs, et d'un ensemble de valeurs réelles qui sont attribuées à chaque arc. Ces valeurs peuvent quantifier un coût de transport, une distance, une durée, etc. Il peut également s'agir d'un bénéfice/intérêt/récompense dans le cas d'une valeur négative.

b) ***Coût d'un chemin*** : Le coût d'un parcours dans un réseau est égal à la somme des coûts des arcs qui le composent. Si des arcs sont parcourus plusieurs fois dans ce chemin, alors leurs coûts est multiplié le nombre de fois qu'on les parcourt.

c) ***Circuit absorbant*** : Lorsque la somme des coûts des arcs qui composent un circuit est négative, il est appelé « circuit absorbant ». Les circuits absorbants posent problème dans le contexte des plus courts chemins car on peut diminuer le coût d'un chemin indéfiniment en tournant dans ce circuit. La figure 5.1 présente un circuit absorbant. Remarquons par exemple qu'à chaque parcourt, le coût total est réduit de 2. Ainsi, le parcourt n°1 coûte : -1, le parcourt n°2 : -6.

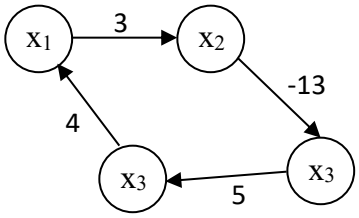


Figure 5.*1* : Circuit absorbant

5.3 Algorithmes de Dijkstra

L'algorithme de Dijkstra s'applique uniquement lorsque les poids sont tous positifs, ce qui exclut la présence des circuits absorbants. Il gère deux types de sommets :

- les sommets visités, dont la distance la plus courte depuis la source est déjà connue ;

- les sommets non visités, dont la distance la plus courte est encore en cours d'estimation.

Pour chaque sommet v l'algorithme conserve une valeur : d(v) qui représente l'estimation actuelle de la distance la plus courte entre la source et le sommet v.

Au départ, la distance à la source est nulle, et la distance à tous les autres sommets est considérée comme infinie. À chaque étape, l'algorithme sélectionne le sommet non visité ayant la plus petite distance temporaire et met à jour les distances de ses sommets voisins.

Soit s le sommet source.

Initialisation
Définissez : d(s)=0 et Pour chaque sommet v≠s ; d(v) = ∝
Marquez tous les sommets comme non visités.

Itération
Répétez les étapes suivantes jusqu'à ce que tous les sommets aient été visités :

1- Sélectionnez le sommet non visité u ayant la plus petite valeur d(u).
2- Marquez le sommet u comme visité.
3- Pour chaque voisin v de u, mettez à jour la distance à l'aide de la règle de relaxation :
$$d(v) = \min(d(v), d(u) + w(u,v))$$
4- Enregistrez u comme prédécesseur de v si la distance est améliorée.

5.4 Algorithme de Bellman Ford

Contrairement à l'algorithme de Dijkstra, l'algorithme de Bellman Ford fonctionne même si certaines arêtes ont des poids négatifs (avec absence de circuit absorbant), ce qui fait un avantage pour lui en plus qu'il est Simple à implémenter. Cependant il a l'inconvénient d'être plus lent que Dijkstra et peu efficace sur les graphes très grands.

« n » étant le nombre de sommets du graphe en question, L'algorithme de Ford Bellman utilise un tableau « d » de n cases, qui va contenir les distances et un tableau « p » de n cases, qui va contenir les prédécesseurs. L'algorithme utilise aussi une matrice carrée n x n. La case [i,j] contient la valeur de l'arc (i,j). Et son fonctionnement est comme suit :

- L'algorithme contient une boucle
- Dans chaque itération, on s'approche mieux de la solution optimale (les plus courts chemins)
- Donc dans chaque itération les tableaux d et p changent de valeurs
- La matrice M est fixe durant tout le déroulement de l'algorithme évidemment

Algorithme de Bellman Ford

Initialisation des tableaux d et p

d(1) = 0 ; p(1) = 1

Pour i = 2 à n faire

d(i) = + ∞
p(i) = 0

Itérations

Tant qu'il existe un arc xy ∈U tel que d(y) > d(x) + M(x,y) faire

d(y) = d(x) + M(x,y)
p(y) = x

Application de Bellman Ford.

Soit le graphe de la figure 5.2, nous voulons trouver les chemins les plus courts depuis le sommet A vers les autres sommets.

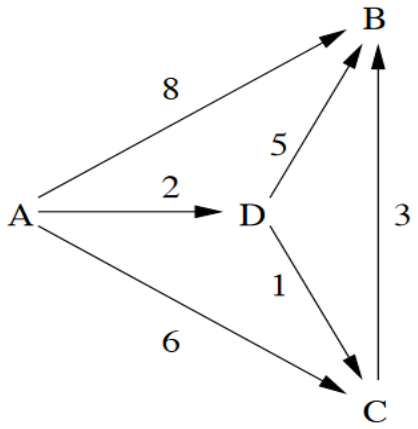


Figure 5.2 Graphe orienté valué

Parcours 1

	A	B	C	D
Initial	**0**	∞	∞	∞
		8	**6**	**2**
		7	**3**	

A	B	C	D
A	**0**	**0**	**0**
	A	**A**	**A**
	D	**D**	

xy	d(y) =	comparer	d(x) + M (x,y) =	⇒	Nouveau d(y)
AB	d(B) = ∞	>	d(A) + M(A,B) = 0 + 8 = 8	⇒	d(B) = **8** ; p(B) = A
AC	d(C) = ∞	>	d(A) + M(A,C) = 0 + 6 = 6	⇒	d(C) = **6** ; p(C) = A
AD	d(D) =∞	>	d(A) + M(A,D) = 0 + 2 = 2	⇒	d(D) = **2** ; p(D) = A
CB	d(B) = 8	<	d(C) + M(C,B) = 6 + 3 = 9	⇒	d(B) ne change pas
DB	d(B) = 8	>	d(D) + M(D,B) = 2 + 5 = 7	⇒	d(B) = **7** ; p(B) = D
DC	d(C) = 6	>	d(D) + M(D,C) = 2 + 1 = 3	⇒	d(C) = **3** ; p(C) = D

Parcours 2

	A	B	C	D
Initial	**0**	**7**	**3**	**2**
		6		

A	B	C	D
A	**D**	**D**	**A**
	C		

xy	d(y) =	comparer	d(x) + M (x,y) =	⇒	Nouveau d(y)
AB	d(B) = 7	<	d(A) + M(A,B) = 0 + 8 = 8	⇒	d(B) ne change pas
AC	d(C) = 3	<	d(A) + M(A,C) = 0 + 6 = 6	⇒	d(C) ne change pas
AD	d(D) = 2	=	d(A) + M(A,D) = 0 + 2 = 2	⇒	d(D) ne change pas
CB	d(B) = 7	>	d(C) + M(C,B) = 3 + 3 = 6	⇒	d(B) = **6** ; p(B) = C
DB	d(B) = 6	<	d(D) + M(D,B) = 2 + 5 = 7	⇒	d(B) ne change pas
DC	d(C) = 3	=	d(D) + M(D,C) = 2 + 1 = 3	⇒	d(C) ne change pas

Parcours 3

	A	B	C	D
Initial	**0**	**6**	**3**	**2**

A	B	C	D
A	**C**	**D**	**A**

xy	d(y) =	comparer	d(x) + M (x,y) =	⇒	Nouveau d(y)
AB	d(B) = 6	<	d(A) + M(A,B) = 0 + 8 = 8	⇒	d(B) ne change pas
AC	d(C) = 3	<	d(A) + M(A,C) = 0 + 6 = 6	⇒	d(C) ne change pas
AD	d(D) = 2	=	d(A) + M(A,D) = 0 + 2 = 2	⇒	d(D) ne change pas
CB	d(B) = 6	=	d(C) + M(C,B) = 3 + 3 = 6	⇒	d(B) ne change pas
DB	d(B) = 6	<	d(D) + M(D,B) = 2 + 5 = 7	⇒	d(B) ne change pas
DC	d(C) = 3	=	d(D) + M(D,C) = 2 + 1 = 3	⇒	d(C) ne change pas

Les chemins les plus courts de A vers les autres sommets sont :

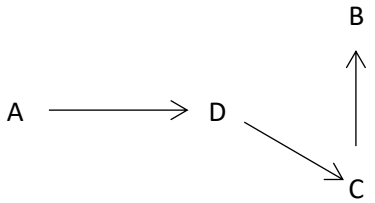


Figure 5.*3* : les plus courts chemins sur le graphe valué

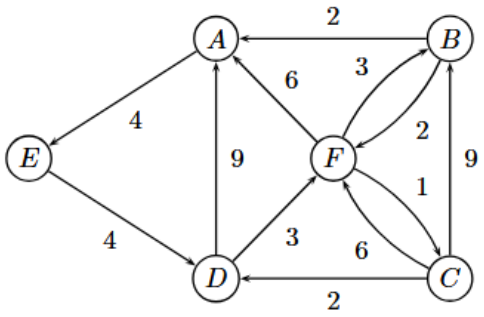
5.5 Conclusion

Nous avons présenté dans ce chapitre servant à résoudre l'un des problèmes majeurs de la théorie de graphe, à savoir : le problème du plus court chemin. L'algorithme de Dijkstra permet de déterminer efficacement les plus courts chemins dans les graphes dont les poids sont non négatifs, tandis que l'algorithme de Bellman-Ford étend cette capacité aux graphes comportant des arcs de poids négatifs, tout en offrant la possibilité de détecter l'existence de circuits absorbants. Ces deux approches illustrent l'importance des méthodes algorithmiques dans la recherche de solutions optimales au sein des réseaux complexes.

Série de TD n°4

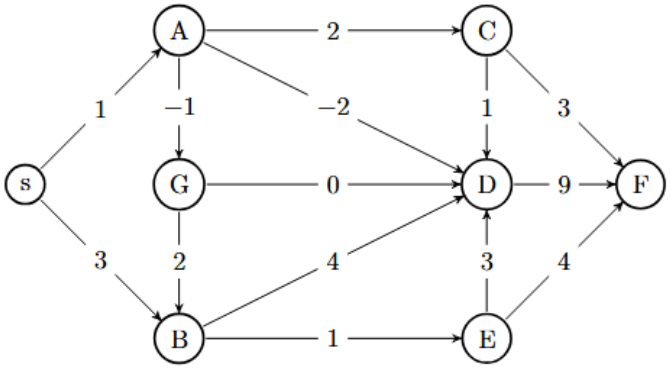
Exercice 1.

Soit le graphe suivant. On veut déterminer les chemins les plus courts depuis le sommet C en utilisant l'algorithme de Dijkstra.



Exercice 2.

Soit le graphe suivant. Trouvez les chemins les plus courts depuis le sommet s par l'application de l'algorithme de Bellman Ford.



Exercice 3.

Soit le tableau suivant donnant information sur les arcs d'un graphe. Dessinez-le, puis calcule avec l'algorithme de Bellman, les plus courts chemins à partir de la source A.

Arc	Poids
A → B	6
A → C	5
A → D	5
B → E	-1
C → B	-2
C → E	1
D → C	-2
D → F	-1
E → F	3

Chapitre 6
Problème des flots

6.1 Introduction

Le problème du flot consiste à faire circuler une certaine quantité d'un "flux" (eau, données, trafic, marchandises, etc.) dans un graphe orienté, en respectant des contraintes de capacité. C'est un problème fondamental en :

- optimisation combinatoire
- recherche opérationnelle
- réseaux de transport
- réseaux informatiques

6.2 Contexte et Définitions

Dans le contexte du flot maximum on dispose d'un **Réseau Orienté** composé de :

- un ensemble de sommets **internes**
- deux autres sommets spécifiques appelés :
  - **Source**
  - **Puits**

Et le tout est relié par des arcs.

La figure 6.1 présente un exemple d'un tel contexte.

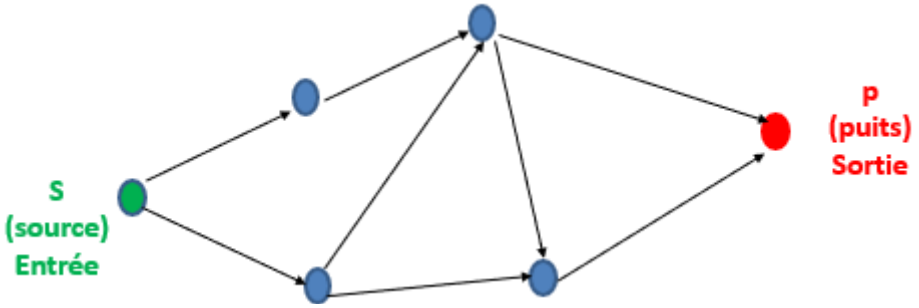


Figure 6.1 : sommets internes, source et puits

Nous aurons besoin dans ce chapitre de quelques hypothèses simplificatrices :

- Il n'y pas d'arcs aller-retour dans le graphe
- En « ***s*** », il n'y a pas d'arcs entrants
- En « ***p*** », il n'y pas d'arcs sortants

Chaque arc du graphe porte une étiquette qui représente ***sa capacité***. Il s'agit d'une quantité de flot maximum que cet arc peut transporter en même temps. Plus la capacité est grande plus on peut faire passer dessus une quantité plus grande sur l'arc.

Dans la suite de ce travail, nous aurons à mettre 2 étiquettes sur chaque arc :

- L'une pour représenter sa capacité limite,
- Et l'autre pour la quantité à transporter sur cet arc
- Pour plus de lisibilité, on utilisera les ***couleurs*** pour les distinguer

La figure 6.2 montre la manière dont ces informations seront représentées graphiquement. D'après cette figure, on va passer 3 unités de flot sur l'arc xy dont la capacité maximale = 5

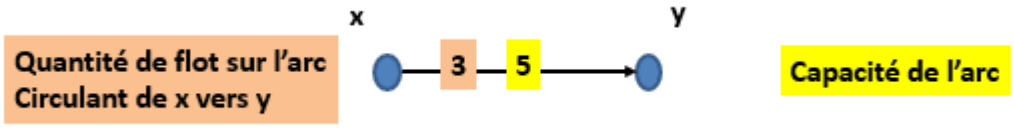


Figure 6.2 : capacité et qte de flot sur l'arc

Dans ce contexte, nous avons des ***contraintes à respecter*** :

1. ***Contrainte de capacité*** : quantité de flot sur un arc doit être toujours inférieure ou égale à la capacité de cet arc**.** Sinon l'arc explose. Autrement dit : l'étiquette marronne devra être toujours inférieur ou égale à l'étiquette jaune. La figure 7.3 illustre cette notion.

2. ***Contrainte de conservation :*** Elle concerne les sommets internes. Tout flot qui rentre en un sommet doit en sortir. Le stockage en un sommet est interdit. La figure 6.4 illustrent des exemples de ce principe.

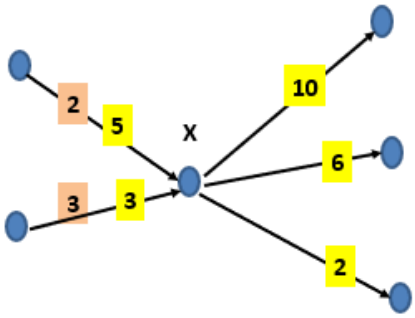


Figure 6.3 : illustration de la contrainte de capacité

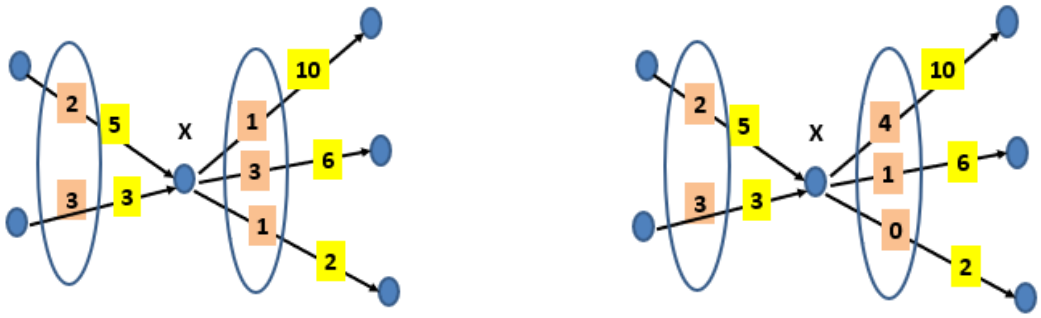


Figure 6.4 : illustration de la contrainte de conservation

Le concept du Flot valide

Un flot est dit valide s'il respecte les contraintes de capacité et de conservation. Vérifions les contraintes de capacité et de conservation sur le graphe de la figure 6.5.

Nous remarquons que ces contraintes sont respectées donc c'est un **flot valide**

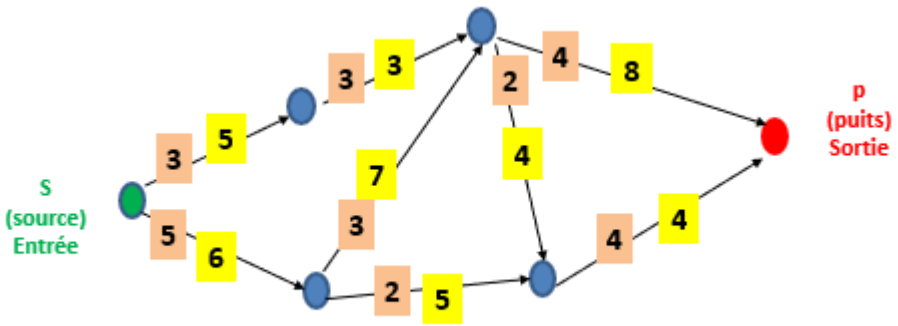


Figure 6.5 flot valide

La valeur du Flot

La valeur du flot c'est le total des quantités qui sortent du sommet « s », ou qui arrivent au sommet « p ». Dans le graphe de la figure 7.6, la valeur du flot $= 3 + 5 = 4 + 4 = 8$

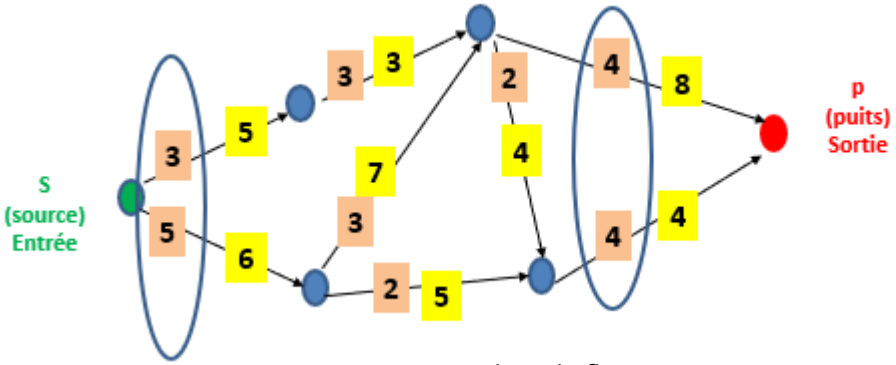


Figure 6.6 : valeur du flot

Mathématiquement, le contexte est modélisé par le **graphe orienté capacitaire** : ***G=(X,E,c)***

- X : sommets
- E : arcs orientés
- $c(u,v) \geq 0$: capacité de l'arc
- Un **flot** est une fonction : $f : E \rightarrow R$
- Un flot valide est tel qu'il satisfait alors :
  - $0 \leq f(u,v) \leq c(u,v)$
  - Pour tout sommet x tel que : $x \neq s$, $x \neq p$, nous avons : $\sum f(u,x) = \sum f(x,v)$
- La valeur du flot : $|f| = \sum f(s,v)$

6.3 Le flot maximum et l'algorithme de Ford Fulkerson

Le problème de flot maximum consiste à chercher la valeur maximale à transporter sur le graphe tout en gardant les contraintes respectées. Donc nous sommes dans un problème d'optimisation de la valeur d'un flot valide. On envoie progressivement du flot de la source s vers le puits p.

Mais après avoir envoyé un certain flot :

- certaines arêtes sont partiellement utilisées
- certaines sont saturées
- parfois on aimerait corriger un mauvais choix précédent, et on utilisera la notion du **graphe résiduel**.

a) *Principe*

- Trouver le flot maximum sur un réseau de transport
- En respectant les contraintes de capacité et de conservation

- Et en fonctionnant à base de chaîne améliorante

b) Chaîne augmentante

Une **chaîne augmentante** est un chemin non-orienté de à s à p, (on peut prendre les arcs dans n'importe quel sens) tel que :

- Sur les arcs pris dans « le bon sens », la valeur du flot est strictement inférieure à la capacité de l'arc ;
- Sur les arcs pris dans « le mauvais sens », la valeur du flot est strictement supérieure à 0

Le **potentiel d'augmentation « p »** est défini pour chaque arc du chemin.

- Pour un arc pris dans le bon sens, p est égal à la différence entre la capacité de l'arc et la valeur du flot;
- Pour un arc pris dans le mauvais sens, p est égal à la valeur du flot (la différence entre la valeur du flot et 0).

La capacité résiduelle,

- Sur un arc direct, elle représente la réponse à « combien on peut encore envoyer sur cet arc ?», sa valeur : $\mathbf{c_f(u,v)=c(u,v) - f(u,v)}$
- Sur un arc inverse, elle représente la réponse à « combien on peut annuler du flot déjà envoyé », sa valeur est : $\mathbf{c_f(v,u) = f(u,v)}$

c) L'algorithme de Ford Fulkerson

L'algorithme de Ford Fulkerson a été proposé par Lester Randolph Ford junior et Delbert R. Fulkerson en 1956 dans leur célèbre article « Maximal Flow Through a Network ». Son principe est le suivant :

Algorithme de Ford Fulkerson

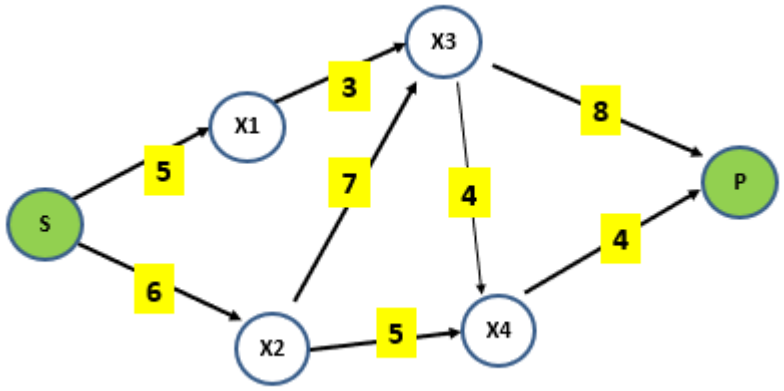
1. Initialiser f = 0
2. Tant qu'il existe un chemin augmentant :
   - Trouver un chemin *P*
   - Calculer :

     $$\Delta = \min c_f(u,v)$$

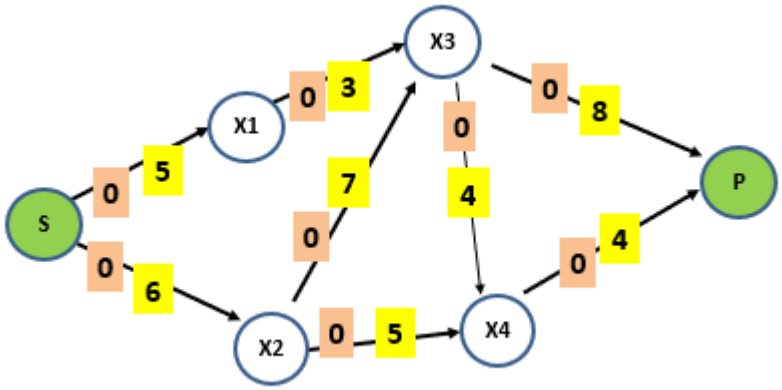
   - Augmenter le flot de Δ sur P
   - Mettre à jour le graphe résiduel
3. Quand aucun chemin augmentant n'existe → flot maximal atteint

Application

Chercher le flot maximum du réseau suivant selon l'algorithme de Ford Fulkerson

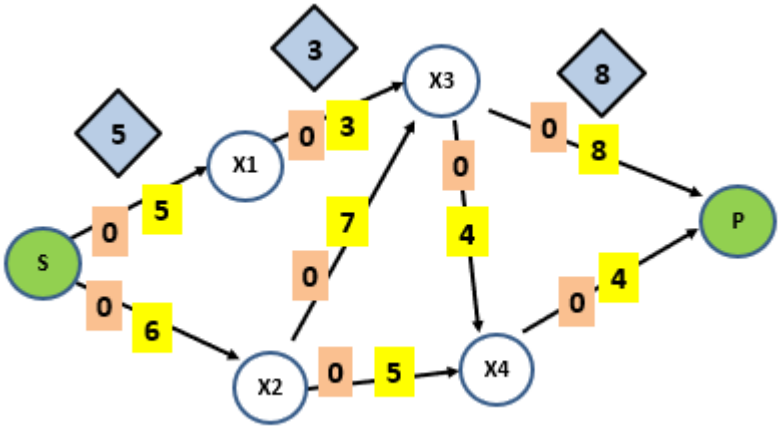


Etape 0 : On commence avec un flot initial nul (valeur = 0)



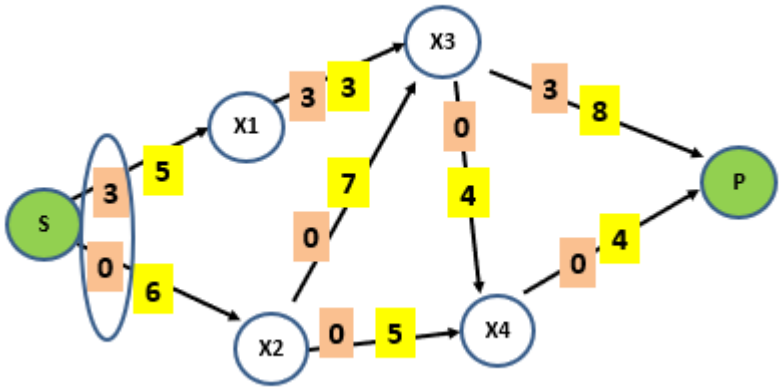
Nous allons améliorer ce flot étape par étape

Etape 1 : on sélectionne un chemin améliorant quelconque : (s x1 x3 p)



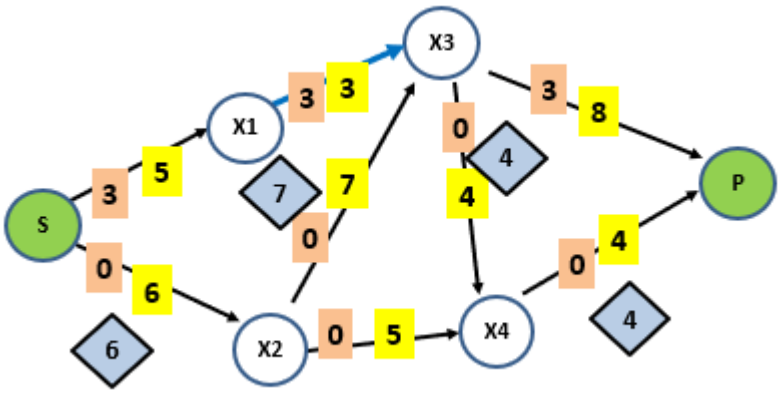
On détermine le plus petit potentiel d'augmentation et on l'injecte dans le chemin : 3

On obtient le Flot suivant :



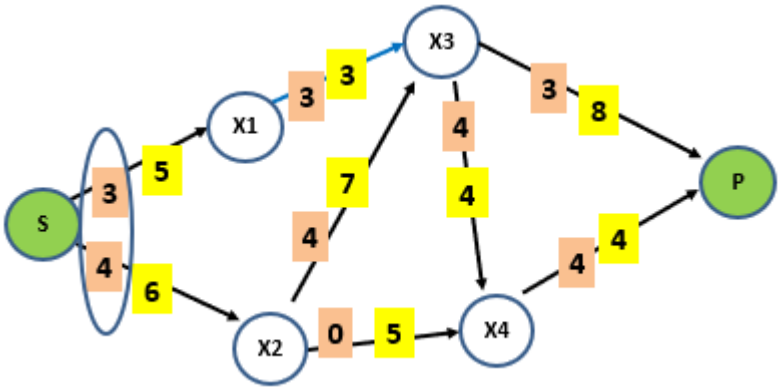
L'arc x_1x_3 est saturé !!! La valeur de Flot obtenue = 3

Etape 2 : on cherche un autre chemin améliorant : (s x2 x3 x4 p)



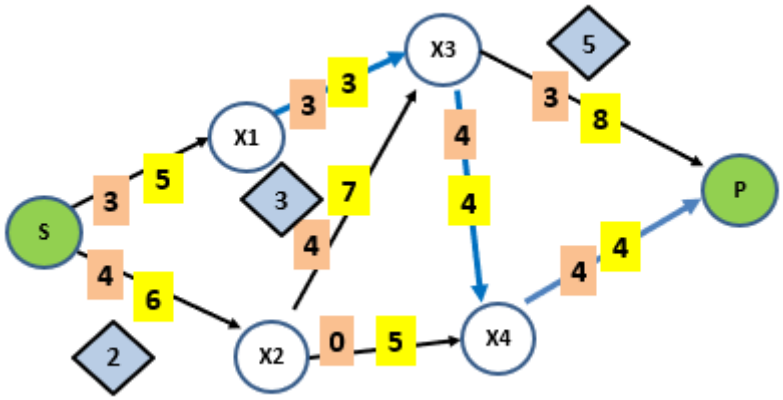
Le plus petit potentiel augmentant = 4, donc on l'ajoute à tous les arcs de ces chemins.

On obtient le Flot suivant :



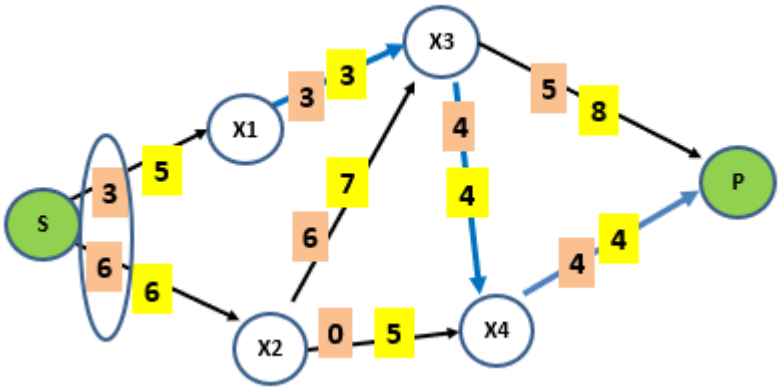
L'arc x_3x_4 et l'arc x_4p sont Saturés !!!! La valeur de Flot obtenue = 7

Etape 3 : On cherche un autre chemin augmentant : (s x2 x3 p)



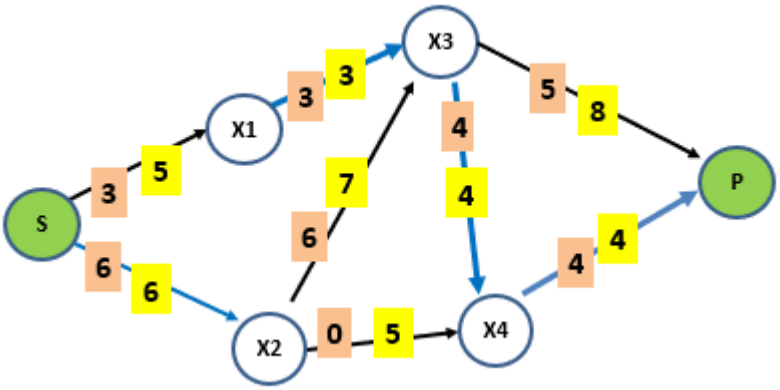
Le plus petit potentiel augmentant = 2, donc on l'ajoute à tous les arcs de cette chaine

On obtient le Flot suivant :



L'arc SX_2 est Saturé !!!! La valeur de Flot obtenue = 9

Etape 4 : Vérifions si on peut faire mieux encore !



Impossible de trouver un chemin augmentant … Donc ce Flot est Maximum

La valeur de Flot Max = 9

6.4 Coupe et coupe minimale

a) Coupe et capacité d'une coupe

Une coupe d'un réseau dont le sommet source est « s », et le sommet puits est « p » est une partition de ses sommets en deux ensembles : S et P tels que :

- $S \cup P = V$ *(l'ensemble total des sommets)*
- $S \cap T = \emptyset$
- $s \in S$
- $p \in P$

Donc la coupe sépare la source du puits. Les arcs de la coupe sont ceux qui vont de S vers P.

(u,v) avec u ∈ S et v ∈ P

Ces arcs représentent les canaux par lesquels le flot peut passer de la source vers le puits. La capacité d'une coupe est la somme des capacités des arcs allant de S vers T. La quantité maximale de flot pouvant traverser cette séparation car tout flot allant de s vers p doit forcément traverser cette coupe.

Une ***coupe minimale*** est la coupe dont la **capacité est minimale** parmi toutes les coupes séparant s et p.

b) Théorème de Ford Fulkerson

Une coupe minimale indique **la limite maximale du flot possible dans le réseau)**

Théorème. Dans un réseau de transport, la capacité de la coupe minimale est égale à son flot maximal.

6.5 Conclusion

Dans ce chapitre, nous avons étudié le problème des flots dans les graphes orientés capacitaires. Nous avons introduit les notions essentielles de réseau de transport, de capacité, de flot valide, ainsi que les contraintes de capacité et de conservation qui garantissent la cohérence des flux dans un réseau. Ensuite, le problème du flot maximum a été abordé. Pour résoudre ce problème, nous avons présenté l'algorithme de Ford-Fulkerson, fondé sur la recherche successive de chaînes augmentantes permettant d'améliorer progressivement la valeur du flot jusqu'à l'obtention d'une solution optimale. Le chapitre a été enrichi par l'étude des coupes et des coupes minimales.

Série de TD n° 5

Exercice 1

Soit le graphe orienté de transport d'eau entre plusieurs stations, représenté par la matrice **A** ci-dessous. Dessiner le graphe et trouver le flot max, en mentionnant tous les détails de votre solution.

de	à	Capacité max
s	a	16
s	c	12
a	b	13
a	d	10
c	d	9
d	b	7
d	e	9
b	t	20
e	t	10

Exercice 2

Une usine de jus veut envoyer du jus depuis un réservoir principal vers une machine d'embouteillage. Le jus circule à travers un réseau de tuyaux, et chaque tuyau possède une capacité maximale (litres/heure). Le but est de calculer le flot maximal du nœud source vers le nœud destination sachant qu'on a les nœuds suivants :

- A : Réservoir principal (source)
- B, C : Stations intermédiaires
- D : Machine d'embouteillage (puits)

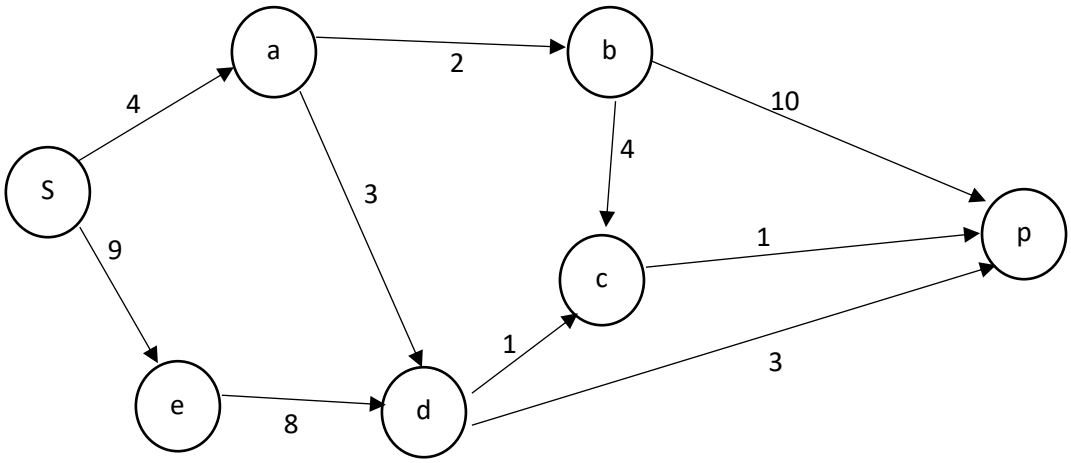
Avec les capacités des arêtes suivantes

- A → B : 10
- A → C : 5
- B → C : 15
- B → D : 10
- C → D : 10

1. Dessiner le **graphe orienté**.
2. Calculer le **flot maximal de A vers D** en utilisant la **méthode de Ford-Fulkerson** ou en simulant manuellement les chemins augmentants.

Exercice 3

Trouvez le flot max du graphe suivant en appliquant l'algorithme de Ford-Fulkerson. Mettez tous les détails de votre solution.



Chapitre 7
Problème d'ordonnancement

7.1 Introduction

Dans la gestion des projets modernes, la maîtrise du temps constitue un facteur essentiel de réussite. En effet, dans un projet les différentes activités doivent être exécutées selon un ordre précis tout en respectant des contraintes de durée et de dépendance. Le problème d'ordonnancement vise précisément à planifier ces activités de manière optimale afin de minimiser la durée globale du projet et d'assurer une utilisation efficace des ressources disponibles. Dans ce chapitre, nous introduirons les notions de projet, de tâche et de contrainte d'ordonnancement. Ensuite, nous verrons les méthodes de modélisation des problèmes de planification à l'aide des graphes potentiels de tâches, les calculs des dates au plus tôt et au plus tard, ainsi que les concepts de marges et de chemin critique. Enfin, nous présenterons les principales méthodes d'ordonnancement, notamment les approches PERT et MPM, largement utilisées dans le domaine de la gestion de projets.

7.2 Notions de : projet, tâches, contraintes logiques, contraintes temporelles

La planification de projet est une application importante de la théorie des graphes utilisée pour planifier et organiser les tâches dans le cadre de projets complexes. Elle permet aux gestionnaires et aux ingénieurs de déterminer l'ordre d'exécution des tâches, de calculer la

durée minimale du projet et d'identifier les activités critiques qui ont une incidence directe sur le délai d'exécution. Nous adopterons le vocabulaire et les concepts suivants :

Projet : Un projet est un ensemble d'activités coordonnées qui doivent être réalisées afin d'atteindre un objectif spécifique dans un délai donné. Il peut s'agir, par exemple, de projets de construction, de développement de logiciels ou de processus de fabrication.

Tâches : Une tâche (ou activité) est une unité de travail de base au sein d'un projet. Chaque tâche se caractérise par :

- ***une durée***
- ***des relations de précédence possibles avec d'autres tâches***

Par exemple :

Tâche	Description	Durée
A	Design	3 jours
B	Implémentation	5 jours
C	Test	2 jours

Contraintes logiques : Les contraintes logiques définissent les relations de priorité entre les tâches. Ces contraintes définissent l'ordre d'exécution des tâches. Par exemple :

- La tâche B ne peut pas commencer tant que la tâche A n'est pas terminée.
- La tâche C dépend de la tâche B.

Contraintes temporelles : elles imposent des restrictions de temps aux tâches et garantissent que la planification respecte les limitations du monde réel. Par exemple :

- La tâche A doit commencer après le jour 2.
- La tâche B doit être terminée avant le jour 10.

7.3 Identification du problème d'ordonnancement

Le problème de planification d'un projet consiste à déterminer :

- L'heure de début de chaque tâche
- L'ordre d'exécution des tâches
- La durée minimale de réalisation du projet

Étant donné :

- un ensemble de tâches
- la durée des tâches
- les contraintes de priorité

L'objectif est d'établir un calendrier qui respecte toutes les contraintes tout en minimisant la durée totale du projet. La théorie des graphes modélise ce problème à l'aide de graphes orientés où :

- les sommets représentent les tâches
- les arêtes représentent les contraintes de priorité

7.4 Ecriture des contraintes sous forme d'inéquations

Les contraintes de planification peuvent être exprimées à l'aide d'inégalités mathématiques. Soit :

ti = heure de début de la tâche i
di = durée de la tâche i

Si la tâche j dépend de la tâche i, alors la tâche j ne peut commencer qu'après la fin de la tâche i. La contrainte devient alors : **$tj \geq ti + di$**

Cette inégalité garantit que le début de la tâche j intervient après la fin de la tâche i.

7.5 Les tâches fictives

Une tâche fictive (ou tâche fictive) est une tâche qui a une durée nulle et qui représente uniquement une dépendance logique. Les tâches fictives sont introduites lorsqu'il est nécessaire de représenter correctement les relations de priorité dans un graphique de planification. L'objectif des tâches fictives est d'éviter toute ambiguïté dans les dépendances en maintenant des relations correctes entre les tâches.

Exemple : Si deux tâches partagent certains prédécesseurs, mais pas tous, une tâche fictive peut être ajoutée pour représenter la structure correcte.

7.6 Le graphe potentiel de tâches

Le graphe potentiel de tâches est un graphe orienté utilisé pour représenter les contraintes de planification. Il a les caractéristiques suivantes :

- chaque sommet représente une tâche
- chaque arête représente une contrainte de précédence
- les poids des arêtes correspondent à la durée des tâches

Le graphe doit satisfaire les conditions suivantes :

- aucun cycle positif
- cohérence avec toutes les contraintes de planification

Ce graphe nous permet de calculer :

- les dates de début les plus précoces
- les dates de début les plus tardives
- les marges de temps

7.7 Dates au plus tôt, dates au plus tard, marge totale, marge libre

- Date de début au plus tôt : La date de début au plus tôt d'une tâche est la date la plus précoce à laquelle elle peut commencer tout en respectant toutes les contraintes. Elle est obtenue en calculant le chemin le plus long entre le nœud de départ et la tâche dans le graphe.
- Date de début au plus tard : La date de début au plus tard est la date la plus tardive à laquelle une tâche peut commencer sans retarder l'ensemble du projet. Elle est calculée en remontant à partir de la date d'achèvement du projet.
- Marge totale : La marge totale d'une tâche mesure le délai maximal pendant lequel une tâche peut être retardée sans retarder l'achèvement du projet. Elle est estimée selon la formule suivante :

  Marge totale = Date de début au plus tard – Date de début au plus tôt

- Marge libre : Elle correspond au retard autorisé sans affecter le début au plus tôt des tâches suivantes. Elle est estimée selon la formule suivante :

Marge libre = Date au plus tôt de la tâche suivante – Date au plus tôt de la tache + durée

Les valeurs des marges permettent d'identifier les tâches flexibles et celles qui sont critiques.

7.8 Tâches critique et chemin critique

Les **tâches critiques** sont les activités du projet qui :

- n'ont aucune marge de retard (marge totale = 0)
- doivent impérativement être réalisées à temps
- sinon, tout le projet sera retardé

Si une tâche critique prend du retard, la date de fin du projet est directement impactée.

Le **chemin critique** est :

- la suite des tâches critiques
- le plus long chemin entre le début et la fin du projet
- celui qui détermine la durée minimale totale du projet

C'est le chemin qui contrôle entièrement le planning.

7.9 Les méthodes d'ordonnancement de projet : MPM et Pert

Il existe deux méthodes utilisées en **ordonnancement de projet**. Nous allons exposer la différence principale entre elles puis nous allons voir les détails de l'une d'elle.

La méthode **PERT** (Program Evaluation and Review Technique) : est une technique de planification qui permet de :

- **organiser les tâches d'un projet**
- **analyser les durées**
- **identifier le chemin critique**

Elle se caractérise par :

- Représentation sous forme de **graphe avec des nœuds (événements)**
- Les **tâches sont représentées par des arcs (flèches)**
- Utilise souvent **3 estimations de durée** : optimiste, pessimiste, la plus probable

Elle est surtout utilisée quand les durées sont **incertaines** (projets de recherche, innovation…).

La méthode **MPM** (Méthode des Potentiels Métra) est une autre technique d'ordonnancement qui permet aussi de :

- planifier les tâches
- déterminer les dates de début et de fin
- trouver le chemin critique

Elle se caractérise par :

- une représentation sous forme de **graphe avec des sommets = tâches**
- Les **liaisons (arcs)** indiquent les contraintes d'ordre
- Chaque tâche est associée à une **durée directement sur le sommet**

Contrairement à PERT, dans MPM, les tâches sont sur les nœuds (et non sur les arcs)

Nous allons exposer ci-dessous les détails de la méthode Pert vu qu'elle est plus utilisée universellement.

a) Composants d'un Graphe PERT

Un graphe PERT est un triplet **R=(X, U, d)**

- **X** : un ensemble **d'Etapes**
- **U** : un ensemble de **Tâches**
- **d** : un ensemble de valeurs qui représentent les **Durées** des Tâches

La Tâche ….

- est un élément de base dans un graphe PERT
- Elle signifie le déroulement d'une opération dans le temps
- Elle est représentée par une flèche.
- Elle doit avoir un nom (code) et une durée connue
- Le nom et la durée sont mentionnés sur la flèche
- Elle doit être délimitée obligatoirement par une étape antérieure et une étape postérieure

Exemple :

Figure 7.1 illustration d'une tâche

2 tâches peuvent être :

- Successives : elles s'exécutent l'une après l'autre

Figure 7.2 illustration de 2 tâches successives

- Simultanées : elles s'exécutent en parallèle et elles ont la même étape antérieure

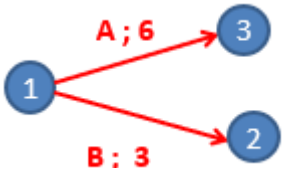


Figure 7.3 illustration de 2 tâches simultanées

- Convergentes : elles arrivent sur la même étape postérieure

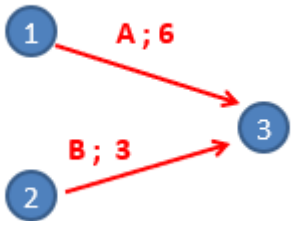


Figure 7.4 illustration de 2 tâches convergentes

La tâche Fictive …

- Elle n'existe pas en réalité,
- Elle est représentée par une flèche en pointillés

- ❑ Elle a une durée nulle
- ❑ On l'utilise pour certaines raisons d'organisation du graphe PERT
- ❑ Elle Indique les contraintes d'enchaînement entre certaines étapes

L' Etape…

- ➢ On l'appelle aussi : nœuds… évènement… sommet
- ➢ Un autre élément de base aussi dans un graphe PERT
- ➢ Elle représente le début ou la fin d'une tâche
- ➢ Elle n'a pas de durée
- ➢ Chaque tâche possède une étape de début et une étape de fin
- ➢ Elle est représentée généralement par un cercle
- ➢ En haut du cercle on mentionne le n° de l'étape
- ➢ chaque tâche doit obligatoirement être **précédée** et **suivie** par une étape

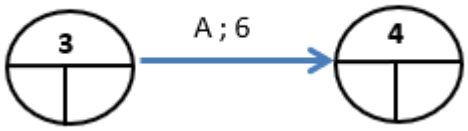


Figure 7.5 tâche A commence après l'étape 3 et se termine à l'étape 4

- ➢ En bas à gauche du cercle la date (durée) **au plutôt**,
- ➢ et à droite la date (durée) **au plus tard** de la tâche

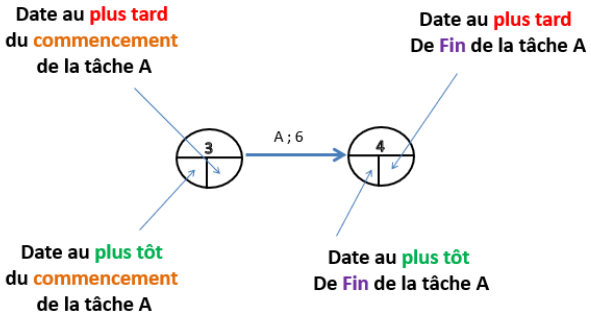


Figure 7.6 Dates au plus tôt

- ➢ Date au plus tôt de l'étape **Début** = 0
- ➢ Date au plus tôt de l'**étape 1** = date au plus tôt de l'étape **Début** + **durée** de **A**

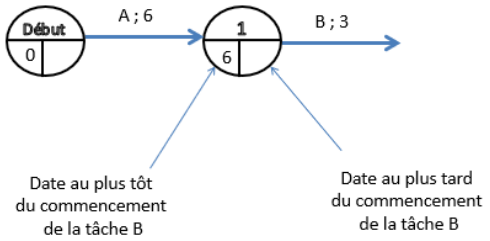


Figure 7.7 Dates au plus tard

b) Les Etapes de Construction d'un PERT

Les différentes étapes pour la construction d'un graphe PERT

1. Trouver les différents **Niveaux** du Graphe Pert à partir du tableau d'**Antériorité** (qui est donné)
2. Trouver le tableau de **Postériorité**
3. Dessiner les **Etapes** du graphes (les sommets)
4. Dessiner les **Tâches**
5. Calculer les **dates au plus tôt** des Tâches
6. Calculer les **dates au plus tard** des Tâches
7. Calculer les **marges**

7.10 Conclusion

Au cours de ce chapitre, nous avons étudié les principes fondamentaux de l'ordonnancement de projets. Nous avons défini les notions de projet, de tâche et de contraintes de précédence puis nous sommes passés à une introduction des concepts essentiels de dates au plus tôt, de dates au plus tard, de marges, offrant ainsi les moyens d'évaluer la flexibilité des tâches au sein d'un projet. Nous avons également mis en évidence l'importance des tâches critiques et du chemin critique, dont la maîtrise est indispensable pour garantir le respect des délais de réalisation. Enfin, la présentation des méthodes d'ordonnancement PERT et MPM a montré comment les graphes constituent un outil puissant d'aide à la décision dans la gestion de projets.

Série de TD n° 6

Exercice n°1

Tracer le réseau PERT du projet composé des tâches suivantes.

Tâches	Antériorités	NIVEAUX
A	**-**	
B	**-**	
C	**B, A**	
D	**A**	
E	**D**	
F	**C**	

Exercice n°2

La construction d'une maison nécessite la réalisation d'un nombre de tâches dont les durées de réalisation et les contraintes de précédence sont données dans le tableau suivant :

Tâches	Désignation	Durée	Tâches antérieures
A	Obtenir des briques	5	/
B	Obtenir des toits	12	/
C	Préparer les fondations	7	/
D	La coquille droite	10	A C
E	La construction du toit	4	D B
F	Travailler Les égouts	7	C
G	Installation	10	D
H	Plâtrer	6	I E G
I	Sanitaires	12	F D
J	Parqueter	5	I G E
K	Aménager le parc	2	N
L	La peinture	6	M J
M	La menuiserie	2	H
N	L'allée	2	D F

1) *Dresser le tableau des niveaux et des postériorités*
2) *Dresser le graphe PERT de ce projet*
3) *Calculer les dates au plus tôt et les dates au plus tard des tâches*
4) *Calculer les marges de chacune des tâches*

Corrigé type de la Série de TD n°1

Exercice 1

a) Démontrez que le nombre de sommets ayant un degré impair est pair.

Réponse : nous avons le théorème suivant :

$$\sum_{xi \in X}^{i=1,n} d(xi) = 2 * taille(G)$$

Donc :

$$\sum_{xi \in X,\, d(xi)\ pair}^{i=1,n} d(xi) + \sum_{xi \in X,\, d(xi)\ impair}^{i=1,n} d(xi) = 2 * taille(G)$$

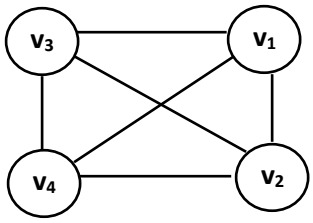
En supposant :

$$\sum_{xi \in X,\, d(xi)\ pair}^{i=1,n} d(xi) = A \qquad \sum_{xi \in X,\, d(xi)\ impair}^{i=1,n} d(xi) = B$$

Nous aurons obligatoirement : Soit A et B sont tous les 2 pairs ou tous les 2 impairs car leur somme est paire. Or nous sommes sûrs que A est pair car elle est la somme de nombres pairs. Donc automatiquement B doit être pair, et comme il est la somme de nombres impairs donc impérativement le nombre des valeurs sommées est pair, et alors le nombre de sommets de degrés impair est pair.

b) Peut-on trouver un groupe de 4 personnes tel que chaque personne est amie avec exactement trois autres personnes ?

Réponse : Oui, nous pouvons trouver un graphe de 4 sommets (personnes) dont chacun possède un degré = 3 (3 amis). Comme le montre le graphe ci-dessous.



c) Peut-on trouver un groupe de cinq personnes tel que chaque personne est amie avec exactement trois autres personnes ?

Réponse : D'après la réponse a), pour que chaque personne aies 3 amis (degré impair), il faut que le nombre de personnes (de sommets) soit pair. Or on parle ici du nombre de 5. Donc impossible.

Exercice 2

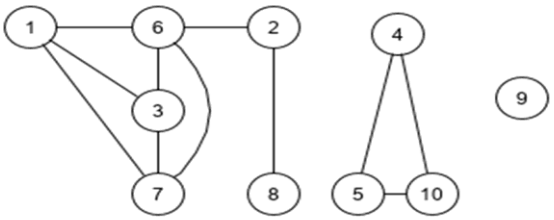
Étant donné un groupe de 10 personnes, le tableau suivant indique les paires de personnes qui ont une relation d'amitié.

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

a) **Représentez cette situation par un graphe G**.

Réponse :

- Un graphe non-orienté est défini par une paire (X, E).
- Les sommets (X) dans le graphe G représentent les personnes.
- La relation d'amitié entre les personnes est représentée par des arêtes (U) dans G



b) Donnez un sommet pendant et un sommet isolé de G.

Réponse : le sommet 9 est isolé (son degré est 0), le sommet 8 est pendant (son degré est 1).

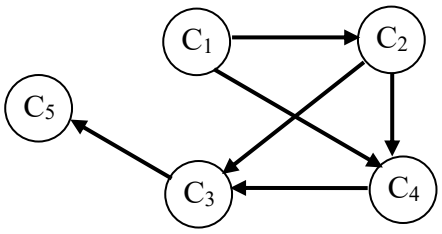
c) **Vérifiez qu'il existe au moins deux personnes ayant le même nombre d'amis**.

Réponse :

deg(1)= deg(3)= deg(7)= 3 amis : Donc les personnes 1, 3 et 7 ont le même nombre d'amis.

deg(2)=deg(4)=deg(5)=deg(10) = 2 amis : Donc les personnes 2, 4, 5 et 10 ont le même nombre d'amis

2.2 Illustration de la situation par un graphe



Exercice 3

Construisez un graphe non orienté ayant au moins deux sommets et tel que tous les sommets ont des degrés distincts. Qu'en déduisez-vous ?

Réponse : On s'aperçoit rapidement que c'est impossible. Ainsi, dans un graphe, il existe toujours deux sommets de même degré.

Nous allons montrer **qu'il n'existe aucun graphe simple** dont tous les sommets ont **des degrés distincts** :

Supposons qu'un tel graphe **existe ;** et qu'il possède **n sommets.** En triant les degrés dans un ordre croissant, on a nécessairement :

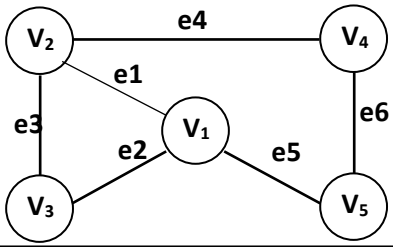
a) **les degrés 0, 1, 2, …, n-1. :** Or, la présence d'un sommet de degré 0, empêche la présence d'un sommet de degré n-1 ! **On obtient ainsi une contradiction**
b) **les degrés 1, 2, …, n :** Pour le sommet de degré n, il aura surement une boucle ce qui est contradictoire avec le faite que le graphe est simple.

De manière plus évidente :

Il y a n valeurs prises dans un ensemble de n-1 valeurs. Ainsi, il y a forcément au minimum une valeur qui se répète. Ceci signifie qu'il y a au moins deux sommets ayant le même degré.

Exercice 4

a) Donnez la matrice d'adjacence et la matrice d'incidence du graphe non orienté suivant :

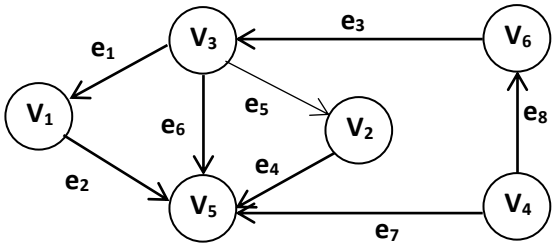


Réponse

	V_1	V_2	V_3	V_4	V_5
V_1	0	1	1	0	1
V_2	1	0	1	1	0
V_3	1	1	0	0	0
V_4	0	1	0	0	1
V_5	1	0	0	1	0

	e1	e2	e3	e4	e5	e6
V_1	1	1	0	0	1	0
V_2	1	0	1	1	0	0
V_3	0	1	1	0	0	0
V_4	0	0	0	1	0	1
V_5	0	0	0	0	1	1

b) Donnez la matrice d'adjacence et la matrice d'incidence du graphe orienté suivant :



Réponse

	V1	V2	V3	V4	V5	V6
V1	0	0	0	0	1	0
V2	0	0	0	0	1	0
V3	1	1	0	0	1	0
V4	0	0	0	0	1	1
V5	0	0	0	0	0	0
V6	0	0	1	0	0	0

	e1	e2	e3	e4	e5	e6	e7	e8
V1	1	-1	0	0	0	0	0	0
V2	0	0	0	-1	1	0	0	0
V3	-1	0	1	0	-1	-1	0	0
V4	0	0	0	0	0	0	-1	-1
V5	0	1	0	1	0	1	1	0
V6	0	0	-1	0	0	0	0	1

c) Représentez graphiquement les graphes associés aux matrices suivantes :

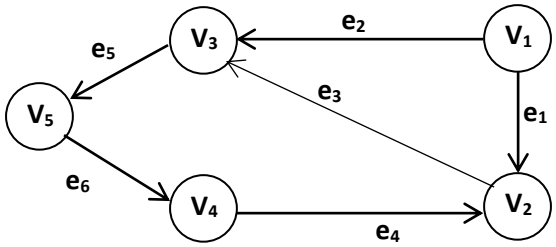
-1	-1	0	0	0	0
1	0	-1	1	0	0
0	1	1	0	-1	0
0	0	0	-1	0	1
0	0	0	0	1	-1

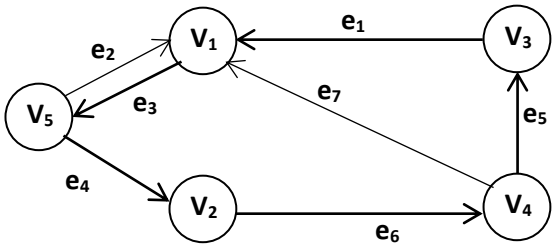
A

1	1	-1	0	0	0	1
0	0	0	1	0	-1	0
-1	0	0	0	1	0	0
0	0	0	0	-1	1	-1
0	-1	1	-1	0	0	0

B

Réponse :

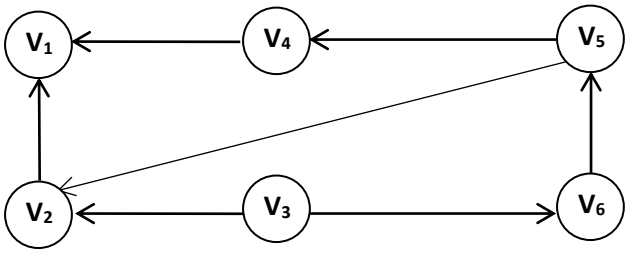




d) Représentez graphiquement le graphe associé à la matrice suivante :

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

Réponse :



Corrigé de la série de TD n°2

Exercice 1 :

On utilise la **formule d'Euler** : n – m + f = 2 ; avec n = 12 et m = 18 alors f = 8 = nbre de faces

Exercice 2 :

Dans un graphe planaire simple, le pourtour de chaque face a au moins 3 arêtes. En comptant les arêtes de toutes les faces, chaque arête est comptée 3 fois, et sachant que chaque arête intervient dans 2 faces nous avons la relation : **3f ≤ 2m.**

$$3f \leq 2m \Rightarrow 3(m - n + 2) \leq 2m \Rightarrow 3m - 3n + 6 \leq 2m \Rightarrow \mathbf{m \leq 3n - 6}$$

Si le graphe est biparti, les pourtours des faces ont au moins 4 arêtes et on a donc : **4f ≤ 2m**

$$4f \leq 2m \Rightarrow 4(m - n + 2) \leq 2m \Rightarrow 4m - 4n + 8 \leq 2m \Rightarrow \mathbf{m \leq 2n - 4}$$

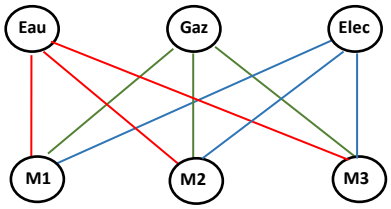
Exercice 3

le graphe complet K_5 n'est pas planaire car : $m(K_5) = 5 * (5-1)/2 = \mathbf{10 > 9} = 3*5 - 6$

le graphe biparti complet $K_{3,3}$ n'est pas planaire car : $m(K_{3,3}) = (3^{3-1} + 3^{3-1})/2 = \mathbf{9 > 8} = 2*6 - 4$

Exercice 4

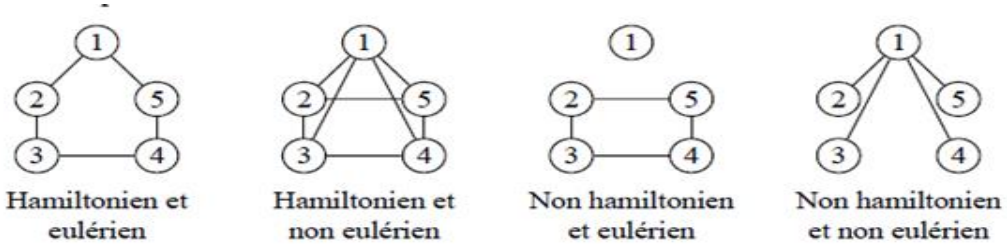
D'après l'énoncé, nous cherchons à savoir si le graphe biparti complet K3,3 est planaire ?



graphe biparti complet $K_{3,3}$

La réponse est que ce graphe biparti n'est pas planaire car d'après la formule ci-dessus, il doit avoir au plus 8 arêtes alors qu'il en a 9.

Exercice 5



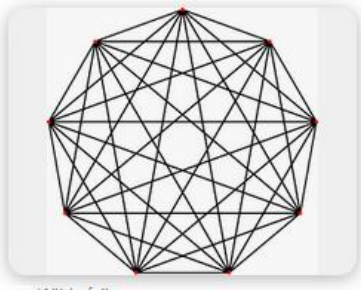
Hamiltonien et eulérien | Hamiltonien et non eulérien | Non hamiltonien et eulérien | Non hamiltonien et non eulérien

Exercice 6

1) **OUI**, il existe une chaîne **eulérienne car le graphe contient 2 sommets avec des degrés impairs A et D**.
   On peut facilement trouver cette chaîne :
   A – C – D – A – B – G – C – F – G – E – F – D
2) **NON,** le guide ne peut pas emprunter tous les tronçons de route en passant une et une seule fois par chacun d'eux et en terminant le circuit à l'hôtel. **Pas de cycle eulérien** car les sommets **A et D n'ont pas des degrés pairs**.

Exercice 7

1) Désignons par les chiffres de 1 à 9 les personnes et considérons **le graphe complet K_9 à 9 sommets.**
2) 1 sommet c'est un joueur,
3) 1 arête signifie que les 2 joueurs sont voisins dans la composition de la table



Une composition de la table correspond à un cycle hamiltonien de K9. Il faut un passage une et une seule fois par chaque sommet (joueur), et revenir au 1er sommet.

Pourquoi une et une seule fois par joueur dans ce jeu ?? Car sinon, le joueur aura 3 adjacents !!

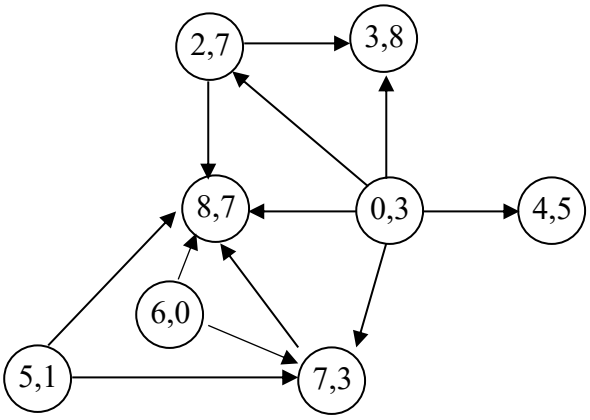
Si 2 compositions de table correspondent à 2 cycles ayant une arête commune, cela signifie que les 2 personnes reliées par cette arête se retrouvent cote à cote à 2 reprises

Ainsi, le problème revient à déterminer : **le nombre de cycles hamiltoniens disjoints de K_9**

Le graphe K_9 possédant 9 x (8/2) = **36 arêtes** et chaque cycle utilisant **9 arêtes**, ce nombre est au maximum égal **4**

Exercice 8

1. Construction du graphe.



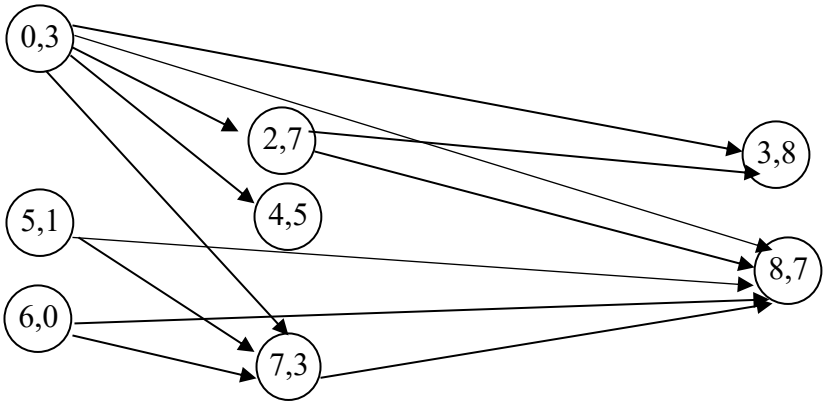
2. Le graphe est transitif. La preuve

Pour que le graphe soit transitif il faut que la relation de domination soit transitive.

Effectivement, nous avons :

(a, b) R (c, d) $\Rightarrow$ a < c et b $\leq$ d
(c, d) R (z, t) $\Rightarrow$ c < z et d $\leq$ t ; Donc : a < z et b $\leq$ t $\Rightarrow$ (a, b) R (z, t)

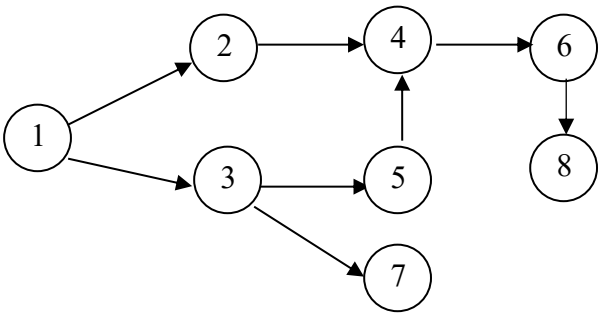
3. Décomposition en niveaux.



4. Une paire est pareto si elle n'a pas de Prédécesseurs. Donc les sommets du niveau 1

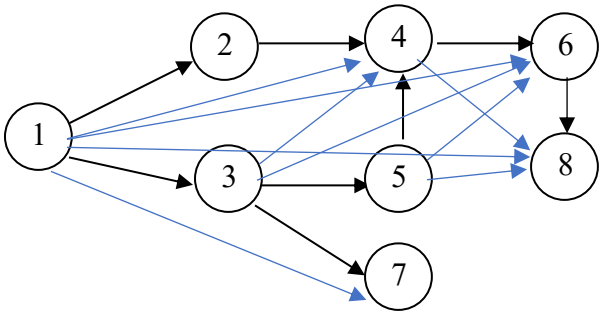
Exercice 9

1. Le graphe des contaminations



2. Le graphe n'est pas fortement connexe, il y a des sommets depuis lesquels on ne peut pas atteindre d'autres, exemple, les sommets 7 et 8.

3. La fermeture transitive G*

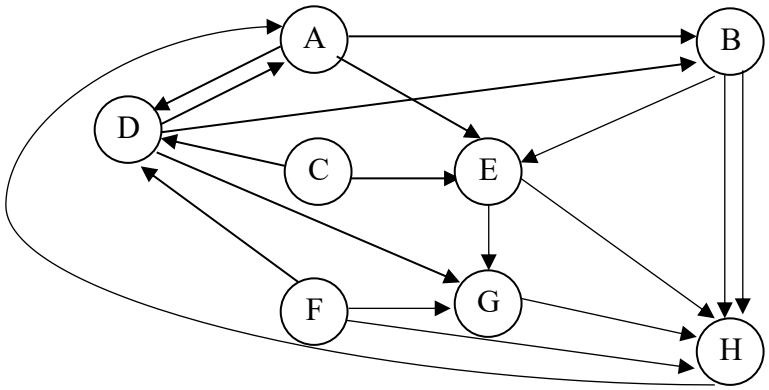


4. Le plus long chemin de G* est 1-3-5-4-6-8, et il est de longueur = 5.

Etant donné que G ne contient pas de circuit, la longueur du plus long chemin représente le nombre de niveaux après une décomposition en niveaux

Exercice 10

1. Construction du graphe



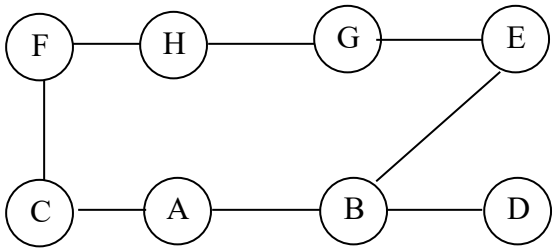
2. Le graphe n'est pas fortement connexe. Par exemple, on ne peut pas aller de A vers C
3. Les composantes fortement connexes

itération	k	A	B	C	D	E	F	G	H	Comp.fort.connexe
1	**1**	+ -	+-	-	+ -	+-	-	+-	+-	C_1 ={A,B,D,E,G,H}
2	**2**			+-						C_2 ={C}
3	**3**						+-			C_3 ={F}

4. F peut être un spammeur car il envoie 3 messages et il n'est jamais successeur. Un spammeur est une personne qui se trouve seule dans une composante connexe

Exercice 11

1 Représentation du graphe



2 **Parcours en largeur** : Le parcours en largeur (BFS) explore le graphe niveau par niveau. Il utilise une file (FIFO). On visite d'abord tous les voisins directs, puis les voisins des voisins.

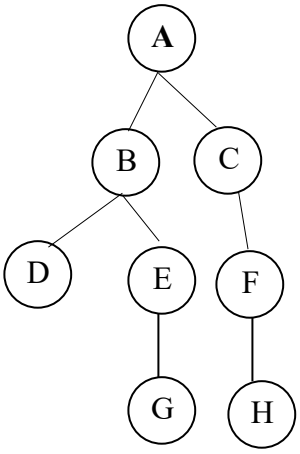
Étapes du BFS à partir de A

Initialisation

- File = [A]
- Visités = {A}
- Ordre = A

Etape	Sommet à traiter	Voisins du sommet	Ajouter dans la file	Contenu de la file	Sommets visités	Ordre
1	A	B, C	B, C	[B, C]	{A, B, C}	A, B, C
2	B	A, D, E (A déjà visité)	D, E	[C, D, E]	{A, B, C, D, E}	A, B, C, D, E
3	C	A, F (A déjà visité)	F	[D, E, F]	{A, B, C, D, E, F}	A, B, C, D, E, F
4	D	B (déjà visité)		[E, F]		
5	E	B, G (B déjà visité)	G	[F, G]	{A, B, C, D, E, F, G}	A, B, C, D, E, F, G
6	F	C, H, (C déjà visité)	H	[G, H]	{A, B, C, D, E, F, G, H}	A, B, C, D, E, F, G, H
7	G	E, H (déjà visités)		[H]		
8	H	Voisins déjà visités		[]		A,B,C,D,E,F,G,H

Arbre BFS
Le BFS donne l'**arbre des plus courts chemins** à partir de A.
Par exemple : Distance(A, D) = 2 ; Distance(A, H) = 3



Parcours en profondeur : Le parcours en profondeur (DFS) explore une branche le plus loin possible avant de revenir en arrière. Il utilise une pile ou la récursivité. On descend en profondeur avant d'explorer les voisins.

Étapes du DFS à partir de A

Départ : A
Voisins : B, C
On choisit **B** (ordre alphabétique). ; Ordre = A

Depuis B
Voisins : A, D, E (A déjà visité) ; On va vers **D ;** Ordre = A, B

Depuis D
Voisin : B (déjà visité) On revient à B.

Depuis B
Prochain voisin non visité : **E ;** Ordre = A, B, D

Depuis E
Voisins : B, G ; (B visité) ; On va vers **G ;** Ordre = A, B, D, E

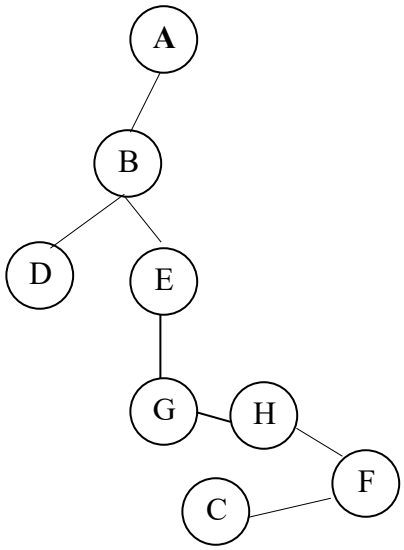
Depuis G
Voisins : E, H ; (E visité) ; On va vers **H ;** Ordre = A, B, D, E, G

Depuis H
Voisins : F, G ; (G visité) ; On va vers **F ;** Ordre = A, B, D, E, G, H

Depuis F
Voisins : C, H ; (H visité) ; On va vers **C ;** Ordre = A, B, D, E, G, H, F

Depuis C
Voisins : A, F ; (déjà visités)

Ordre DFS final : A,B,D,E,G,H,F,C



Corrigé de la Série de TD n°3

Exercice 1

1. Graphe 1 = **arbre** (non-orienté + connexe + sans cycle)

   Graphe 2 = **anti-arborescence** (orienté et il y a un sommet qui a deux prédécesseurs)

   Graphe 3 = **une forêt d'arborescences** (dans le cas où c'est un graphe non-orienté, chaque composante connexe est un arbre)

   Graphe 4 = **arborescence** (orienté où chaque sommet a un seul prédécesseur et il procède une racine). Si les arcs sont transformés en arêtes alors le graphe est un arbre.

2. Graphe 1 = graphe **biparti complet**, nombre d'arbres $=m^{n-1}.n^{m-1}= 2.2= 4$ arbres

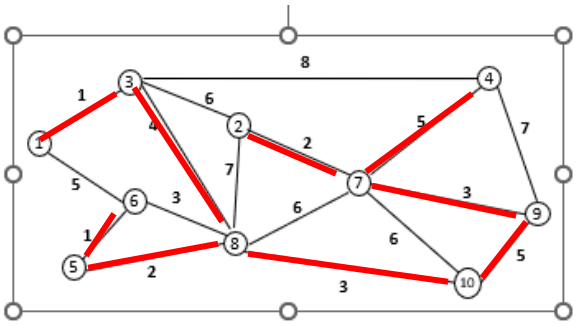
   Graphe 2 = graphe **complet**, nombre d'arbres $n^{n-2} = 4^2= 16$ arbres

Exercice 2

Application de kruskal

arête	1-3	5-6	2-7	5-8	6-8	7-9	8-10	3-8	1-6	4-7	9-10	3-2	7-8	7-10	2-8	4-9	3-4
poids	1	1	2	2	3	3	3	4	5	5	5	6	6	6	7	7	8
indice	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

k	Décision	Poids
1	Rajouter 1-3	1
2	Rajouter 5-6	1
3	Rajouter 2-7	2
4	Rajouter 5-8	2
5	Ne pas rajouter 6-8	-
6	Rajouter 7-9	3
7	Rajouter 8 -10	3
8	Rajouter 3-8	4
9	Ne pas rajouter 1-6	-
10	Rajouter 4-7	5
11	Rajouter 9-10	5
	N = 10 sommets, M= 10 -1 arêtes	Total = 26



Le poids minimal = 26

Exercice 3

Calcul des coûts des arêtes.

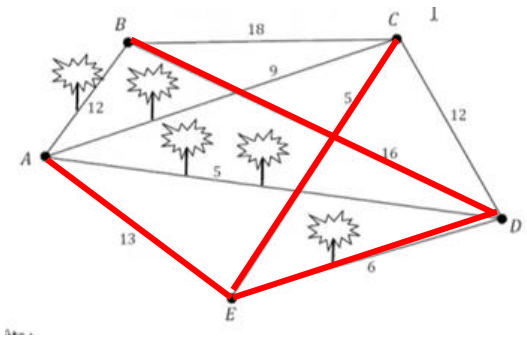
Arête	Coût
A-B	12*6 + 30 = 102
A-C	9*6 + 30 = 84
A-D	5*6 + 30*2 = 90
A-E	13*6 = 78
B-C	18*6 = 108
B-D	16 * 6 = 96
C-D	12*6 = 72
C-E	5*6 = 30
D-E	6*6 +30 = 66

Application de l'algorithme de Kruskal

Nous avons n=5 sommets, l'arbre partiel sera composé de k = n-1 = 4 arêtes

Arête	C-E	D-E	C-D	A-E	A-C	A-D	B-D
Poids	30	66	72	78	84	90	96
indice	1	2	3	4	5	6	7

k	Décision sur l'arête	Cout
1	Rajouter C-E	30
2	Rajouter D-E	66
3	Ne pas Rajouter C-D	
4	Rajouter A-E	78
5	Ne pas rajouter A-C	
6	Ne pas Rajouter A-D	
7	Rajouter B-D	96
	Total	270



Exercice 4

***T* = Ø**
***A* = {0}**

Itération	A	T	Poids de l'arbre
1	0,1	0-1	
2	0,1,4	0-1, 1-4	
3	0,1,4,3	0-1, 1-4, 4-3	

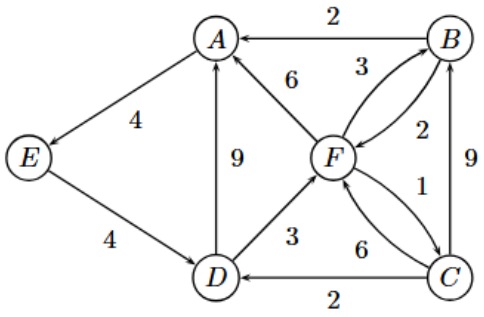
4	0,1,4,3,7	0-1, 1-4, 4-3, 4-7	
5	0,1,4,3,7,8	0-1, 1-4, 4-3, 4-7, 7-8	
6	0,1,4,3,7,8,2	0-1, 1-4, 4-3, 4-7, 7-8, 8-2	
7	0,1,4,3,7,8,2,5	0-1, 1-4, 4-3, 4-7, 7-8, 8-2, 4-5	
8	0,1,4,3,7,8,2,5,6	0-1, 1-4, 4-3, 4-7, 7-8, 8-2, 4-5, 5-6	

Les sommets sont tous couverts.

Corrigé de la Série de TD n°4

Exercice 1

Plus court chemin en appliquant Dijkstra

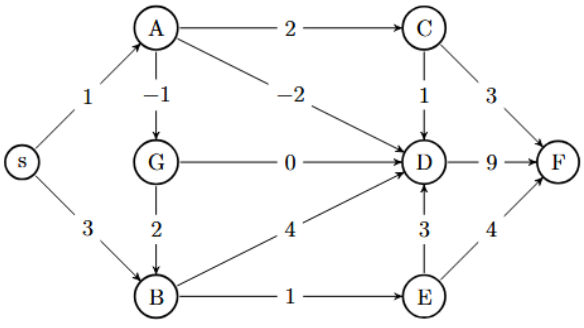


C	B	F	D	E	A	**Chaines minimales**
C(0)						**C(0)**
x	CB(9)	CF(6)	**CD(2)**			**CD(2)**
x		**CDF(5)**	x		CDA(11)	**CDF(5)**
x	**CDFB(8)**	x	x		CDFA(11)	**CDFB(8)**
x	x	x	x		CDFBA(10)	**CDFBA(10)**
x	x	x	x	**CDFBAE(14)**	x	**CDFBAE(14)**
x	x	x	x	x	x	

- la chaîne la plus courte de C à E est donc CDF BAE avec un poids de 14
- la chaîne la plus courte de C à A est donc CDF BA avec un poids de 10
- la chaîne la plus courte de C à B est donc CDF B avec un poids de 8
- la chaîne la plus courte de C à F est donc CDF avec un poids de 5
- la chaîne la plus courte de C à D est donc CD avec un poids de 2

Exercice 2

L'exécution de Bellman peut être résumée par le tableau suivant, où dans chaque case, on note les modifications liées au traitement de la tête de file de d, la distance par rapport à s et npred le nombre de prédécesseurs restant à traiter : d/npred



File F	s	A	B	C	D	E	F	G
	0/0	∝/1	∝/2	∝/1	∝/4	∝/1	∝/3	∝/1
s		1/0	3/1					
A				3/0	−1/4			0/0
C					−1/3		6/2	
G			2/0		−1/2			
B					−1/1	3/0		
E					−1/0		6/1	
D							6/0	
F								

L'ordre d'entrée dans la file F est donc s, A, C, G, B, E, D, F .

L'arborescence des plus courts chemins et le tableau des distances sont donnés ci-dessous :

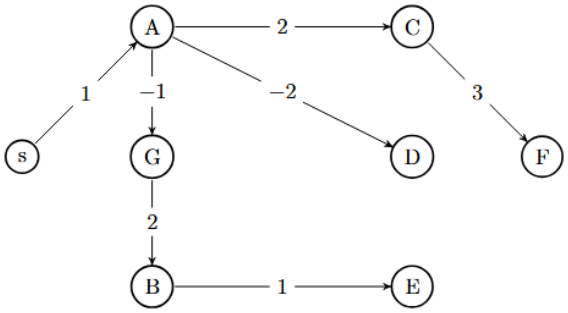


Tableau des distances :

s	A	B	C	D	E	F	G
0	1	2	3	−1	3	6	0

Exercice 3

Arc	Poids
A → B	6
A → C	5
A → D	5
B → E	-1
C → B	-2
C → E	1
D → C	-2
D → F	-1
E → F	3

Initialisation

Sommet	A	B	C	D	E	F
d	0	∝	∝	∝	∝	∝
p	–	–	–	–	–	–

Relaxation : résultat final après application de l'lagorithme

Sommet	A	B	C	D	E	F
d	0	3	3	5	2	4
p	-	C	D	A	B	D

Chemins les plus courts :

$A \rightarrow D \rightarrow C \rightarrow B$ (coût 3)

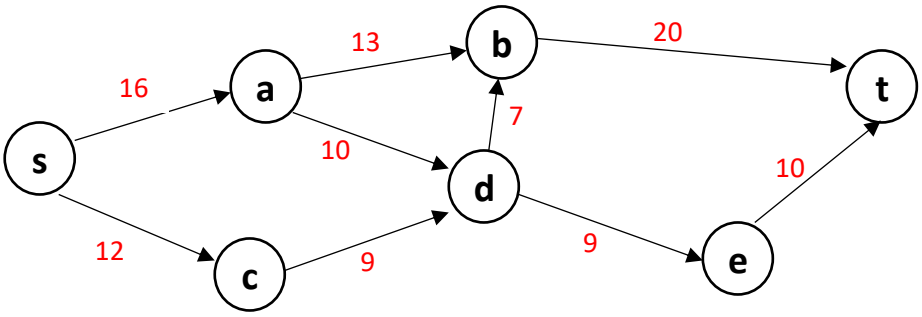
$A \rightarrow D \rightarrow C \rightarrow B \rightarrow E$ (coût 2)

$A \rightarrow D \rightarrow F$ (coût 4)

Corrigé de la Série de TD n°5

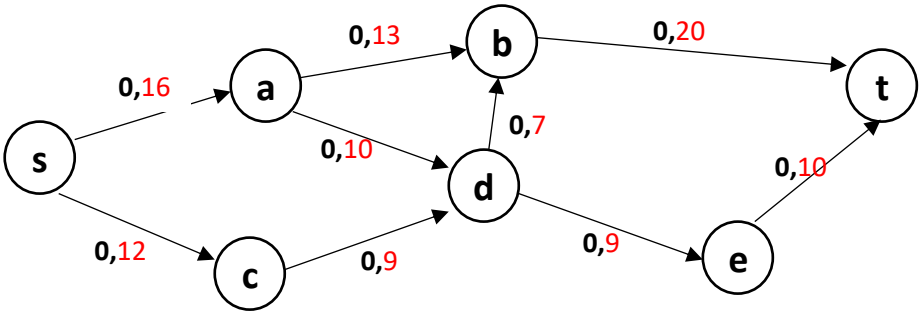
Exercice 1

Le graphe d'après la matrice A :



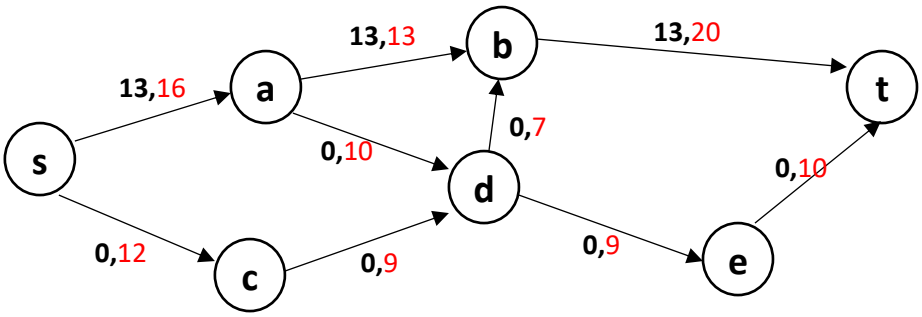
Algorithme de Ford Fulkerson pour trouver le flot max.

Itération 1 : Flot null 1 pt



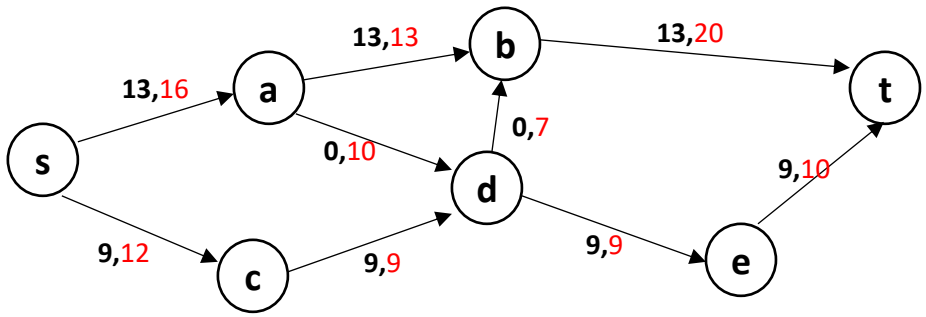
Itération 2 : on prend la chaine augmentante s, a, b, t

La plus petite capacité qu'on peut ajouter = 13. On l'ajoute au flot. **1 pt**



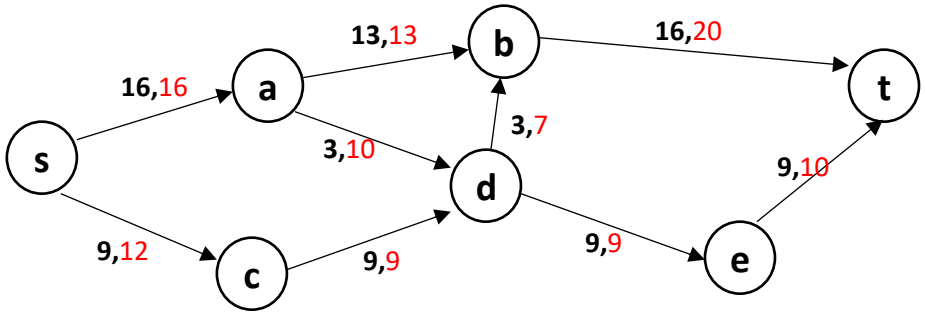
Itération 3 : on prend la chaine augmentante s, c, d, e, t

La plus petite capacité qu'on peut ajouter = 9. On l'ajoute au flot. **1 pt**



Itération 4 : on prend la chaine augmentante s, a, d, b, t

La plus petite capacité qu'on peut ajouter = 3. On l'ajoute au flot. **1 pt**



Les arcs (s,a) et (c,d) sont saturés, ils empêchent l'amélioration du flot.

Donc le flot max de ce réseau = 16 + 9 = 25 1 pt

Exercice 2

Étape 1 : chemin A → B → D

Capacités : A → B = 10 ; B → D = 10

Capacité minimale : min(10,10)=10 , On envoie **10** unités de flot.

Flot actuel : A→B = 10 , B→D = 10

Flot total = **10**

Étape 2 : chemin A → C → D

Capacités restantes : A → C = 5 ; C → D = 10

Capacité minimale : min(5,10)=5, On envoie **5** unités.

Flot actuel ajouté = **5 ;** Flot total = 10+5=1510 + 5 = 1510+5=15

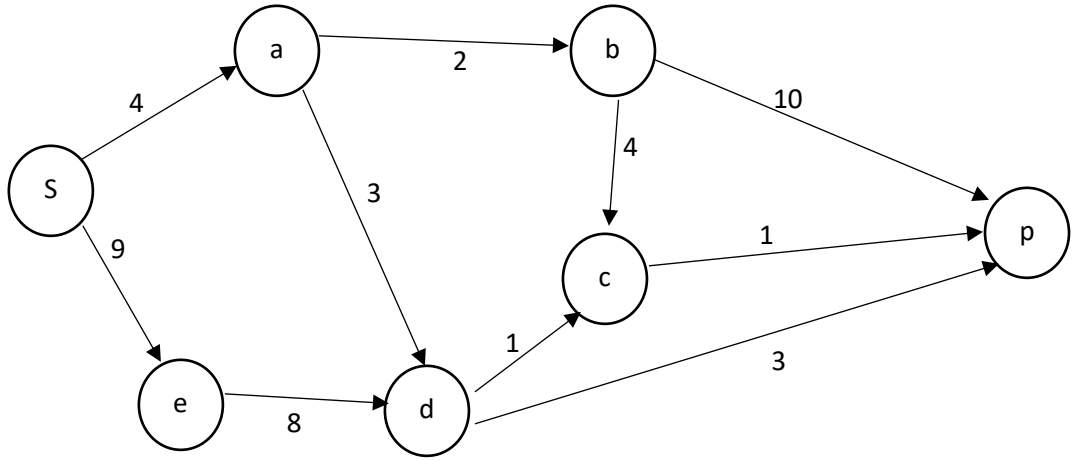
Étape 3 : vérifier d'autres chemins

Capacité restante : A → B = **0,** A → C = **0**

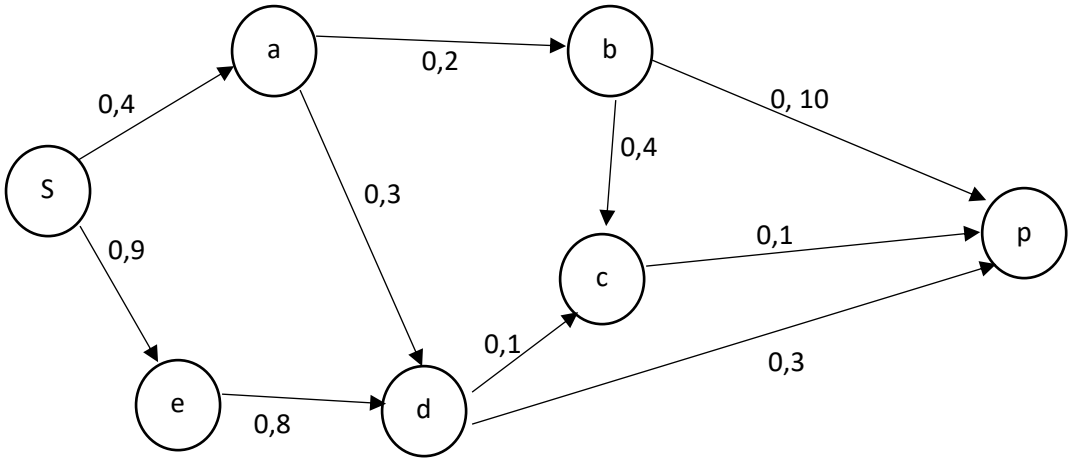
Donc aucun autre chemin partant de A n'a de capacité disponible. L'algorithme s'arrête.
Le flot maximal de A vers D est :15

Exercice 3

Application de l'algorithme de Ford Fulkerson sur le graphe.



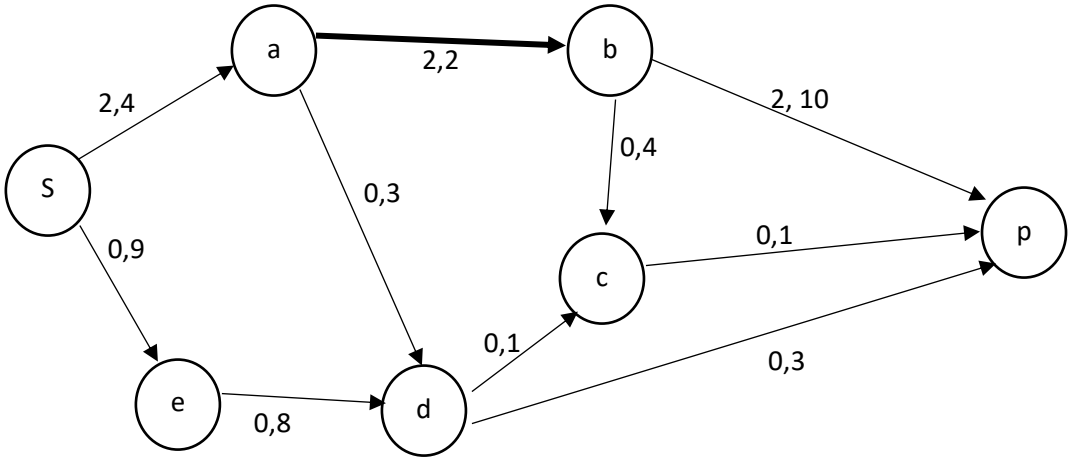
Etape 0 : On commence avec un flot initial nul (valeur = 0)



Nous allons améliorer ce flot étape par étape

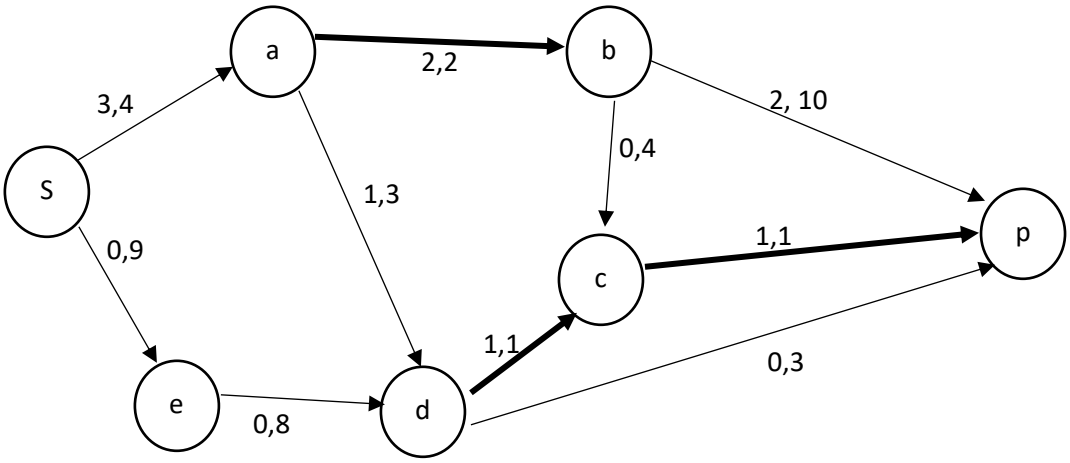
Etape 1 : On sélectionne un chemin augmentant : s,a,b,p

Le plus potentiel d'augmentation =2, on l'injecte sur le chemin, et on obtient le flot = 2 avec saturation de l'arc (a,b) :



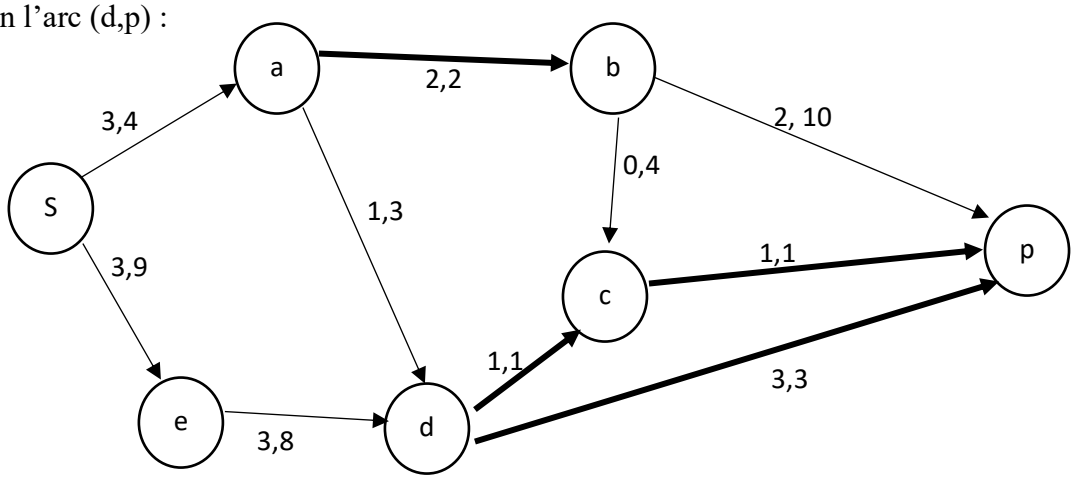
Etape 2 : On sélectionne un chemin augmentant : s,a,d,c,p

Le plus potentiel d'augmentation =1, on l'injecte sur le chemin, et on obtient le flot = 3 avec saturation des arcs (d,c) et (c,p) :



Etape 3 : On sélectionne un chemin augmentant : s,e,d,p

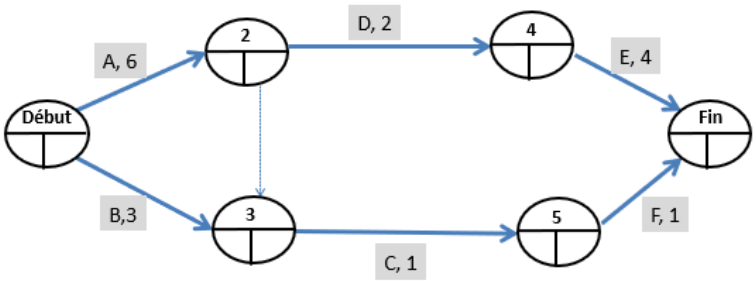
Le plus potentiel d'augmentation =3, on l'injecte sur le chemin, et on obtient le flot = 6 avec saturation l'arc (d,p) :



On ne peu plus trouver un autre chemin augmentant, donc le flot max = 6

Corrigé de série de TD n° 6

Exercice 1

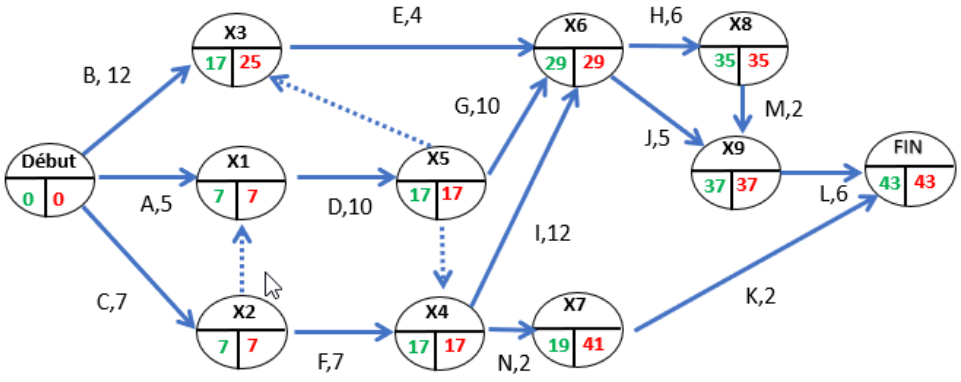


Exercice 2

Tableau des niveaux et de postériorités.

Tâches	Durée	Tâches antérieures	Niveaux	Postériorité
A	5	/	**1**	**D**
B	12	/	**1**	**E**
C	7	/	**1**	**D, F**
D	10	A, C	**2**	**E,G,I,N**
E	4	D, B	**3**	**H,J**
F	7	C	**2**	**I,N**
G	10	D	**3**	**H,J**
H	6	I, E, G	**4**	**M**
I	12	F, D	**3**	**H,J**
J	5	I, G, E	**4**	**L**
K	2	N	**4**	/
L	6	M, J	**6**	/
M	2	H	**5**	**L**
N	2	D, F	**3**	**K**

Le graphe :



Conclusion

Ce module a offert une introduction complète aux concepts et méthodes fondamentaux de la théorie des graphes, un domaine clé des mathématiques discrètes qui trouve de nombreuses applications en informatique, en recherche opérationnelle et en analyse des réseaux. Tout au long du cours, les étudiants ont exploré des notions essentielles telles que les représentations des graphes, les algorithmes de parcours (notamment la recherche en profondeur et la recherche en largeur), les algorithmes de chemin le plus court, les réseaux de flux, la connectivité et les techniques de planification de projets.

L'objectif de ce module était non seulement de développer une solide compréhension théorique des graphes et de leurs propriétés, mais aussi de doter les étudiants d'outils pratiques pour modéliser et résoudre des problèmes concrets. En étudiant des algorithmes tels que ceux de Dijkstra, Bellman-Ford et Ford-Fulkerson, les apprenants ont acquis une meilleure compréhension des processus d'optimisation et de prise de décision dans des systèmes complexes.

En outre, une attention particulière a été accordée aux méthodes de planification de projet (PERT et MPM), soulignant l'importance des chemins critiques et des contraintes de temps dans les contextes de planification et de gestion. Cela a permis aux étudiants de relier des concepts abstraits liés aux graphes à des applications pratiques en ingénierie et en gestion de projet.

À la fin de ce module, les étudiants devraient être capables d'analyser, de modéliser et de résoudre des problèmes à l'aide d'approches basées sur les graphes, tout en comprenant l'efficacité et les limites des algorithmes étudiés. Dans l'ensemble, ce cours pose des bases solides pour la poursuite d'études en algorithmes avancés

Références

Livres

1. Claude BERGE. *Graphes et hypergraphes*. 2nd édition. Dunod, 1973.
2. Reinhard Diestel. *Graph theory. Graduate texts in mathematics*, sixth edition, 2024.
3. Michel GONDRAN et Michel MINOUX. *Graphes et algorithmes*. Eyrolles, 1995.
4. Santanu Saha Ray. *Graph theory with algorithms and its applications in applied science and technology*. Springer 2013.
5. Bondy, J. A., & Murty, U. S. R. *Graph Theory*. Springer. 2008
6. West, D. B. *Introduction to Graph Theory* (2nd Edition). Prentice Hall. 2001

Ressources en ligne

1. MIT OpenCourseWare - Graph Theory and Additive Combinatorics https://ocw.mit.edu
2. Coursera - Graph Theory by University of California San Diego & National Research University https://www.coursera.org/learn/graph-theory
3. Graph Theory - Brilliant.org https://www.brilliant.org/courses/graph-theory/