Université Mustapha
Stambouli de Mascara

POLYCOPIÉ DE COURS

THÉORIE DES GRAPHES

Dr. Faiza Mahi

Ce Cours est Destiné aux
Étudiants de Deuxième Année
Licence Spécialité Systèmes
Informatiques

RÉPUBLIQUE ALGÉRIENNE DÉMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mustapha Stambouli de Mascara

Faculté des Sciences Exactes

Polycopié de Cours

Théorie des Graphes

Dr. Faiza Mahi

Avant-Propos

Ce cours est destiné aux étudiants de deuxième année Licence informatique, spécialité Systèmes Informatiques.

- **Connaissances préalables recommandées :** Notions de mathématiques, Algorithmique, et Struc- tures de Données.
- **Charge horaire :** Une séance de cours par semaine (1h30) et une séance de TD par semaine (1h30).

Ce polycopié vise à introduire aux étudiants la notion de théorie des graphes et son utilité dans la résolution de problèmes concrets de la vie courante, tels que :

- Le calcul de coût minimal,
- La recherche du meilleur chemin.
- La recherche du flot maximum.
- L'ordonnancement des tâches, etc.

Nous développons progressivement une théorie mathématique où les algorithmes classiques apparaissent comme des conséquences naturelles des concepts introduits.

Table des matières

Introduction Générale

Introduction Générale

Les graphes apparaissent dans diverses situations du monde réel, comme il existe des réseaux routiers, des réseaux d'eau et d'électricité, des réseaux informatiques et, plus récemment, des réseaux sociaux ! Si vous recherchez le temps le plus rapide pour vous rendre au travail, le moyen le moins cher de connecter un ensemble d'ordinateurs à un réseau ou un algorithme efficace pour trouver automatiquement des communautés et des leaders d'opinion sur Facebook. Le début a été hésitant : l'histoire veut que la théorie des graphes ait commencé avec l'article du mathématicien suisse Leonhard Euler sur le problème des ponts de Königsberg.

Ce cours est destiné aux étudiants de deuxième année en informatique. Son objectif est de présenter à l'étudiant d'une part la modélisation des problèmes sous forme des graphes et d'autre part ce cours contiendra un ensemble de techniques permettant à l'étudiant de résoudre ces problèmes à travers des algorithmes comme la recherche de chemin minimal, le flot maximal, le problème d'ordonnancement. Le contenu du cours est essentiellement le suivant. Nous parlerons également des algorithmes des plus courts chemins. Nous terminerons par les arbres couvrants minimaux, qui servent à planifier les réseaux routiers, téléphoniques et informatiques et trouveront également des applications dans le clustering et les algorithmes approximatifs

— **Chapitre 1 :** Décomposition des graphes apparaissent dans diverses situations du monde réel comme il existe des réseaux routiers, des réseaux informatiques et, plus récemment, des réseaux sociaux. Pour cela en presente les concepts fondamentales qui permet de modéliser des relations et des interactions entre des objets à l'aide de sommets et d'arcs , d'arêtes, Avant d'exploiter pleinement le potentiel des graphes, il est essentiel de comprendre leurs concepts fondamentaux, tels que les différents types de graphes (orientés, non orientés, pondérés), les notions de chemin, de cycle, de connexité et de degré des sommets.

— **Chapitre 2 :** Ce chapitre explore les concepts fondamentaux des arbres et des arborescences, qui sont des structures particulières de graphes jouant un rôle clé dans l'optimisation et la modélisation des réseaux. Il aborde d'abord la construction d'un arbre, qui est un graphe connexe et acyclique, puis la construction d'une forêt, qui est un ensemble d'arbres disjoints. Ensuite, deux algorithmes majeurs de recherche d'arbres couvrants minimaux sont étudiés : L'algorithme de Kruskal, qui construit un arbre couvrant minimal en ajoutant successivement les arêtes de poids minimal tout en évitant la formation de cycles. L'algorithme de Sollin (ou Borůvka), qui fonctionne par regroupement progressif des composants connexes en choisissant les arêtes de plus petit poids. Ces méthodes sont essentielles pour résoudre des problèmes d'optimisation liés aux réseaux électriques, aux télécommunications et à la conception de systèmes efficaces.

— **Chapitre 3 :** Ce chapitre présente les problèmes du plus court chemin et les principaux algorithmes utilisés pour les résoudre. Il explore différentes variantes du problème et introduit trois algorithmes classiques : L'algorithme de Dantzig : basé sur une approche

matricielle. L'algorithme de Dijkstra : méthode gloutonne efficace pour les poids positifs. L'algorithme de Ford : adapté aux graphes avec poids négatifs.

— **Chapitre 4 :** Ce chapitre présente le problème de flot maximum, Pour résoudre ce problème, nous étudierons l'algorithme de Ford-Fulkerson, nous permettra ainsi d'explorer des applications concrètes des flots dans divers domaines et de comprendre comment optimiser la gestion des réseaux complexes.

— **Chapitre 5 :** Ce chapitre présente le problème de l'ordonnancement par l'utilisation du Réseau de PERT

CHAPITRE

1

NOTIONS FONDAMENTALES DE THÉORIE DES GRAPHES

1.1 Introduction

La théorie des graphes est une branche fondamentale des mathématiques qui permet de modéliser et d'analyser des relations entre différents objets. Ces objets sont souvent représentés par des sommets (ou nœuds) reliés entre eux par des arêtes, formant ainsi un réseau de connexions. Utilisée dans divers domaines tels que l'informatique, les sciences sociales, la biologie, et la physique, la théorie des graphes offre des outils puissants pour explorer des structures complexes, qu'il s'agisse de réseaux de transport, de circuits électroniques, de réseaux sociaux ou même de connexions dans les systèmes biologiques.
En étudiant les propriétés des graphes, nous pouvons identifier des chemins optimaux, comprendre la connectivité d'un réseau, et analyser la robustesse d'un système. Ce chapitre aborde les concepts essentiels de la théorie des graphes, en commençant par les définitions de base et en explorant l'histoire du domaine. Nous verrons également des exemples pratiques et des représentations graphiques couramment utilisées.
L'histoire de la théorie des graphes débute au XVIIIe siècle avec le célèbre problème des Sept ponts de Königsberg. La ville de Königsberg (aujourd'hui Kaliningrad en Russie), traversée par la rivière Pregel, comportait sept ponts reliant différentes parties de la ville. Le défi était de trouver un chemin permettant de traverser chaque pont une seule fois et de revenir au point de départ. En 1736, le mathématicien suisse Leonhard Euler (1707-1783) prouva que ce problème n'avait pas de solution, posant ainsi les bases de la théorie des graphes. Il montra que la résolution de tels problèmes de parcours nécessitait une modélisation mathématique spécifique.
Le problème des ponts de Königsberg marque le point de départ de la théorie des graphes, qui sera enrichie par d'autres mathématiciens au cours des siècles suivants. Parmi les contributions majeures, on peut citer :
Gustav Kirchhoff (1824-1887), qui appliqua la théorie des graphes à l'analyse des circuits électriques en 1847. William Rowan Hamilton (1805-1865), qui introduisit en 1857 le concept de chemin hamiltonien, un chemin passant une seule fois par chaque sommet d'un graphe, illustré par le "jeu icosien". Depuis, la théorie des graphes a évolué et est devenue une discipline clé pour résoudre des problèmes complexes dans de nombreux domaines scientifiques et techniques.

1.2 Historique

1.2.1 Les Septs ponts de Konigsberg

La ville de Konigsberg, appelée Kaliningrad depuis 1946, est située dans une enclave russe au bord de la mer baltique (autrefois en Prusse orientale). Cette enclave est entourée à l'est et au nord par la Lituanie et au sud par Pologne (dans la carte datant de 1918). C'est en 1736 que le mathématicien suisse Léonhard EULER (1707-1783) a résolu le problème des ponts de Königsberg. On considère généralement que la théorie des graphes a eu comme point de départ la résolution de ce problème par Euler. Si la ville de Königsberg est célèbre pour ses ponts on peut aussi citer quelques hommes célèbres qui y sont nés : le philosophe Kant (1724 – 1804), le physicien Kirchhoff (1824 – 1887) et le mathématicien Hilbert (1862 – 1943). Travaux d'EULER(1735) : Comment traverser les 7 Ponts de la ville de Königsberg(Russie) une seule fois et revenir au point de départ, le Théorème d'EULER affirme sue ce problème n'admet pas de solution. Kirchhoff (1847) : analyse des circuits éléctriques. Hamilton (1857) :

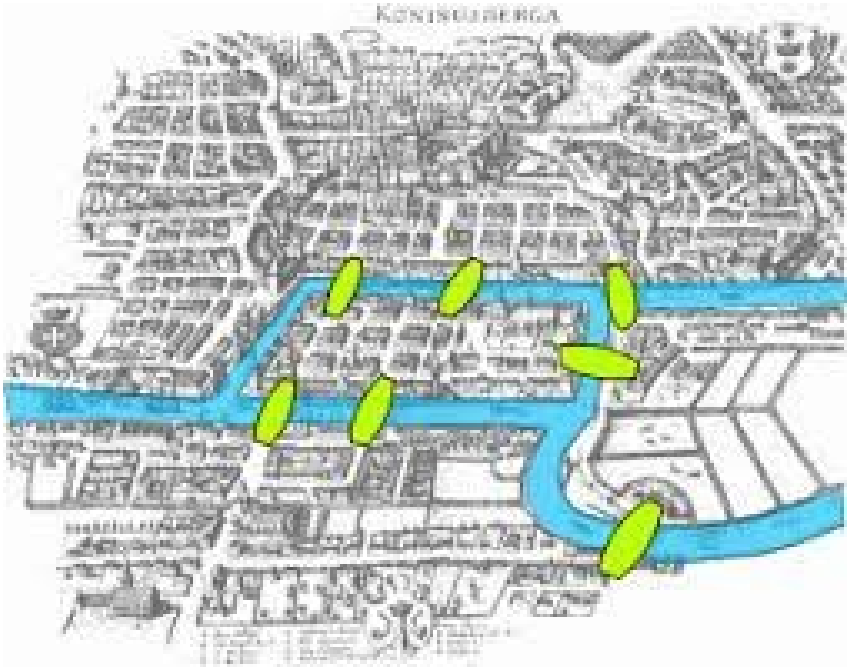


FIGURE 1.1 – Pont Euler

trouver un chemin passant une seule fois par les 18 villes du jeu icosien.

1.3 Graphe

Un graphe peut être défini comme un groupe de sommets et d'arêtes utilisés pour relier ces sommets. Un graphe peut être vu comme un arbre cyclique, où les sommets (nœuds) maintiennent une relation complexe entre eux au lieu d'avoir une relation parent-enfant. Un graphe est composé de points appelés sommets ou nœuds. Certains de ces points sont reliés entre eux par des arêtes. Ces arêtes sont souvent représentées par des segments mais elles peuvent très bien être courbes.

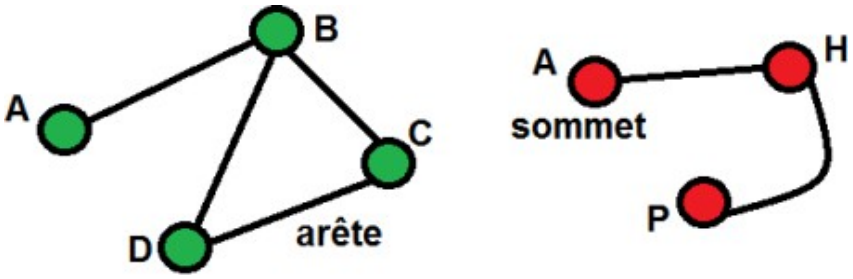


FIGURE 1.2 – Graphe

1.3.1 Graphe orienté

Un **graphe orienté** G est un couple (X,U) défini par :

— Un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ dont les éléments sont appelés **sommets** (ou **nœuds**).

— Un ensemble $U = \{u_1, u_2, \ldots, u_m\}$ dont les éléments sont appelés **arcs**. Un arc est un **couple ordonné** de sommets. Ainsi, l'ensemble U est un sous-ensemble du produit cartésien :

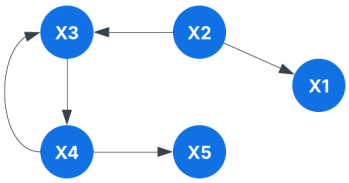
$$U \subseteq X \times X = \{(x_i, x_j) \mid x_i \in X, x_j \in X\}$$

— Un arc (x_i, x_j) a pour **extrémité initiale** le sommet x_i et pour **extrémité finale** (ou **terminale**) le sommet x_j.

— Le **nombre** n de sommets d'un graphe est appelé l'**ordre** du graphe.

— Graphiquement, les sommets d'un graphe sont représentés par des **points**, et les arcs par des **flèches** reliant les sommets dans le sens de l'ordre indiqué.

Exemple 1 :



Graphe orienté

$$X = \{x_1, x_2, x_3, x_4, x_5\}$$
$$U = \{(x_2, x_1),\ (x_2, x_3),\ (x_3, x_4),\ (x_4, x_3),\ (x_4, x_5)\}$$

Ordre du graphe : $n = 5$ sommets

1.3.2 Graphe non orienté

Un **graphe non orienté** G est un couple (X, U) défini par :

— Un ensemble $X = \{x_1, x_2, \ldots, x_n\}$ dont les éléments sont appelés **sommets** (ou **nœuds**).

— Un ensemble $U = \{u_1, u_2, \ldots, u_m\}$ dont les éléments sont appelés **arêtes**. Une arête est un **couple non ordonné** de sommets. Ainsi, l'ensemble U est un sous-ensemble de l'ensemble des parties à deux éléments de X :

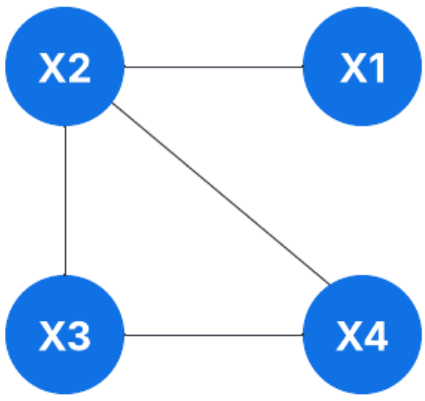
$$U \subseteq \big\{\{x_i, x_j\} \mid x_i, x_j \in X,\ x_i \neq x_j\big\}$$

— Une arête $\{x_i, x_j\}$ relie les sommets x_i et x_j, sans orientation : $\{x_i, x_j\} = \{x_j, x_i\}$.

— Le **nombre** n de sommets d'un graphe est appelé l'**ordre** du graphe.

— Graphiquement, les sommets sont représentés par des **points**, et les arêtes par des **segments** reliant les sommets (sans flèche).

Exemple 2 :



Graphe non orienté

$$X = \{x_1, x_2, x_3, x_4\}$$
$$U = \{\{x_1, x_2\},\ \{x_2, x_3\},\ \{x_3, x_4\},\ \{x_4, x_2\}\}$$

Ordre du graphe : $n = 4$ sommets

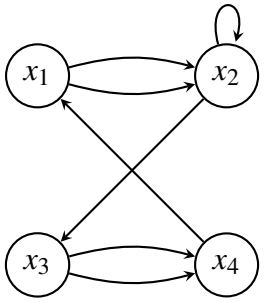
1.3.3 P-graphe

Un **p-graphe** est un graphe dans lequel il existe au maximum p arcs de la forme (x_i, x_j) entre deux sommets quelconques x_i et x_j pris dans cet ordre.

Si $p = 1$, on parle alors de **1-graphe** ou simplement **graphe**.

Exemple 3 : (2-graphe)

Considérons le graphe suivant, qui est un **2-graphe** car il y a au plus deux arcs orientés entre chaque paire de sommets donnés dans un sens donné.



Remarques :

— Il y a deux arcs de x_1 vers x_2 : cela respecte la contrainte du 2-graphe.

— De même, deux arcs de x_3 vers x_4.

— Aucun couple de sommets n'a plus de deux arcs orientés dans la même direction.

— La boucle sur x_2 est un arc de la forme (x_2, x_2).

Une Boucle est un arc (ou arête) dont l'extrémité initiale coïncide avec l'extrémité finale.

Exemple 4 :

Une boucle est notée $u = (x, x)$.

Graphe et applications multivoques

Soit $G = (X, U)$ un graphe orienté.

Dans ce contexte, on peut associer à chaque sommet $x \in X$ un ensemble de sommets liés à lui par des arcs. Ces relations définissent ce que l'on appelle des **applications multivoques**.

— L'ensemble des **successeurs** de x, noté $\Gamma^+(x)$, est défini comme :

$$\Gamma^+(x) = \{y \in X \mid (x, y) \in U\}$$

Il s'agit des sommets accessibles directement à partir de x par un arc orienté.

— L'ensemble des **prédécesseurs** de x, noté $\Gamma^-(x)$, est défini comme :

$$\Gamma^-(x) = \{y \in X \mid (y, x) \in U\}$$

Ce sont les sommets qui possèdent un arc allant vers x.

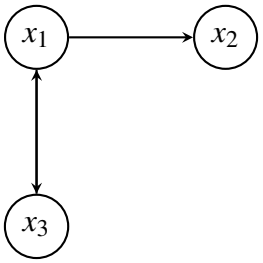
— L'ensemble des **voisins** de x est :

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

Il regroupe les successeurs et les prédécesseurs de x.

Remarque : Une **application multivoque** (ou relation multivoque) est une application qui associe à un élément x un ensemble (éventuellement vide ou contenant plusieurs éléments) au lieu d'un seul. Ainsi, l'application $\Gamma^+ : X \to \mathscr{P}(X)$ (où $\mathscr{P}(X)$ est l'ensemble des parties de X) est multivoque.

Exemple 5 : Soit le graphe suivant :



On a alors :

$$\Gamma^+(x_1) = \{x_2, x_3\}$$
$$\Gamma^-(x_1) = \{x_3\}$$
$$\Gamma(x_1) = \{x_2, x_3\}$$

1.3.4 Degré d'un sommet

Soit $G = (X, U)$ un graphe orienté.

— Le **demi-degré extérieur** d'un sommet x, noté $d^+(x)$, est le nombre d'arcs ayant x comme extrémité initiale :

$$d^+(x) = |\{u \in U \mid I(u) = x\}|$$

— Le **demi-degré intérieur** d'un sommet x, noté $d^-(x)$, est le nombre d'arcs ayant x comme extrémité terminale :

$$d^-(x) = |\{u \in U \mid T(u) = x\}|$$

— Le **degré total** d'un sommet x, noté $d(x)$, est la somme de ses deux demi-degrés :

$$d(x) = d^+(x) + d^-(x)$$

Remarque : Une boucle compte pour deux dans le calcul du degré total d'un sommet (une fois en entrée, une fois en sortie).

Propriétés :

— Dans un graphe orienté, la somme des demi-degrés extérieurs est égale à celle des demi-degrés intérieurs :

$$\sum_{x \in X} d^+(x) = \sum_{x \in X} d^-(x)$$

— Dans un graphe non orienté, la somme des degrés des sommets est un nombre pair.

— **Lemme des poignées de main :** La somme des degrés des sommets est égale au double du nombre d'arêtes m :

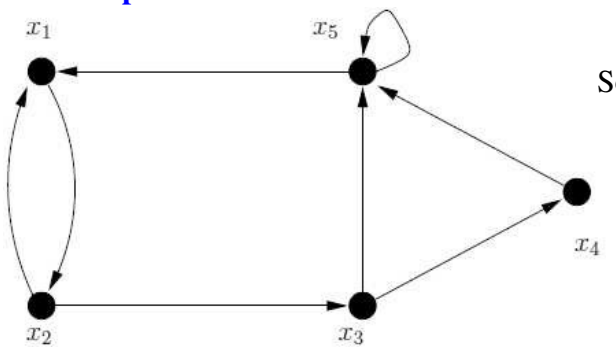
$$\sum_{x \in X} d(x) = 2m$$

Terminologie :

— Si $d(x) = 0$, le sommet x est dit **isolé**.

— Si $d(x) = 1$, le sommet x est dit **pendant**.

Exemple 6 :



Soit $X = \{x_1, x_2, x_3, x_4, x_5\}$, on a :

$$\Gamma^+(x_3) = \{x_4, x_5\} \quad \Rightarrow d^+(x_3) = 2$$
$$\Gamma^-(x_3) = \{x_2\} \quad \Rightarrow d^-(x_3) = 1$$
$$\Gamma(x_3) = \{x_2, x_4, x_5\} \quad \Rightarrow d(x_3) = 3$$

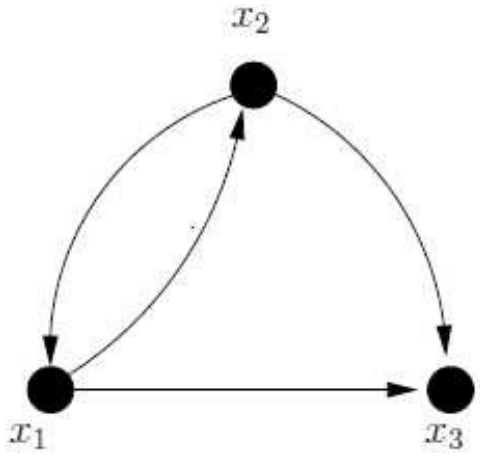
1.3.5 Types de graphes

Graphe simple

Un graphe est dit **simple** s'il ne contient ni boucles ni arêtes (ou arcs) multiples. Si G est simple, alors :

$$d(x) = |\Gamma(x)|$$

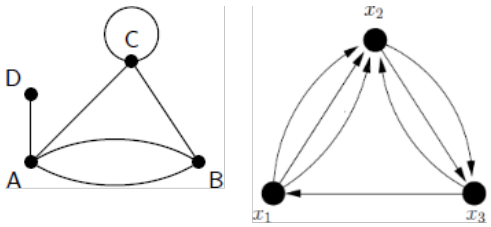
Exemple 7 :



Graphe multiple

Un graphe est dit **multiple** s'il contient au moins deux arcs (ou arêtes) parallèles entre une même paire de sommets, ou des boucles.

Exemple 8 :

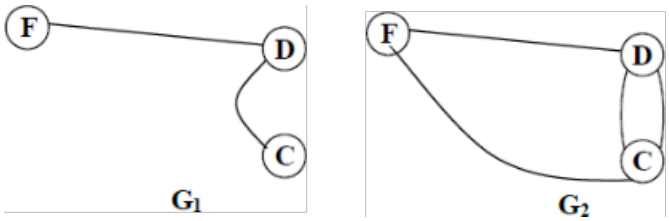


Graphe complet

Un graphe est dit **complet** si tous les sommets sont adjacents deux à deux. Le graphe complet d'ordre n (contenant n sommets) est noté K_n.

Exemple : Dans le graphe G_1, les sommets F et C ne sont pas adjacents, donc G_1 n'est pas complet. Le graphe G_2 est complet car tous ses sommets sont adjacents.

Exemple 9 :

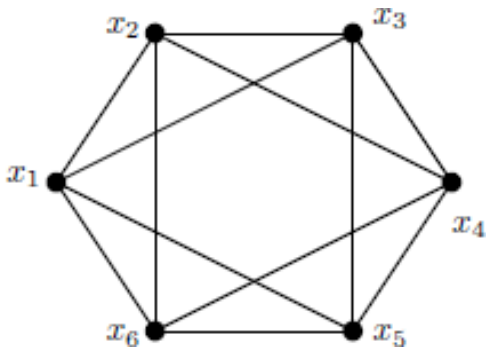


Graphe régulier

Un graphe est **k-régulier** si tous ses sommets ont un degré égal à k, c'est-à-dire :

$$\forall x \in X, \quad d(x) = k$$

Exemple 10 :



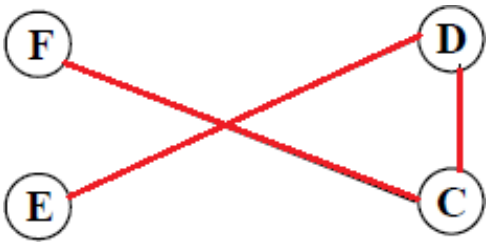
Graphe complémentaire

À un graphe simple $G = (X,U)$, on peut définir un graphe complémentaire $Gw = (X,Uw)$ comme suit : $u \notin U$.

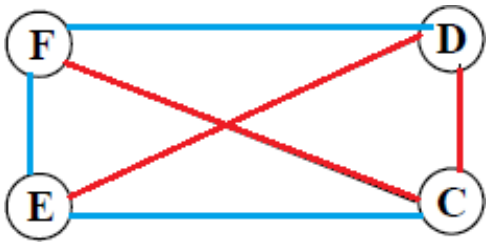
C'est-à-dire : une arête (ou un arc) appartient au graphe complémentaire Gw si elle n'appartient pas au graphe initial G.

Exemple 11 :

Conséquence :

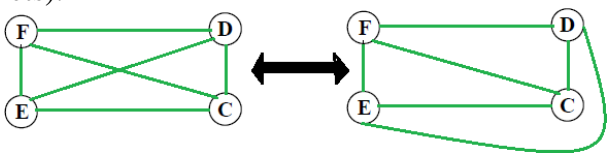


$G \cup \bar{G}$ est un graphe simple complet



Graphe planaire

Un graphe est dit **planaire** s'il peut être dessiné dans un plan sans que ses arêtes ne se croisent (sauf aux sommets).

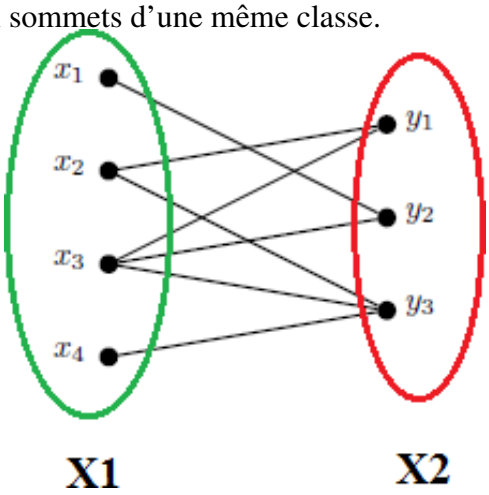


Graphe biparti

Un graphe est **biparti** si on peut partitionner son ensemble de sommets en deux classes X_1 et X_2 telles que :

$$X_1 \cap X_2 = \emptyset, \quad X_1 \cup X_2 = X$$

et aucune arête ne relie deux sommets d'une même classe.



1.3.6 Graphe partiel, Sous-graphe, Sous-graphe partiel

Soit $G = (X,U)$ un graphe orienté (resp. $G = (X,E)$ un graphe non orienté). Soit $A \subseteq X$ un sous-ensemble de sommets et $V \subseteq U$ (resp. $V \subseteq E$).

Graphe partiel Le graphe partiel de G engendré par l'ensemble d'arcs (resp. d'arêtes) V est le graphe $G_V = (X,V)$.

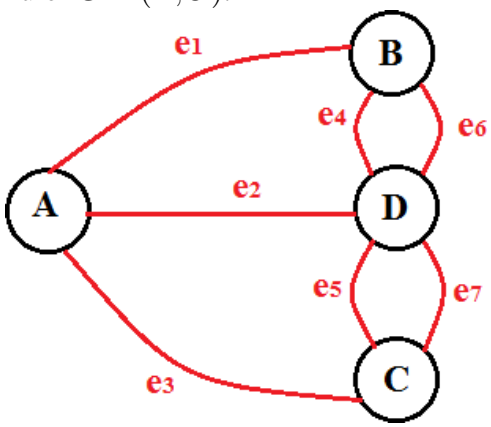
Sous-graphe Le sous-graphe de G engendré par l'ensemble de sommets A est :

— Dans le cas orienté : $G_A = (A,U_A)$ où $U_A = \{u \in U \mid \text{origine}(u) \in A \text{ et extrémité}(u) \in A\}$

— Dans le cas non orienté : $G_A = (A,E_A)$ où $E_A = \{e = (x,y) \in E \mid x \in A \text{ et } y \in A\}$

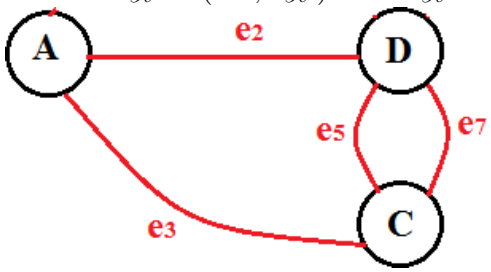
Sous-graphe partiel Un sous-graphe partiel de G engendré par l'ensemble de sommets A et l'ensemble d'arcs (resp. d'arêtes) V est le graphe $G_{A,V} = (A,V_A)$ où V_A est l'ensemble des arcs (resp. arêtes) de V ayant leurs extrémités dans A.

Exemple Soit le graphe d'Euler $G = (X,U)$.



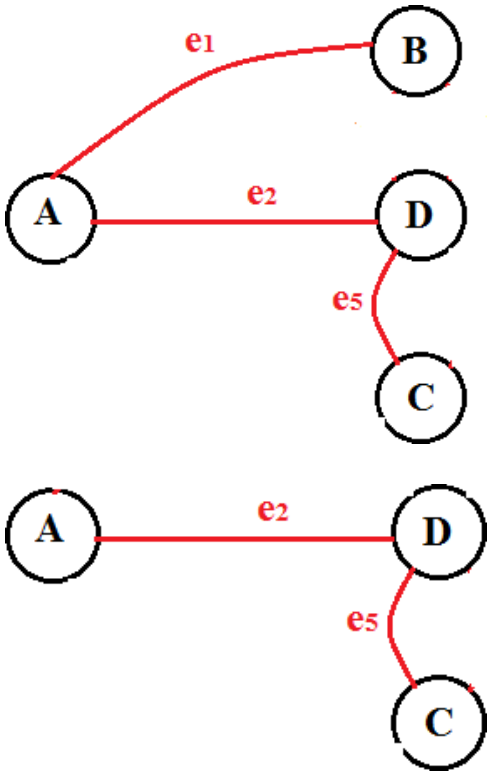
— Soient $\mathscr{H} = \{A,D,C\}$ et $\mathscr{M} = \{e_1,e_2,e_5\}$

— Le sous-graphe engendré par $\mathscr{H}$ est $G_{\mathscr{H}} = (\mathscr{H},E_{\mathscr{H}})$ avec $E_{\mathscr{H}} = \{e_2,e_3,e_5,e_7\}$



— Le graphe partiel engendré par $\mathscr{M}$ est $G_{\mathscr{M}} = (X,\mathscr{M})$

— Le sous-graphe partiel engendré par $\mathscr{H}$ et $\mathscr{M}$ est $G_{\mathscr{H},\mathscr{M}} = (\mathscr{H},V_{\mathscr{M}})$

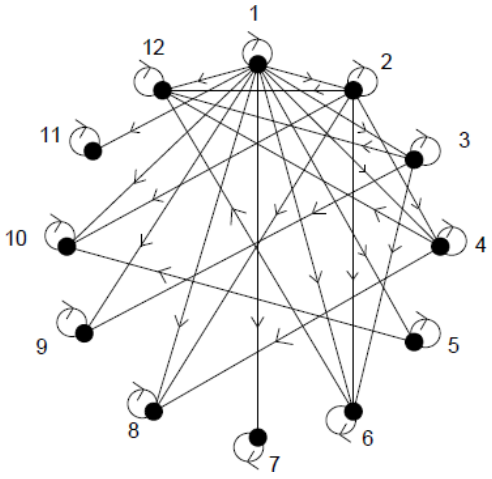


Propriétés des graphes orientés

- **Graphe réflexif** : $\forall x \in X,\ (x,x) \in U$
- **Graphe symétrique** : G est symétrique ssi $\forall x,y \in X,\ (x,y) \in U \Rightarrow (y,x) \in U$
- **Graphe antisymétrique** : G est antisymétrique ssi $\forall x,y \in X,\ (x,y) \in U \Rightarrow (y,x) \notin U$
- **Graphe transitif** : G est transitif ssi $\forall x,y,z \in X,\ (x,y) \in U \wedge (y,z) \in U \Rightarrow (x,z) \in U$

Modélisation

Exemple 1 Construire un graphe orienté dont les sommets sont les entiers de 1 à 12 et dont les arcs représentent la relation ”être diviseur de”.



Exemple 2 Trois professeurs $P1, P2, P3$ devront donner lundi prochain un certain nombre d'heures de cours à trois classes $C1, C2, C3$:

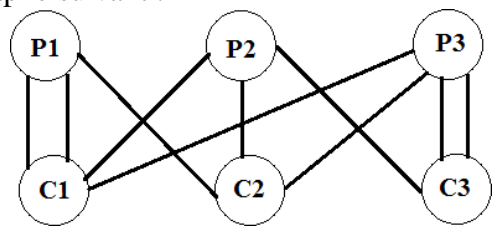
— $P1$doit donner 2 heures de cours à $C1$ et 1 heure à $C2$

— $P2$ doit donner 1 heure de cours à $C1$, 1 heure à $C2$ et 1 heure à $C3$

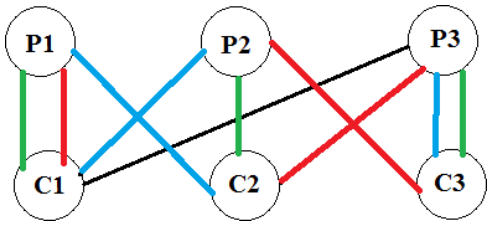
— $P3$doit donner 1 heure de cours à $C1$, 1 heure à $C2$ et 2 heures à $C3$

Comment représenter cette situation par un graphe ?
Combien faudra-t-il de plages horaires au minimum ? Aidez-vous du graphe pour proposer un horaire du lundi pour ces professeurs.

Solution On obtient le graphe suivant :



En colorant les arêtes de ce graphe (1 couleur = 1 heure de l'horaire), en prenant garde que chaque sommet n'ait pas deux arêtes incidentes de même couleur.



On obtient le résultat de droite. De ce graphe coloré, on tire l'horaire suivant :

	P1	P2	P3
1ère heure	C1	C3	C2
2ème heure	C1	C2	C3
3ème heure	C2	C1	C3
4ème heure			C1

1.4 Algorithme de K-coloration

Une **K-coloration** d'un graphe G (simple, sans boucles) est une application qui associe à chaque sommet de G une couleur de l'ensemble $\{1, \ldots, K\}$ de telle façon que les sommets voisins aient des couleurs distinctes.

De façon équivalente, une K-coloration de G correspond à une partition de X en K ensembles stables (indépendants) :

$$X = X_1 \cup X_2 \cup \cdots \cup X_K$$

où X_i est l'ensemble des sommets colorés en i.

Un graphe G est **K-coloriable** s'il peut être colorié avec K couleurs.

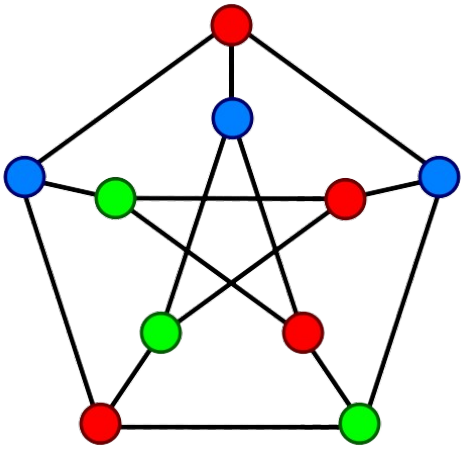
Le **nombre chromatique** d'un graphe G, noté $\chi(G) = K$, est le nombre minimum de couleurs nécessaires pour colorier G.

Un graphe G est **K-chromatique** si et seulement si $\chi(G) = K$.

Algorithme de Welsh et Powell :

1. Ordonner les sommets par degrés décroissants.
2. Traiter les sommets dans cet ordre, en affectant à chaque sommet X_i une couleur différente de celles de ses voisins déjà coloriés.

Exemple



Représentation matricielle

A un graphe $G = (X,U)$ avec n sommets et m arcs, on associe trois types de matrices : [...à compléter]

1.4.1 La matrice d'adjacence

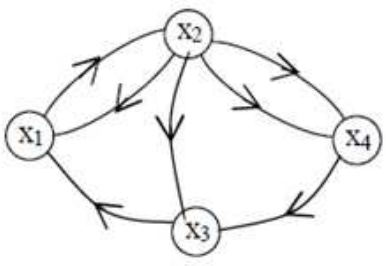
La matrice d'adjacence du graphe $G = (X,U)$ est une matrice $n \times n$ dont les éléments prennent deux valeurs : 1 ou 0. Chaque ligne et chaque colonne correspondent à un sommet du graphe. Ainsi, chaque élément de la matrice indique la relation qui existe entre deux sommets :

— 1 signifie que les deux sommets sont reliés par un arc orienté ;

— 0 signifie que les deux sommets ne sont pas reliés par un arc.

Exemple :

La matrice d'adjacence de G est la suivante :



	X_1	X_2	X_3	X_4
X_1	0	1	0	0
X_2	1	0	1	1
X_3	1	0	0	0
X_4	0	0	1	0

Notation : Les éléments de la matrice d'adjacence sont définis par :

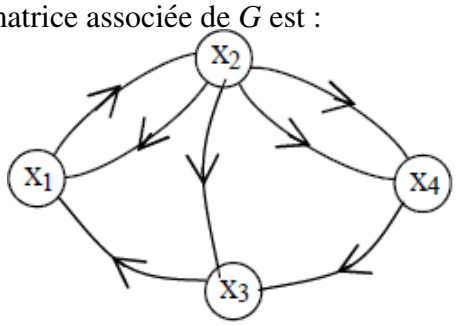
$$a_{ij} = \begin{cases} 1 & \text{si l'on a un arc orienté de } x_i \text{ vers } x_j \\ 0 & \text{sinon} \end{cases}$$

1.4.2 La matrice associée

La matrice associée du graphe $G = (X, U)$ est une matrice $n \times n$, où chaque ligne et chaque colonne correspondent à un sommet du graphe. Les éléments indiquent le nombre d'arcs orientés dans le même sens reliant deux sommets.

Exemple :

La matrice associée de G est :



	X1	X2	X3	X4
X1	0	1	0	0
X2	1	0	1	2
X3	1	0	0	0
X4	0	0	1	0

Notation :

$$a_{ij} = k \quad \text{si l'on a } k \text{ arcs orientés de la forme } (x_i, x_j)$$

Remarque :

— La somme des valeurs d'une ligne donne le **demi-degré extérieur** du sommet.

— La somme des valeurs d'une colonne donne le **demi-degré intérieur** du sommet.

1.4.3 La matrice d'incidence aux arcs

La matrice d'incidence aux arcs d'un graphe $G = (X, U)$ est une matrice $n \times m$, ses éléments prennent les valeur 1,0 ou -1. Chaque ligne de la matrice est associée à un sommet et chaque colonne à un arc. Chaque élément de la matrice indique la relation entre un sommet et un arc comme suit :

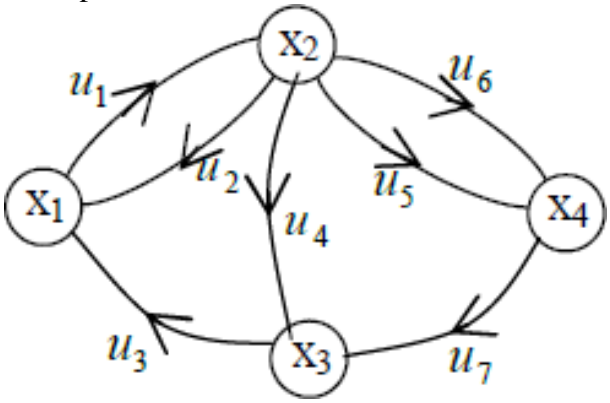
— +1 signifie que le sommet est une extrémité initiale de l'arc.

— -1 signifie que le sommet est une extrémité terminale de l'arc.

— 0 signifie qu'il n'existe pas de relations entre le sommet et l'arc.

Exemple :

Soit un graphe d'ordre 4, composé de 7 arcs :



La matrice d'incidence aux arcs de G est :

	u_1	u_2	u_3	u_4	u_5	u_6	u_7
x_1	+1	-1	-1	0	0	0	0
x_2	-1	+1	0	+1	+1	+1	0
x_3	0	0	+1	-1	0	0	-1
x_4	0	0	0	0	-1	-1	+1

Cette matrice ne convient pas pour les graphes avec boucles.

Notation :

$$a_{ij} = \begin{cases} +1 & \text{si } x_i \text{ est l'extrémité initiale de l'arc } u_j \\ -1 & \text{si } x_i \text{ est l'extrémité terminale de l'arc } u_j \\ 0 & \text{dans les autres cas} \end{cases}$$

Remarque :

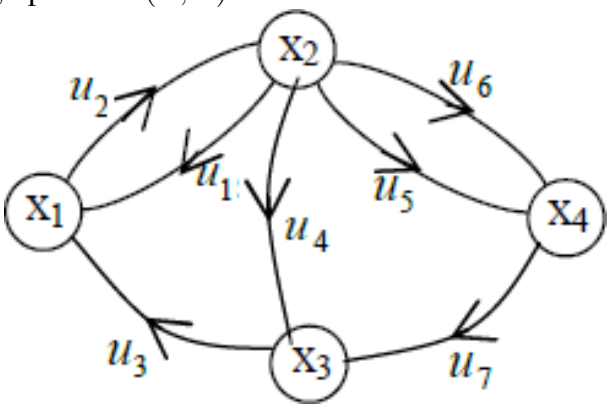
Dans la matrice d'incidence, on a :

— Le nombre de valeurs égales à $+1$ sur une ligne donne le **degré extérieur** du sommet correspondant.

— Le nombre de valeurs égales à -1 sur une ligne donne le **degré intérieur** du sommet correspondant.

1.4.4 Dictionnaire des successeurs et des prédécesseurs

Exemple : Soit le graphe $G = (X, U)$ suivant :



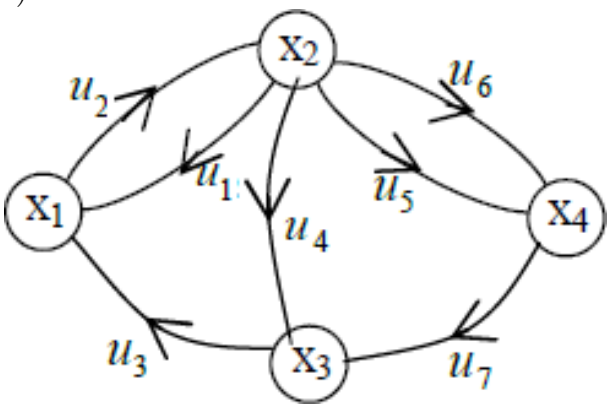
Le dictionnaire des successeurs et des prédécesseurs de G est le suivant :

Sommet	Dictionnaire des successeurs $\Gamma^+(x_i)$	Dictionnaire des prédécesseurs $\Gamma^-(x_i)$
x_1	x_2	x_2, x_3
x_2	x_1, x_3, x_4	x_1
x_3	x_1	x_2, x_4
x_4	x_3	x_2

1.4.5 Représentation sagittale

On examine tous les arcs du graphe G, puis on détermine pour chaque arc son **extrémité initiale** et son **extrémité terminale**.

Exemple :
Soit le graphe $G = (X, U)$ suivant :



La représentation sagittale du graphe G est la suivante :

Arc U_i	U_1	U_2	U_3	U_4	U_5	U_6	U_7
$I(U_i)$	x_2	x_1	x_3	x_2	x_2	x_2	x_4
$T(U_i)$	x_1	x_2	x_1	x_3	x_4	x_4	x_3

1.5 La connexité dans un graphe

1.5.1 Le graphe fortement connexe

Un graphe G est dit **fortement connexe** si tous ses sommets ont deux à deux la relation de forte connexité, autrement dit si G contient une seule composante fortement connexe.

Exemple :
Le graphe G précédent contient deux composantes fortement connexes :

$$C_1 = \{x_1, x_2, x_3\}, \quad C_2 = \{x_4, x_5\}$$

Donc le graphe n'est **pas fortement connexe**.

Si on ajoute l'arc (x_5, x_3) au graphe précédent, le graphe obtenu contient une seule composante fortement connexe :

$$C = \{x_1, x_2, x_3, x_4, x_5\}$$

Il est alors **fortement connexe**.

1.5.2 La recherche des composantes fortement connexes

Soit G un graphe orienté. Un moyen de vérifier si ce graphe est fortement connexe est d'appliquer l'**algorithme de marquage** suivant, qui permet de déterminer toutes ses composantes fortement connexes.

Principe : L'idée de cet algorithme est de parcourir le graphe à partir d'un sommet x dans le sens direct (en suivant les flèches des arcs) et dans le sens opposé (en suivant les flèches en sens inverse).

Le premier ensemble obtenu regroupe les sommets **accessibles à partir de** x, le deuxième regroupe les sommets **qui peuvent atteindre** x. L'intersection de ces deux ensembles forme la composante fortement connexe contenant x.

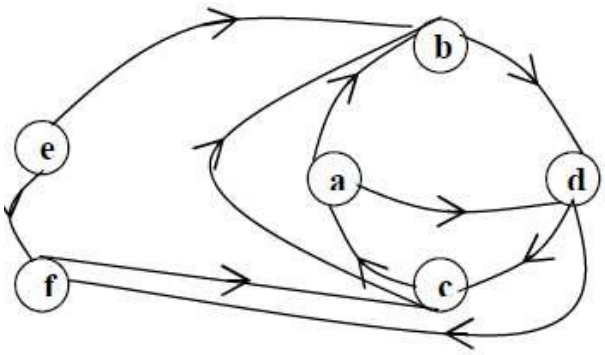
Algorithme :

— **Données :** un graphe orienté $G = (X, U)$.

— **Résultat :** le nombre k de composantes fortement connexes de G et la liste $\{C_1, C_2, \ldots, C_k\}$.

0. Initialisation : $k = 0$, $W = X$.
1. Tant que $W \neq \emptyset$, faire :
   (a) Choisir un sommet $x \in W$ et le marquer par $(+)$ et $(-)$.
   (b) Marquer tous les successeurs directs et indirects de x avec $(+)$.
   (c) Marquer tous les prédécesseurs directs et indirects de x avec $(-)$.
   (d) Poser $k = k + 1$, $C_k =$ sommets marqués avec $(+)$ et $(-)$.
   (e) $W = W \setminus C_k$, effacer toutes les marques.
2. Le nombre de composantes fortement connexes de G est k.

Application : Soit le graphe $G = (X, U)$ suivant :



Du graphe G, on constate qu'il y a un chemin reliant le sommet e à b, mais on n' a pas de chemin reliant le sommet b à e, alors e et b n'ont pas une relation de forte connexité, donc il existe plus d'une composante fortement connexe, on conclut que le graphe n'est pas fortement connexe, On applique l'algorithme de marquage précédent pour déterminer les composantes fortement connexes :

Initialisation : $k = 0$, $W = X = \{a, b, c, d, e, f\}$

Itération 1 :

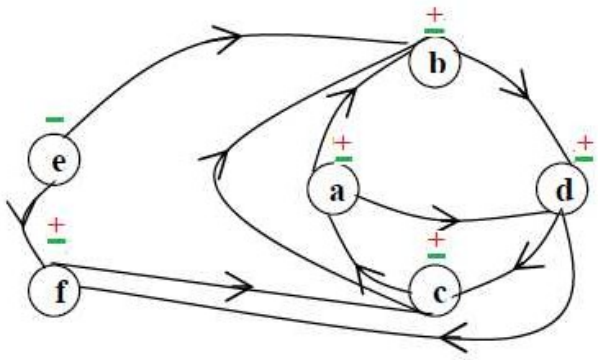
On choisit dans W, le sommet $\{a\}$ et on le marque par $(+)$ et $(-)$. On marque ensuite :

— Suivant l'orientation des arcs, les sommets b et d (successeurs directs du sommet a) avec le signe $(+)$, ensuite les sommets c et f (successeurs indirects du sommet a) avec le signe $(+)$.

— Suivant l'orientation des arcs en sens inverse, le sommet c (prédécesseur direct du sommet a) avec le signe $(-)$, ensuite les sommets b, d, e, f (prédécesseurs indirects du sommet a) avec le signe $(-)$.

Soit $C_1 = \{a, b, c, d, f\}$ l'ensemble des sommets marqués à la fois par $(+)$ et $(-)$. On retire de W les sommets de C_1, on obtient : $W = \{e\}$, $G = \varnothing$.

Itération 2 :



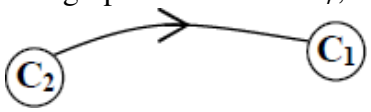
On marque le sommet e d'un signe $(+)$ et $(-)$, on constate qu'il n'y a pas d'autres sommets à marquer. Alors $C_2 = \{e\}$, l'ensemble des sommets marqués à la fois par $(+)$ et $(-)$.

On retire de W les sommets de C_2, on obtient : $W = G = \varnothing$, terminé.

Le nombre de composantes fortement connexes de G est 2, elles sont :

$$C_1 = \{a, b, c, d, f\} \quad \text{et} \quad C_2 = \{e\}$$

Au graphe G, on fait correspondre un graphe réduit noté G_r, représenté comme suit :



1.6 Mise en ordre d'un graphe connexe et recherche de circuit

1.6.1 Mise en ordre :

Ordonner un graphe signifie classer les sommets par niveaux, en respectant le sens des arcs.

Procédure :

— **Données :** un graphe $G = (X, U)$.

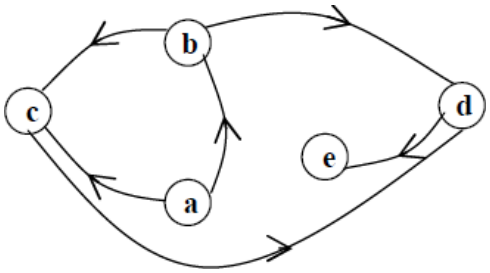
— **Résultat :** les différents niveaux de sommets du graphe ainsi que le graphe ordonné de G.

0. On détermine le dictionnaire des prédécesseurs du graphe $\Gamma^-(x)$.
1. On repère dans le dictionnaire des prédécesseurs du graphe les sommets n'ayant pas de prédécesseurs $\Gamma^-(x) = \varnothing$.
   - 1.1 On pose N_0 l'ensemble des sommets du graphe n'ayant pas de prédécesseurs, on l'appelle niveau nul.
   - 1.2 On barre dans la colonne de $\Gamma^-(x)$ tous les sommets de niveau nul N_0, on obtient une nouvelle colonne $\Gamma^-(x)$, avec G_1 le sous-graphe engendré par $X \setminus N_0$.
2. On repère dans la nouvelle colonne $\Gamma^-(x)$ les sommets n'ayant pas de prédécesseurs $\Gamma^-(x) = \varnothing$.
   - 2.1 On pose N_1 l'ensemble des sommets du graphe n'ayant pas de prédécesseurs.

2.2 On barre dans la colonne de $\Gamma^-(x)$ tous les sommets de niveau N_1, on obtient une nouvelle colonne $\Gamma^-(x)$, avec G_2 le sous-graphe engendré par $X \setminus (N_0 \cup N_1)$.

3. On continue le même procédé jusqu'à ce qu'on termine le graphe et on représente ainsi le graphe ordonné par niveaux de G.

Exemple : Soit le graphe G=(X, U) représentant le processus de transformation d'une matière première a dans un atelier.



1. Soit le dictionnaire des prédécesseurs du graphe G :

X	$\Gamma_G^-(x)$
a	Φ
b	~~a~~
c	~~a~~-b
d	c-b
e	d

Le sommet a n'a pas de prédécesseurs, il est donc de niveau nul $N_0 = \{a\}$.
On barre le sommet a de la colonne $\Gamma^-(x)$, on obtient la nouvelle colonne $\Gamma^-(x)$.
Le sommet b n'a plus de prédécesseurs, il est donc de niveau un $N_1 = \{b\}$.

X	$\Gamma_{G1}^-(x)$
a	-
b	Φ
c	~~b~~
d	c-~~b~~
e	d

On barre le sommet b de la colonne $\Gamma^-(x)$, on obtient la nouvelle colonne $\Gamma^-(x)$.
G_1 et G_2 sont les sous-graphes obtenus à chaque étape.

2. Le sommet c n'a plus de prédécesseurs, il est donc de niveau deux $N_2 = \{c\}$.

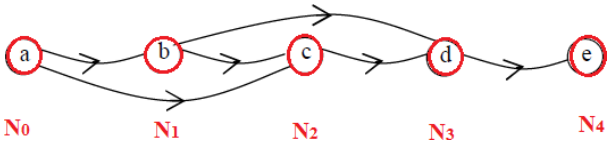
X	$\Gamma^-_{G2}(x)$
a	-
b	-
c	Φ
d	~~c~~
e	d

On barre le sommet c de la colonne $\Gamma^-(x)$, on obtient le nouveau dictionnaire.

3. Le sommet d n'a plus de prédécesseurs, il est donc de niveau trois $N_3 = \{d\}$.

X	$\Gamma^-_{G3}(x)$
a	-
b	-
c	-
d	Φ
e	~~d~~

On a examiné tous les sommets du graphe G, on a ainsi le graphe ordonné de G représenté ci-dessous.



1.6.2 Recherche de circuit :

À une étape donnée de l'ordonnancement d'un graphe, la définition des niveaux se bloque (i.e. il n'existe pas de sommets n'ayant pas de prédécesseurs), donc la mise en ordre du graphe est impossible. On dit alors que le graphe possède un circuit.

Comment faire pour déterminer ce circuit ?

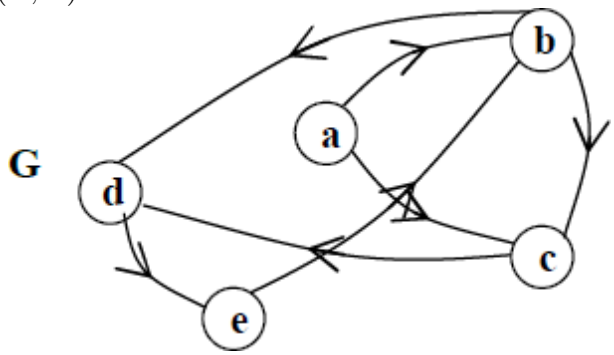
On applique la procédure suivante :

Soit $(X, \Gamma^-(x))$ le dernier dictionnaire des prédécesseurs du graphe G, où l'application de l'algorithme de mise en ordre est bloquée.

1. On repère dans la colonne $\Gamma^-(x)$ un sommet quelconque x_0. Ce sommet nous renvoie à sa position dans la colonne X.
2. On lit à nouveau dans la colonne $\Gamma^-(x)$ un nouveau sommet x_1 correspondant au prédécesseur de x_0, lequel nous renvoie à sa position dans la colonne X, et ainsi de suite.
3. Cette procédure s'arrête lorsque, dans la suite des sommets lus, un sommet se répète. Le circuit est alors compris entre ces deux occurrences du même sommet.

Remarque :
L'écriture de la suite des sommets se fait de gauche à droite, c'est-à-dire $(x_0, x_1, \ldots, x_i, \ldots, x_p, \ldots, x_i)$, et la lecture du circuit se fait de droite à gauche.

Application :
Soit le graphe $G = (X, U)$ suivant :



1. Soit le dictionnaire des prédécesseurs du graphe G :
   Le sommet a n'a pas de prédécesseurs, il est donc de niveau nul $N_0 = \{a\}$.

X	$\Gamma_G^-(x)$
a	Φ
b	~~a~~-e
c	~~a~~-b
d	c-b
e	d

   On barre le sommet a de la colonne $\Gamma^-(x)$, on obtient le dictionnaire suivant :

X	$\Gamma^-_{G1}(x)$
a	-
b	e
c	b
d	c-b
e	d

2. À cette étape, on ne peut pas déterminer le niveau N_1 car il n'y a pas de sommets n'ayant pas de prédécesseurs. Donc, le graphe n'est pas ordonnable ; G contient alors un circuit.
   Pour déterminer ce circuit :
   — On choisit un sommet dans $\Gamma^-(x)$, soit e. Ce sommet nous renvoie à sa ligne dans X.

X	$\Gamma^-_{G1}(x)$
a	-
b	e
c	b
d	c-b
e	d

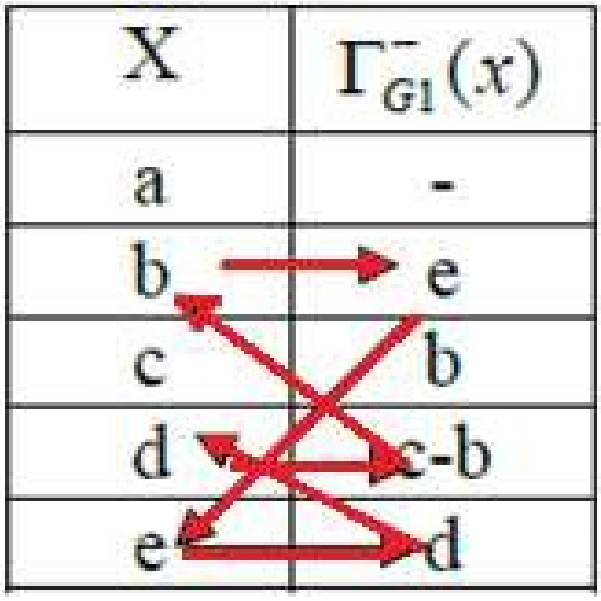
   — On choisit un sommet dans la ligne de e, soit d, celui-ci nous renvoie à sa ligne dans X.

X	$\Gamma^{-}_{G1}(x)$
a	-
b	e
c	b
d	b-c
e	d

— On choisit un sommet dans la ligne de d, soit b, ce sommet nous renvoie à sa ligne dans X.

X	$\Gamma^{-}_{G1}(x)$
a	-
b	e
c	b
d	b-c
e	d

— On choisit un sommet dans la ligne de b, soit e. On arrête la procédure car le sommet e s'est répété.

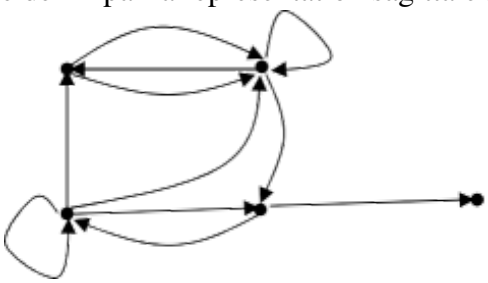


La suite des sommets trouvée est : $e \to d \to b \to e$.
La lecture du circuit se fait de droite à gauche, on obtient ainsi le circuit : **(e b d e)**.

1.7 Exercices

1.7.1 Exercice n°1

Soit $G = (X,U)$ le graphe défini par la représentation sagittale suivante :



a) Quel est l'ordre et le p-graphe de ce graphe ?
Donner deux boucles.
Donner deux arcs adjacents et deux sommets adjacents.

b) Peut-on avoir un 1-graphe G_1 de ce graphe ? Comment ?
Donner la représentation sagittale du 1-graphe G_1.
Soit $G_1 = (X_1,U_1)$, donner les éléments des ensembles X_1 et U_1.

1.7.2 Exercice n°2

Soit un graphe $G = (X,U)$ défini par :

$$X = \{x_1,x_2,x_3,x_4,x_5,x_6\}$$

$$U = \{(x_1,x_1),(x_1,x_2),(x_2,x_3),(x_2,x_4),(x_2,x_6),(x_4,x_4),(x_4,x_1),(x_4,x_2),(x_5,x_1),(x_5,x_6),(x_6,x_1)\}$$

1. Donner la représentation sagittale de ce graphe.
2. Pour chaque sommet x_i, donner $\Gamma^+(x_i)$ et $\Gamma^-(x_i)$.
3. Pour chaque sommet x_i, donner le degré sous la forme :

$$d_G(x_i) = d^+(x_i) + d^-(x_i)$$

4. Est-ce que ce graphe est un graphe régulier ? Justifiez votre réponse.
5. Soit $A \subset X$ défini par :

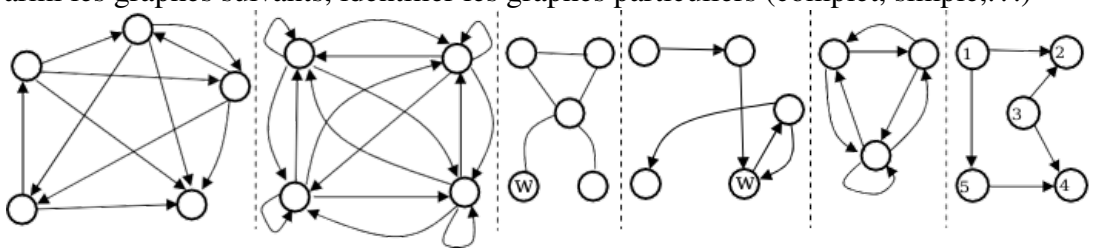
$$A = \{x_1, x_2, x_3, x_4\}$$

Donner l'ensemble des arcs incidents à A sous la forme :

$$W(A) = W^+(A) \cup W^-(A)$$

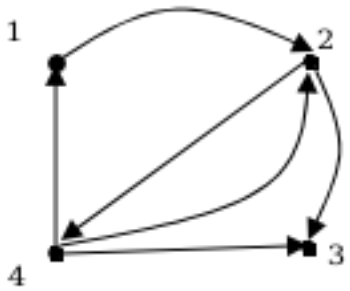
1.7.3 Exercice n°3

Parmi les graphes suivants, identifier les graphes particuliers (complet, simple,...)



1.7.4 Exercice n°4

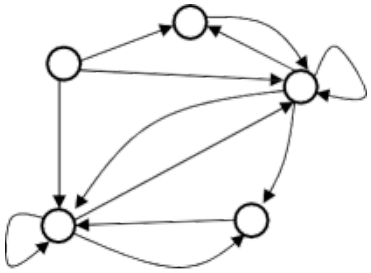
Soit un graphe $G = (X, U)$ défini par :



1. Donner un **sous-graphe** G_s du graphe G.
2. Donner un **graphe partiel** G_p du graphe G.
3. Donner un **sous-graphe partiel** G_{sp} du graphe G.
4. Donner le **graphe complémentaire** $Gw = (X, Uw)$ du graphe G.

1.7.5 Exercice n°5

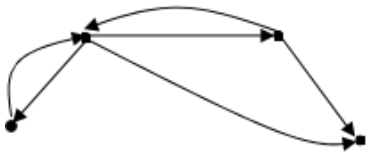
Soit (X,U) le graphe suivant :



1. Préciser le type des séquences suivantes (chemin, chaîne, cycle, circuit, etc.) :
   (a) (a,b,c,d,e)
   (b) (a,c,d,e,c,b)
   (c) (a,c,d,e,c,d,e)
   (d) (c,d,e,c)
   (e) (b,c,d,e,c,b)
   (f) (a,b,c,e)
   (g) (c,d,e)
2. Dégager :
   — Un **chemin hamiltonien**, s'il en existe un.
   — Un **circuit hamiltonien**, s'il en existe un.

1.7.6 Exercice n°6

Soit (X,U) le graphe défini par :



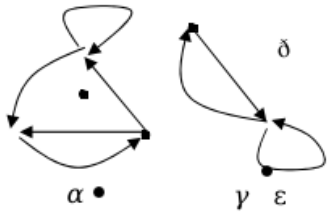
Donner sa représentation :

— Par correspondance.

— En casier.

1.7.7 Exercice n°7

Soit (X,U) le graphe défini par :



Donner :

— Sa matrice booléenne (incidence sommet–sommet).
— Sa matrice d'incidence sommet–arc.
— Sa matrice latine.

1.7.8 Exercice n°8

Soit un graphe (X,U) défini par :

$$X = \{1,2,3,4,5,6,7,8\}, \quad U = \{(x,y) \mid x \text{ divise } y\}$$

a. Donner la représentation sagittale de G.
b. Comment reconnaît-on sur G quand un nombre $x \in X$ est premier ?
c. Comment reconnaît-on sur G quand deux nombres x et $y \in X$ sont premiers entre eux ?
d. Comment peut-on voir sur G le nombre de diviseurs d'un nombre $x \in X$?

CHAPITRE

2

ARBRES ET ARBORESCENCES

2.1 Introduction

Les arbres et les arborescences sont des structures fondamentales utilisées dans de très nombreux domaines : informatique, science sociale, statistique, intelligence artificielle,...

2.2 Concepts de Base

Nombre Cyclomatique et Nombre Cocyclomatique

Étant donné un graphe G avec n sommets, m arcs et p composantes connexes :

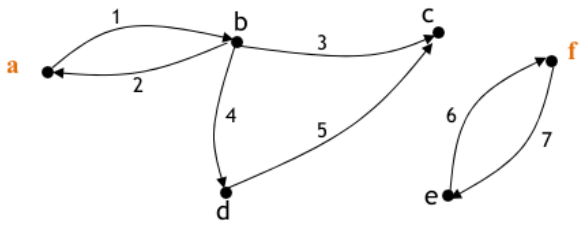
— La dimension de la base des cycles (cycles indépendants) est $\mu(G) = m - n + p$; $\mu(G)$ est appelé le "nombre cyclomatique" du graphe.

— La dimension de la base des cocycles (cocycles indépendants) est $\nu(G) = n - p$; $\nu(G)$ est appelé le "nombre cocyclomatique" du graphe.

Du fait que le nombre de chemins, cycles et cocycles existants dans un graphe peut être très grand, les nombres cyclomatique et cocyclomatique permettent de mesurer le nombre de cycles et cocycles indépendants dans un graphe donné.

Exemple 1 :

$n = 6$, $m = 7$, $p = 2$



$\mu(G) = m - n + p = 3$ donc il existe 3 cycles élémentaires indépendants.
$\nu(G) = n - p = 4$ donc il existe 4 cocycles élémentaires indépendants.

2.3 Arbres

2.3.1 Définitions

Les **arbres** sont des graphes particuliers, très utilisés en algorithmique et en informatique.

Soit $G = (X, U)$ un graphe d'ordre $n = |X|$ et de taille $m = |U|$.

Le graphe G est un **arbre** s'il est :

— **connexe** : il existe un chemin entre toute paire de sommets,

— **acyclique** : il ne contient aucun cycle.

On appelle **feuille** d'un arbre tout sommet qui est **adjacent à une seule arête** (c'est-à-dire de degré 1).

Exemple :

FIGURE 2.1 – Exemples d'arbres

2.3.2 Propriétés Fondamentales d'un Arbre

Un arbre vérifie les propriétés suivantes :

1. Le nombre d'arêtes (ou arcs) est :
$$m = n - 1$$
2. Si l'on supprime une arête, le graphe devient **déconnecté** (il ne reste plus connexe).
3. Si l'on ajoute une arête, le graphe devient **cyclique** (il contient alors un cycle).

2.3.3 Caractéristiques Équivalentes d'un Arbre

Pour un graphe H d'ordre $n \geq 2$, les propriétés suivantes sont équivalentes et caractérisent un arbre :

1. H est connexe et sans cycle.
2. H est sans cycle et possède $n-1$ arêtes.
3. H est connexe et possède $n-1$ arêtes.
4. L'ajout d'une arête à H crée un cycle unique.
5. La suppression d'une arête rend le graphe non connexe.
6. Il existe un seul chemin entre toute paire de sommets.

2.3.4 Forêt

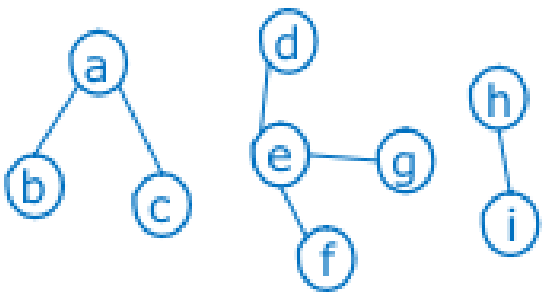
Une **forêt** est un graphe dont chaque composant connexe est un *arbre*. Autrement dit, une forêt est un graphe **acyclique**.

Sur un ordinateur, une forêt peut être représentée à l'aide de deux tableaux :

— **NA** : le nombre d'arêtes de chaque arbre de la forêt,

— **AR** : les arêtes des arbres.

Exemple : Soit la foret suivante



On peut la sauvegarder sur ordinateur sous forme de :

NA :

2	3	1

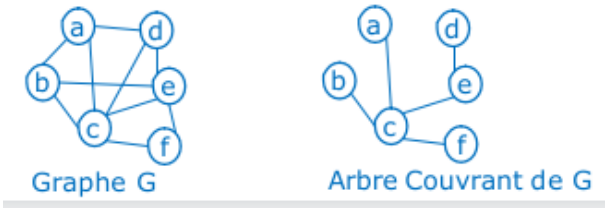
AR :

a	a	e	e	e	h
b	c	d	f	g	l

2.3.5 Arbre Couvrant

— Un **arbre** $T = (X_T, U_T)$ est un **arbre couvrant** du graphe $G = (X, U)$ si T est un graphe partiel de G, c'est-à-dire :

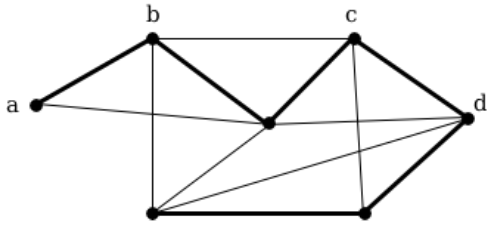
$$X_T = X \quad \text{et} \quad U_T \subset U.$$



2.3.6 Algorithme de détermination d'un arbre partiel d'un graphe connexe

1. Chercher une arête (ou un arc) dont la suppression **ne déconnecte pas** le graphe.
2. Si une telle arête existe, la supprimer du graphe, puis reprendre à l'étape 1.
3. Sinon, le graphe est un arbre partiel. La procédure s'arrête lorsqu'il n'y a plus d'arêtes à supprimer.

Exemple



Les arêtes en gras forment l'arbre partiel, les autres ont été éliminées.

2.4 Algorithmes de Recherche d'un Arbre Partiel de Valeur Minimale

Soit un graphe $G = (X, U)$, où X est l'ensemble des sommets et U celui des arêtes (ou arcs). À chaque arête $u \in U$ est associé un coût $c(u)$, appelé aussi *valeur* ou *poids*. La **valeur totale** d'un arbre partiel $H = (X, V)$ est donnée par :

$$S(H) = \sum_{u \in V} c(u)$$

Le problème consiste à déterminer un arbre partiel H de G tel que $S(H)$ soit minimal, c'est-à-dire que la somme des coûts des arêtes dans H soit la plus petite possible. Ce minimum est pris parmi tous les arbres couvrants possibles du graphe G. Si tous les coûts sont distincts, cet arbre minimal est unique.

Ce problème a de nombreuses applications, par exemple dans la construction de réseaux (électriques, routiers, etc.). Considérons un ensemble de n sites devant être reliés par des lignes à haute tension. L'objectif est de construire un réseau les connectant tous avec une longueur totale minimale. Ce problème peut être modélisé par un graphe complet et valué $G = (X, U, C)$, où X est l'ensemble des sites, U l'ensemble des connexions possibles, et C une fonction de coût sur les arêtes (i, j), représentant typiquement une distance.

Résoudre ce problème revient à trouver un arbre couvrant de poids minimal.

2.4.1 Algorithme de Kruskal (1956)

L'algorithme de Kruskal est un algorithme glouton permettant de construire un arbre couvrant de poids minimum pour un graphe non orienté, connexe et pondéré. Il sélectionne les arêtes de poids croissant en évitant la formation de cycles.

Description informelle

1. Ordonner les arêtes de G par poids croissant.
2. Initialiser l'arbre A comme un ensemble vide d'arêtes.
3. Parcourir chaque arête e dans l'ordre :
   — Si e ne forme pas de cycle avec les arêtes déjà dans A, alors l'ajouter à A.
4. Répéter jusqu'à obtenir $n-1$ arêtes.

Pseudocode de l'algorithme (Kruskal)

Algorithm 1: Algorithme de Kruskal

Input: $G = (X, U, W)$ un graphe non orienté, pondéré
Output: A : un arbre couvrant de poids minimum

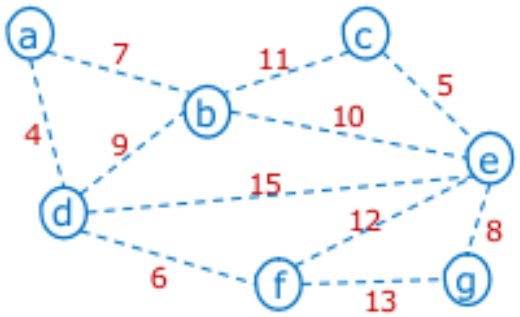
```
Trier les arêtes U par ordre croissant de poids : L ← {u_1, u_2, ..., u_m}
A ← ∅
k ← 1
while |A| < |X| − 1 et k ≤ m do
    if l'ajout de u_k à A ne crée pas de cycle then
        A ← A ∪ {u_k}
    end
    k ← k + 1
end
return A
```

Sous-procédure : Test de cycle

— Soit $u = (x_u, y_u)$ une arête et $A = (X_A, U_A)$ l'arbre partiel actuel.
— **Si** $x_u \in X_A$ et $y_u \in X_A$, **alors** ajouter u créerait un cycle.
— **Sinon**, aucun cycle n'est formé.

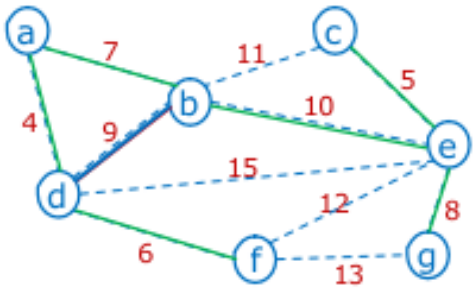
Exemple 1 :

Retrouver l'arbre couvrant de poids minimal de ce graphe à l'aide de l'algorithme de Kruskal :



Solution :

Arête	Poids	Décision
a–d	4	Pas de circuit
c–e	5	Pas de circuit
d–f	6	Pas de circuit
a–b	7	Pas de circuit
e–g	8	Pas de circuit
b–d	9	Circuit
b–e	10	Pas de circuit
b–c	11	Nombre d'arêtes = 6, soit $\|X\|-1$, donc on arrête ici.
e–f	12	Non traitée
f–g	13	Non traitée
d–e	15	Non traitée



L'arbre couvrant de poids minimal est :
$A = \{ad, ce, df, ab, eg, bc\}$,
son poids est : $w = 4+5+6+7+8+11 = \boxed{41}$.

2.4.2 Algorithme de Sollin (1961)

L'algorithme de Sollin, aussi appelé **Algorithme de Borůvka** proposé en 1961, est l'un des premiers algorithmes pour résoudre le problème de l'arbre couvrant de poids minimum. Il repose sur une stratégie gloutonne par étapes successives de fusion de composantes.

Description intuitive

On procède par étapes : à chaque étape, on joint chaque sommet à son voisin le plus proche, c'est-à-dire à l'arête sortante de coût minimal. Cela crée des sous-graphes disjoints (sous-arbres). Ensuite, chaque sous-graphe est traité comme un seul sommet. L'opération est répétée jusqu'à obtention d'un arbre couvrant unique du graphe initial.

Principe algorithmique (formulation de Borůvka)

L'idée est de commencer par une forêt où chaque sommet est une composante indépendante. À chaque itération, on relie chaque composante à une autre via l'arête sortante de plus faible poids. On fusionne ainsi les composantes progressivement jusqu'à ce qu'il n'en reste qu'une seule.

Pseudocode de l'algorithme

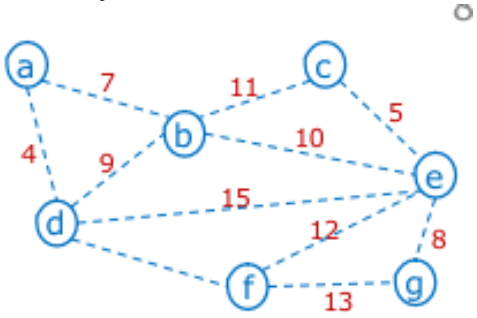
Algorithm 2: Algorithme de Borůvka (ou Sollin)

Input: $G = (X, U, W)$: graphe non orienté, pondéré
Output: F : un arbre couvrant de poids minimum

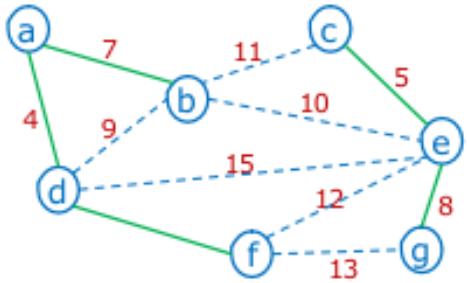
```
F ← forêt initiale contenant chaque sommet comme composante individuelle
while |F| > 1 do
    L ← ∅                          // Liste des arêtes minimales sélectionnées
    foreach composante C ∈ F do
        Trouver une arête u_p ∈ U de poids minimum liant C à une autre composante C'
        Ajouter u_p à L
    end
    Ajouter les arêtes de L à F (fusionner les composantes connectées)
end
return F
```

Exemple 1 :

Initialisation : $F = \{a, b, c, d, e, f, g\}$



Itération 1 : $|F| = 7 > 1$

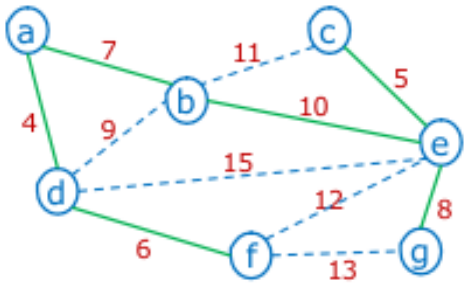


$L = \{ab, ad, ce, df, eg\}$

Ajoutons L à F, on obtient :

$F = \{\{ad, ab, df\}, \{ce, eg\}\}$

Itération 2 : $|F| = 2 > 1$



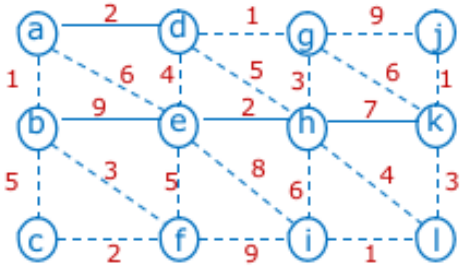
$L = \{be\}$
Ajoutons L à F, on obtient :
$F = \{\{ad, ab, df, ce, eg, be\}\}$

Itération 3 : $|F| = 1$. Fin.
F est un arbre couvrant de poids minimum.

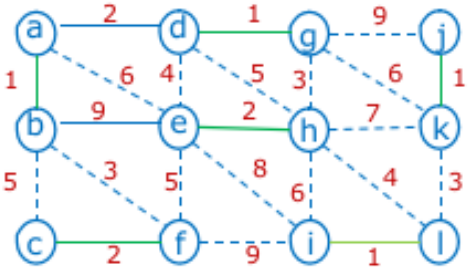
Poids total : $7+4+10+5+6+8 = \boxed{40}$

Exemple 2 :

Retrouver l'arbre de poids minimum à l'aide de l'agorithme de Boruvka.
Initialisation : $F = \{a, b, c, d, e, f, g, h, i, j, k, l\}$

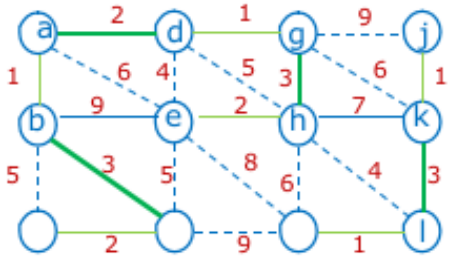


Itération 1 : $|F| = 12 > 1$



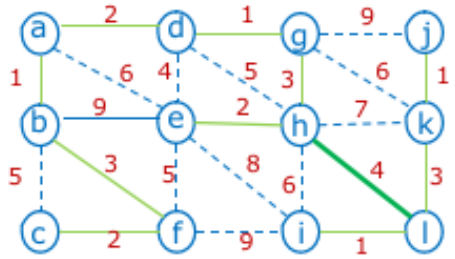
$L = \{\text{ab, cf, dg, eh, il, jk}\}$
Ajoutons L à F, on obtient :
$F = \{\{ab\},\{cf\},\{dg\},\{eh\},\{il\},\{jk\}\}$

Itération 2 : $|F| = 6 > 1$



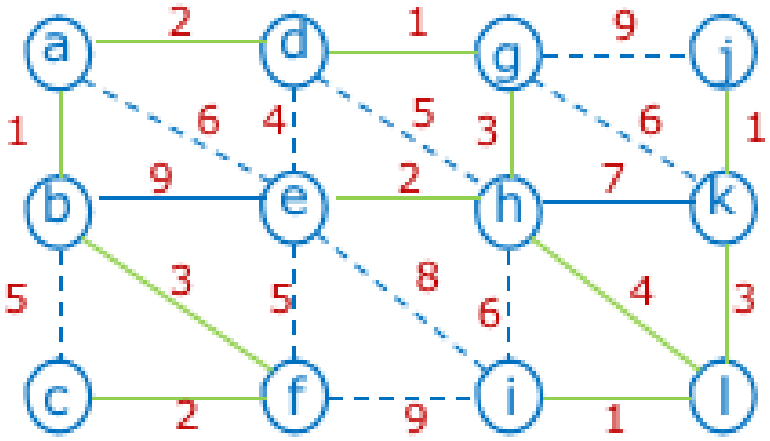
$L = \{\text{ad, bf, gh, kl}\}$
Ajoutons L à F, on obtient :
$F = \{\{\text{ab, ad, bf, cf, dg, gh, eh}\},\{\text{jk, kl, il}\}\}$

Itération 3 : $|F| = 2$



$L = \{\text{hl}\}$
Ajoutons L à F, on obtient :
$F = \{\{\text{ab, ad, bf, cf, dg, gh, eh, hl, jk, kl, il}\}\}$

Itération 4 : $|F| = 1 \Rightarrow$ **arbre couvrant de poids minimum**



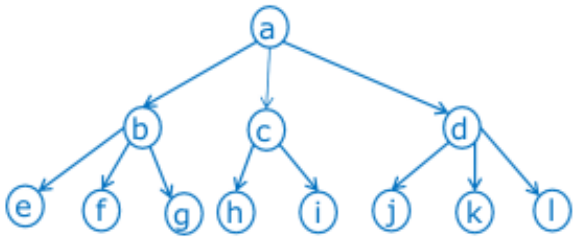
Poids total : (2+1+2+2+1) + (1+3+1+3) + (3+4) = 8 + 8 + 7 = $\boxed{23}$

2.5 Arborescence

2.5.1 Définitions

Racine

Dans un graphe orienté (X, U), un sommet est dit **racine** s'il n'a aucun prédécesseur. Si tous les autres sommets du graphe peuvent être atteints depuis cette racine par un chemin orienté, alors l'arbre est dit **enraciné**, ou bien une **arborescence**.



Branche

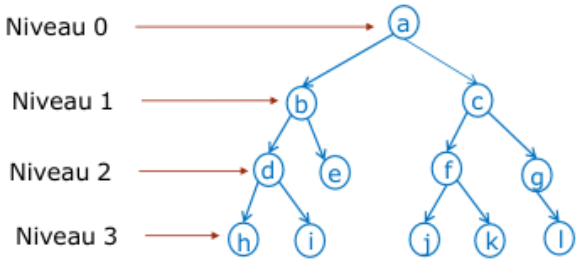
Une **branche** est une chaîne (chemin orienté) reliant la racine à une feuille de l'arbre (un sommet sans successeur).

Unicité du chemin

Dans une arborescence, il existe un **chemin unique** de la racine vers tout autre sommet du graphe. Cela garantit une structure sans cycles ni ambigüité de parcours.

Arborescence k-aire

Une arborescence est dite k**-aire** si chaque sommet possède au plus k **successeurs** (appelés ses **fils**). Lorsque $k = 2$, on parle d'un **arbre binaire**.



Niveaux

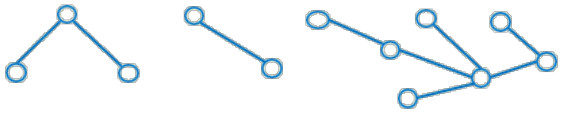
Le **niveau** d'un nœud dans une arborescence correspond à sa distance (en nombre d'arcs) par rapport à la racine. Tous les sommets équidistants de la racine appartiennent au même niveau. La racine est, par convention, au **niveau 0**.

Hauteur

La **hauteur** (ou **profondeur**) d'une arborescence est la longueur (en nombre d'arcs) du plus long chemin partant de la racine jusqu'à une feuille.

Forêt

Une **forêt** est un graphe non connexe dont chaque composante connexe est un arbre. Chaque arbre peut être considéré individuellement comme une arborescence si une racine y est définie.



Une Foret

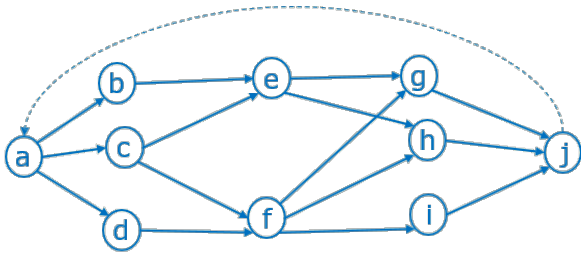
Réseau

Un **réseau** est un graphe orienté **fortement connexe**, sans boucles. Il possède deux sommets particuliers :

— **sommet d'entrée**,

— **sommet de sortie**,

liés éventuellement par un **arc fictif** garantissant la forte connexité du graphe.



Un Réseau

{a} : entrée du réseau, {j} : sortie du réseau

2.6 Représentation et Applications des Arbres en Informatique

2.6.1 Représentation Informatique des Arbres

En informatique, les arbres peuvent être représentés de différentes manières selon le contexte et le type d'arbre considéré.

Représentation Générale des Graphes

Les arbres peuvent être vus comme des cas particuliers de graphes. Ainsi, ils peuvent être représentés de manière générale par :

— Matrices d'adjacence
— Listes d'adjacence (ou listes d'arcs)

Arbres Binaires (Représentation Récursive)

— Un arbre est un pointeur vers un sommet.
— Un sommet contient :
  — Une valeur
  — Deux pointeurs : un vers le fils gauche, un vers le fils droit

Arbres k-aires (Représentation Récursive)

— Un arbre est un pointeur vers un sommet.
— Un sommet contient :
  — Une valeur
  — Deux pointeurs : un vers le *premier fils*, un vers la *liste des frères*

2.6.2 Applications des Arbres Couvrants

Les arbres couvrants ont de nombreuses applications en informatique et en génie des réseaux :
— Connexion économique d'un ensemble de stations (téléphones, terminaux, usines, etc.)
— Génération de graphes partiels pour l'analyse rapide de grands graphes
— Conception d'algorithmes de routage
— Résolution approchée de problèmes complexes : voyageur de commerce, arbre de Steiner
— Conception de réseaux efficaces

Théorème de Cayley 1889

Dans un graphe complet à n sommets, il existe n^{n-2} arbres couvrants.

2.7 Fonction Ordinale

2.7.1 Définition

Soit (X,Γ) un graphe sans circuit. On définit les sous-ensembles $N_0, N_1, N_2, \ldots, N_r$ comme suit :

$$
\begin{aligned}
N_0 &= \{x_i \mid \Gamma^{-1}(x_i) = \emptyset\} \\
N_1 &= \{x_i \mid \Gamma^{-1}(x_i) \subseteq N_0\} \\
N_2 &= \{x_i \mid \Gamma^{-1}(x_i) \subseteq N_0 \cup N_1\} \\
&\vdots \\
N_r &= \left\{ x_i \mid \Gamma^{-1}(x_i) \subseteq \bigcup_{k=0}^{r-1} N_k \right\}
\end{aligned}
$$

Le plus petit entier r est tel que $\Gamma(N_r) = \emptyset$.
Les sous-ensembles N_k forment une **partition** de X et sont appelés *niveaux*.
La **fonction ordinale** $\phi(x_i)$ est définie par :

$$x_i \in N_k \Rightarrow \phi(x_i) = k$$

2.7.2 Méthode de Détermination

a) Représenter la matrice booléenne du graphe ; poser $k = 0$.

b) Ajouter une ligne totale A_k en bas de la matrice avec des croix pour les colonnes dont les lignes ont été supprimées.

c) Faire la somme de chaque colonne et inscrire les totaux dans A_k.

d) Les sommets ayant un total nul constituent le niveau N_k ; supprimer les lignes correspondantes.

e) Incrémenter k et retourner à l'étape b.

f) Arrêter quand A_k ne contient que des croix.

2.8 Circuits et Chemins

2.8.1 Circuits et chemins Eulériens

Certains problèmes de graphes consistent à trouver un chemin entre deux sommets de sorte que chaque arête soit parcourue exactement une fois, ou à trouver un chemin entre deux sommets en visitant chaque sommet exactement une fois. Ces chemins sont mieux connus respectivement sous les noms de **chemin d'Euler** et de **chemin hamiltonien**.

Définitions

— **Chemin d'Euler** : chemin qui utilise chaque arête d'un graphe exactement une fois. Il commence et se termine à des sommets différents.

— **Circuit d'Euler** : circuit qui utilise chaque arête d'un graphe exactement une fois. Il commence et se termine au même sommet.

Conditions d'existence

Il existe des critères simples pour déterminer si un multigraphe possède un chemin d'Euler ou un circuit d'Euler. Pour qu'un multigraphe ait un circuit d'Euler, tous les degrés des sommets doivent être pairs.

Théorème 01 : Un multigraphe connexe (ou un graphe simple) avec au moins deux sommets a un circuit d'Euler si et seulement si chacun de ses sommets a un degré pair.
Preuve : Chaque fois qu'un circuit passe par un sommet, il contribue deux fois à son degré. Comme il s'agit d'un circuit, il commence et se termine au même sommet, ajoutant un degré au début et un autre à la fin. Ainsi, chaque sommet a un degré pair. Par exemple, dans le cas du graphe de Königsberg, certains sommets ont un degré impair, donc aucun circuit d'Euler n'y est possible.

Théorème 02 : Un multigraphe connexe (ou un graphe simple) a un chemin d'Euler mais pas un circuit d'Euler si et seulement s'il a exactement deux sommets de degré impair.
Preuve : Cette affirmation découle de la preuve précédente. Comme un chemin d'Euler peut commencer et se terminer en des sommets différents, ces deux sommets peuvent avoir un degré impair. Tous les autres sommets doivent avoir un degré pair pour que le chemin soit possible.

2.8.2 Circuits et chemins hamiltoniens

Définitions

— **Chemin hamiltonien** : chemin simple dans un graphe G qui passe par chaque sommet exactement une fois.

— **Circuit hamiltonien** : circuit simple dans un graphe G qui passe par chaque sommet exactement une fois.

Remarques

Contrairement aux chemins et circuits d'Euler, il n'existe pas de critères simples, à la fois nécessaires et suffisants, pour déterminer s'il existe des chemins ou des circuits hamiltoniens dans un graphe.

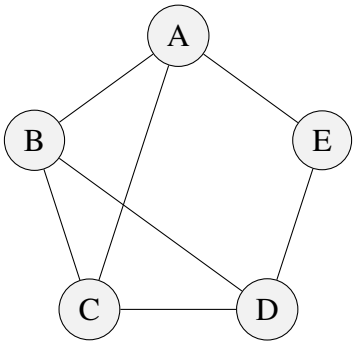
Cependant, certains critères peuvent exclure l'existence d'un circuit hamiltonien. Par exemple, s'il existe un sommet de degré 1 dans un graphe, alors il est impossible que ce graphe possède un circuit hamiltonien.

Il existe également des théorèmes qui donnent des *conditions suffisantes* (mais non nécessaires) pour garantir l'existence de circuits hamiltoniens dans certains graphes.

2.8.3 Exercice

Soit le graphe G défini par les sommets A, B, C, D, E et les arêtes suivantes :

$$\{(A,B),(B,C),(C,D),(D,E),(E,A),(A,C),(B,D)\}$$



1. Déterminez si ce graphe possède un chemin d'Euler ou un circuit d'Euler. Justifiez votre réponse.
2. Déterminez si ce graphe possède un chemin hamiltonien ou un circuit hamiltonien. Justifiez votre réponse.

Solution

1. Chemins et circuits d'Euler

Pour qu'un graphe possède un chemin d'Euler, il doit avoir exactement deux sommets de degré impair. Pour qu'il possède un circuit d'Euler, tous les sommets doivent avoir un degré pair.

Calcul des degrés des sommets :

$$\begin{aligned}\deg(A) &= 3,\\ \deg(B) &= 3,\\ \deg(C) &= 3,\\ \deg(D) &= 3,\\ \deg(E) &= 2.\end{aligned}$$

Comme il y a quatre sommets de degré impair (A, B, C, et D), le graphe ne possède **ni chemin d'Euler ni circuit d'Euler**.

2. Chemin et circuit hamiltonien

Un **chemin hamiltonien** est un chemin qui passe par chaque sommet exactement une fois. Un **circuit hamiltonien** est un circuit qui passe par chaque sommet exactement une fois et revient au point de départ.

Analyse : En traçant un chemin, nous constatons qu'il est possible de passer par chaque sommet exactement une fois selon le chemin :

$$A \to B \to C \to D \to E,$$

ce qui constitue un **chemin hamiltonien**.

Cependant, il n'existe pas de **circuit hamiltonien** car il n'est pas possible de revenir au sommet initial A après avoir visité chaque sommet une seule fois.

$\Rightarrow$ Le graphe possède un chemin hamiltonien mais pas de circuit hamiltonien.

2.9 Conclusion

Dans ce chapitre, nous avons discuté en profondeur des notions de chemin hamiltonien et de chemin d'Euler. Nous avons présenté une définition générale suivie de quelques exemples pour nous assurer que le lecteur comprend les principes fondamentaux. Enfin, nous avons rassemblé les points abordés et présenté une comparaison de haut niveau entre ces deux concepts.

CHAPITRE

3

PROBLÈME DU PLUS COURT CHEMIN

3.1 Introduction

Le problème du plus court chemin est un problème fondamental en théorie des graphes et en optimisation combinatoire. Il consiste à trouver le chemin de coût minimal entre deux sommets d'un graphe pondéré. Ce problème a de nombreuses applications dans divers domaines tels que la planification des itinéraires, les réseaux informatiques, la logistique et les télécommunications.

Plusieurs algorithmes classiques permettent de résoudre ces problèmes, notamment :

— L'algorithme de Dantzig : une approche matricielle adaptée aux graphes complets.

— L'algorithme de Dijkstra : une méthode gloutonne pour les graphes avec des poids positifs.

— L'algorithme de Ford (Bellman-Ford) : efficace pour traiter les graphes avec des poids négatifs.

Chaque algorithme présente des avantages et des limitations selon la nature du graphe et le type de problème à résoudre.

3.2 Les Algorithmes de Plus Court Chemin

3.2.1 Algorithme de Dijkstra

Introduction

L'algorithme de Dijkstra, conçu par Edsger Dijkstra en 1956, demeure un pilier de la théorie des graphes. Il s'applique aux graphes $G = (X, U)$ pondérés par des valeurs positives $l(u)$ pour tout $u \in U$. Il permet de déterminer les longueurs des plus courts chemins à partir d'un sommet source $a \in X$ vers tous les autres sommets.

Le principe consiste à affecter une valeur $\text{dist}(x)$ à chaque sommet $x \in X$, initialisée à 0 pour le sommet source a et à $+\infty$ pour les autres. L'algorithme tente de réduire cette valeur au fur et à mesure de son exécution.

Pseudocode de l'algorithme Dijkstra

Algorithm 3: Algorithme de Dijkstra

Input: $G = (X, U, W)$ un graphe pondéré orienté avec $w_i > 0$ pour tout i, et un sommet source s

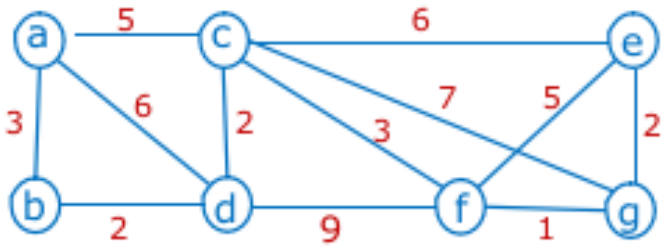
Output: Tableau des distances D et des prédécesseurs P

```
1  foreach x ∈ X do
2  |   D[x] ← ∞
3  |   P[x] ← ∅
4  end
5  D[s] ← 0
6  F ← X                          // Ensemble des sommets non encore traités
7  while F ≠ ∅ do
8  |   x ← arg min_{z∈F} D[z]
9  |   F ← F \ {x}
10 |   foreach y ∈ Successeurs(x) do
11 |   |   if D[y] > D[x] + w(x,y) then
12 |   |   |   D[y] ← D[x] + w(x,y)
13 |   |   |   P[y] ← x
14 |   |   end
15 |   end
16 end
```

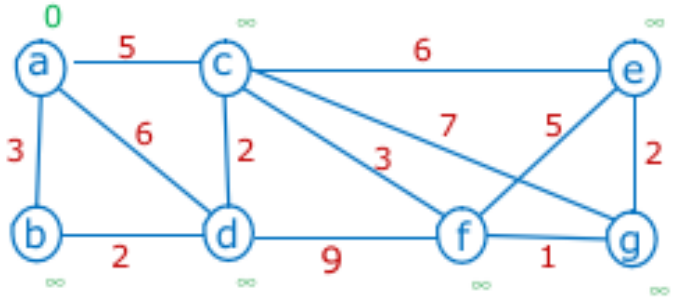
Exemple 1

Appliquer l'algorithme de Dijkstra pour retrouver les plus courts chemins entre le sommet a et les autres sommets du graphe suivant :

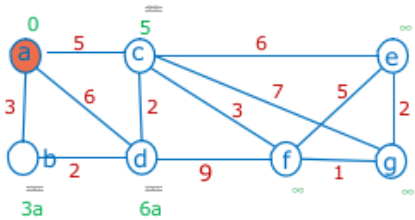
1iere approche : Graphique



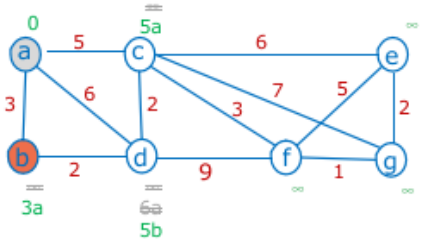
Initialisation



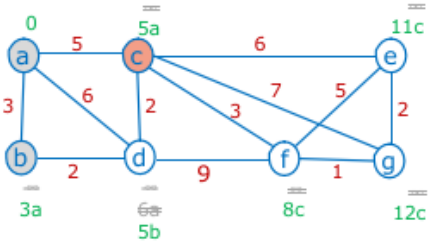
Itération 1:



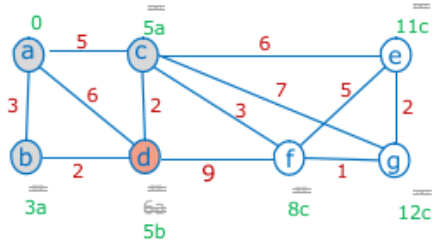
Itération 2 :



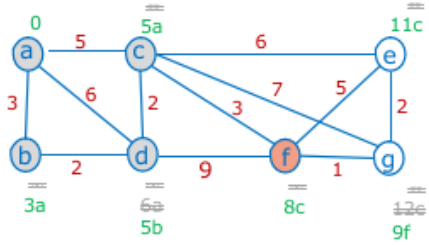
Itération 3 :



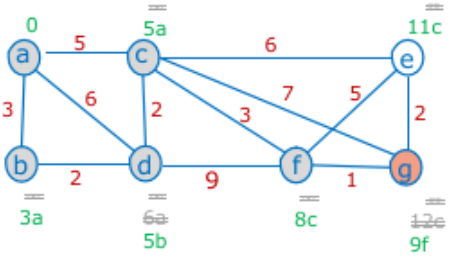
Itération 4 :



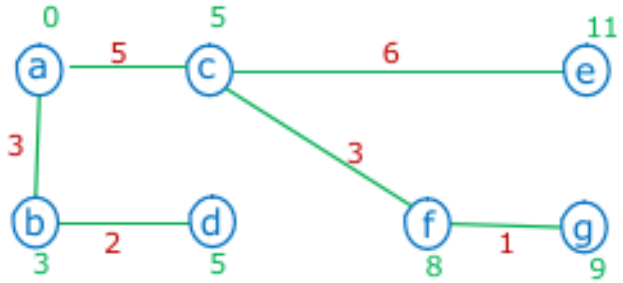
Itération 5 :



Itération 6 :

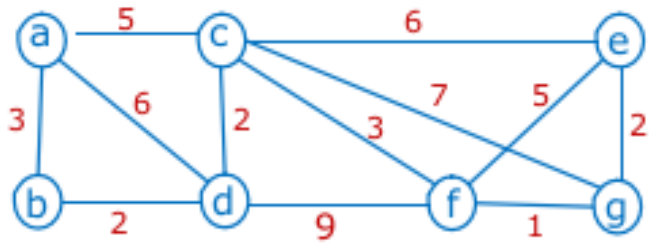


Résultat de l'algorithme



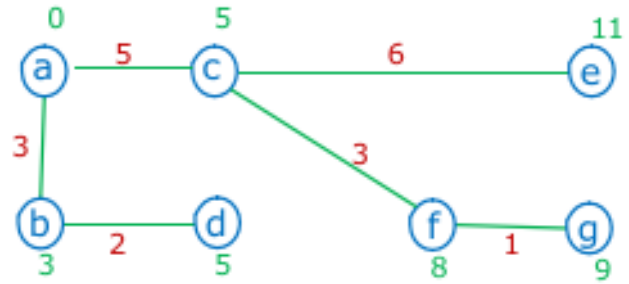
	a	b	c	d	e	f	g
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f

2ieme approche : Tableau



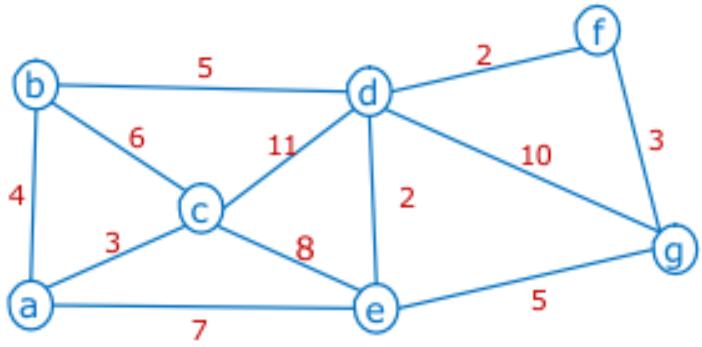
	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞
a	0	**3*a***	**5*a***	**6*a***	∞	∞	∞
b	0	3*a*	**5*a***	**5*b***	∞	∞	∞
c	0	3*a*	5*a*	**5*b***	**11*c***	**8*c***	**12*c***
d	0	3*a*	5*a*	5*b*	**11*c***	**8*c***	**12*c***
f	0	3*a*	5*a*	5*b*	**11*c***	8*c*	**9*f***
g	0	3*a*	5*a*	5*b*	**11*c***	8*c*	9*f*

	a	b	c	d	e	f	g
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f



Exemple 2

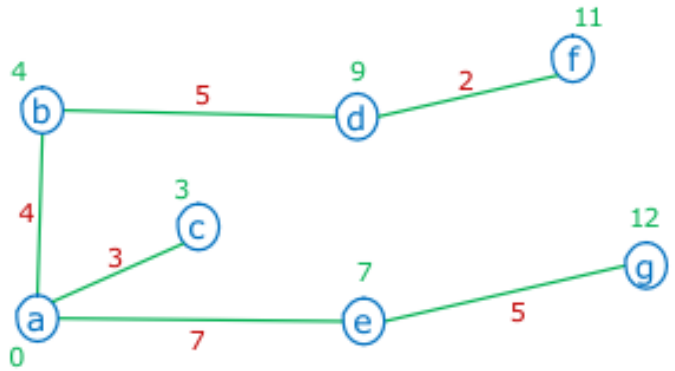
Appliquer l'algorithme de Dijkstra pour déterminer les plus courts chemins en partant du sommet *a* du graphe ci-dessous :



Solution :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞
a	0	**4*a***	**3*a***	∞	**7*a***	∞	∞
c	0	**4*a***	3*a*	**14*c***	**7*a***	∞	∞
b	0	4*a*	3*a*	**9*b***	**7*a***	∞	∞
e	0	4*a*	3*a*	**9*b***	7*a*	∞	**12*e***
d	0	4*a*	3*a*	9*b*	7*a*	**11*d***	**12*e***
f	0	4*a*	3*a*	9*b*	7*a*	11*d*	**12*e***

	a	*b*	*c*	*d*	*e*	*f*	*g*
D	0	4	3	9	7	11	12
P	∅	a	a	b	a	d	e



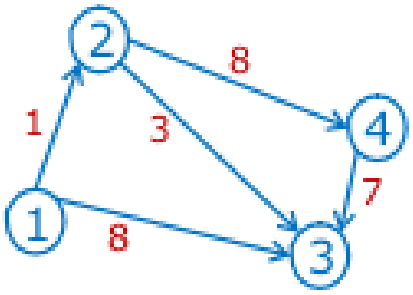
3.2.2 Algorithme de Dantzig

Il détermine les plus courts chemins entre tous les couples de sommets (x,y) du graphe (problème P2). Les résultats sont donnés dans :

— **Matrice des distances D** $= [d_{ij}]$ où d_{ij} est la plus courte distance entre les sommets i et j.

— **Matrice des prédécesseurs P** $= [p_{ij}]$ où p_{ij} est le prédécesseur de j dans le plus court chemin entre i et j.

Exemple : Soit le graphe à 4 sommets suivant :



$$\mathbf{D} = \begin{bmatrix} d_{11} & d_{12} & d_{13} & d_{14} \\ d_{21} & d_{22} & d_{23} & d_{24} \\ d_{31} & d_{32} & d_{33} & d_{34} \\ d_{41} & d_{42} & d_{43} & d_{44} \end{bmatrix} \qquad \mathbf{P} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \\ p_{41} & p_{42} & p_{43} & p_{44} \end{bmatrix}$$

— $[d_{i1},\ d_{i2},\ d_{i3},\ d_{i4}]$ est le vecteur des plus courtes distances entre le sommet i et les autres sommets du graphe.

— $[p_{i1},\ p_{i2},\ p_{i3},\ p_{i4}]$ sont les prédécesseurs des sommets 1, 2, 3 et 4 sur les plus courts chemins à partir du sommet i, avec $i = 1 \ldots 4$.

Principe de l'algorithme : L'idée est de :

— Commencer par le graphe G_0 possédant un seul sommet x_1 et construire les matrices :

$$D^{(0)} = [0] \quad \text{et} \quad P^{(0)} = [1]$$

— Construire le graphe G_1 en ajoutant le sommet x_2 à G_0, puis construire les matrices :

$$D^{(1)} = \begin{bmatrix} 0 & d \\ d_{21} & d_{22} \end{bmatrix} \quad \text{et} \quad P^{(1)} = \begin{bmatrix} 0 & p \\ p_{21} & p_{22} \end{bmatrix}$$

— Ajouter graduellement d'autres sommets $x_3, \ldots, x_n$ jusqu'à l'obtention du graphe $G_n = G$ en construisant au passage les matrices :

$$D^{(2)}, \ldots, D^{(n-1)} \quad \text{et} \quad P^{(2)}, \ldots, P^{(n-1)}$$

avec

$$D^{(n-1)} = D \quad \text{et} \quad P^{(n-1)} = P$$

Théorème : Si le graphe G est sans circuit absorbant, l'algorithme de Dantzig résout le problème P2 en un temps $\mathcal{O}(n^3)$.

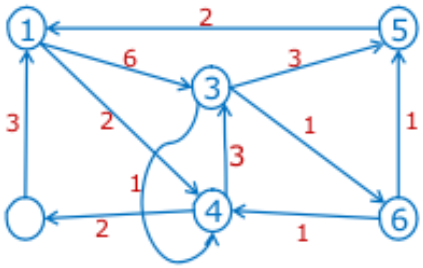
Pseudocode de l'algorithme Dantzig

Algorithm 4: Procédure de Dantzig pour les plus courts chemins (Problème P2)

```
   Input: M                                                    // Matrice d'adjacence
   Output: D = [d_ij],   P = [p_ij]
 1 Initialisation : n = dimension(M)
 2 for i ← 1 to n do
 3     for j ← 1 to n do
 4         if i = j then
 5             d_ij ← 0,   p_ij ← i
 6         else
 7             d_ij ← ∞,   p_ij ← 0
 8 for k ← 1 to n − 1 do
 9     for i ← 1 to k do
10         for j ← 1 to k do
               // Bloc A | propagation de distances via le sommet j
11             d ← D(i, j) + M(j, k + 1)
12             if d < D(i, k + 1) then
13                 D(i, k + 1) ← d
14                 P(i, k + 1) ← j
15             d ← M(k + 1, j) + D(j, i)
16             if d < D(k + 1, i) then
17                 D(k + 1, i) ← d
18                 if i = j then
19                     P(k + 1, i) ← k + 1
20                 else
21                     P(k + 1, i) ← P(j, i)
22     for i ← 1 to k do
23         for j ← 1 to k do
               // Bloc B | mise à jour complète
24             d ← D(i, k + 1) + D(k + 1, j)
25             if d < D(i, j) then
26                 D(i, j) ← d
27                 P(i, j) ← P(k + 1, j)
```

Exemple :

Utiliser l'algorithme de Dantzig pour retrouver tous les plus courts chemins entre les sommets du graphe suivants :



Solution :

$$M = \begin{bmatrix} 0 & \infty & 6 & 2 & \infty & \infty \\ 3 & 0 & \infty & \infty & \infty & \infty \\ \infty & \infty & 0 & 1 & 3 & 1 \\ \infty & 2 & 3 & 0 & \infty & \infty \\ 2 & \infty & \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 1 & 1 & 0 \end{bmatrix}$$

— **K = 0 :** On commence avec le sommet $\{1\}$.

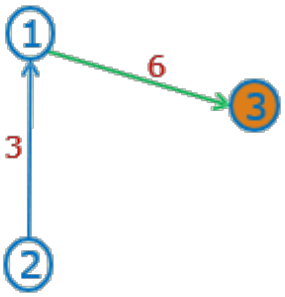
$$D^{(0)} = [0], \quad P^{(0)} = [1]$$

— **K = 1 :** On ajoute le sommet $\{2\}$.

$$D^{(1)} = \begin{bmatrix} 0 & \infty \\ 3 & 0 \end{bmatrix}, \quad P^{(1)} = \begin{bmatrix} 1 & 0 \\ 2 & 2 \end{bmatrix}$$

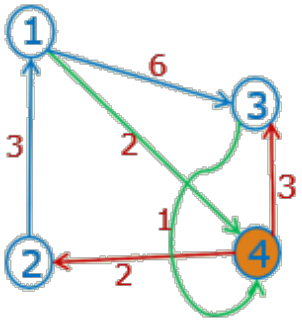
— **K = 2 :** On ajoute le sommet $\{3\}$.

$$D^{(2)} = \begin{bmatrix} 0 & \infty & 6 \\ 3 & 0 & 9 \\ \infty & \infty & 0 \end{bmatrix}, \quad P^{(2)} = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 2 & 1 \\ 0 & 0 & 3 \end{bmatrix}$$



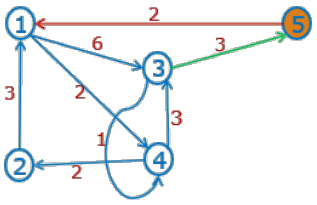
— **K = 3 :** On ajoute le sommet $\{4\}$.

$$D^{(3)} = \begin{vmatrix} 0 & 4 & 5 & 2 \\ 3 & 0 & 8 & 5 \\ 6 & 3 & 0 & 1 \\ 5 & 2 & 3 & 0 \end{vmatrix}, \quad P^{(3)} = \begin{bmatrix} 1 & 4 & 4 & 1 \\ 2 & 2 & 4 & 1 \\ 2 & 4 & 3 & 3 \\ 2 & 4 & 4 & 4 \end{bmatrix}$$



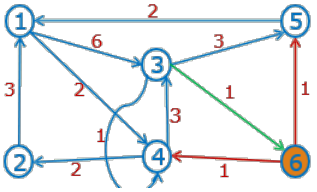
— **K = 4 :** On ajoute le sommet $\{5\}$.

$$D^{(4)} = \begin{bmatrix} 0 & 4 & 5 & 2 & 8 \\ 3 & 0 & 8 & 5 & 11 \\ 5 & 3 & 0 & 1 & 3 \\ 5 & 2 & 3 & 0 & 6 \\ 2 & 6 & 7 & 4 & 0 \end{bmatrix}, \quad P^{(4)} = \begin{bmatrix} 1 & 4 & 4 & 1 & 3 \\ 2 & 2 & 4 & 1 & 3 \\ 5 & 4 & 3 & 3 & 3 \\ 2 & 4 & 4 & 4 & 3 \\ 5 & 4 & 4 & 1 & 5 \end{bmatrix}$$



— **K = 5 :** On ajoute le sommet $\{6\}$.

$$D^{(5)} = \begin{bmatrix} 0 & 4 & 5 & 2 & 7 & 6 \\ 3 & 0 & 8 & 5 & 10 & 9 \\ 4 & 3 & 0 & 1 & 2 & 1 \\ 5 & 2 & 3 & 0 & 5 & 4 \\ 2 & 6 & 7 & 4 & 0 & 8 \\ 3 & 3 & 4 & 1 & 1 & 0 \end{bmatrix}, \quad P^{(5)} = \begin{bmatrix} 1 & 4 & 4 & 1 & 6 & 3 \\ 2 & 2 & 4 & 1 & 6 & 3 \\ 5 & 4 & 3 & 3 & 6 & 3 \\ 2 & 4 & 4 & 4 & 6 & 3 \\ 5 & 4 & 4 & 1 & 5 & 3 \\ 5 & 4 & 4 & 6 & 6 & 6 \end{bmatrix}$$



3.2.3 Algorithme de Ford (Bellman-Ford)

Contrairement à Dijkstra, cet algorithme peut gérer les poids négatifs. Il repose sur le principe de ”relaxation” :

1. Initialiser la distance de départ à 0, les autres à $+\infty$.
2. Répéter $n-1$ fois :
   — Pour chaque arête, mettre à jour la distance des voisins si un chemin plus court est trouvé.
3. Vérifier à la fin s'il existe un cycle de poids négatif.

Pseudocode de l'algorithme Ford

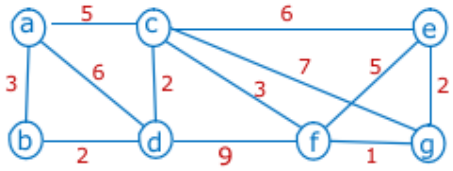
Algorithm 5: Procédure Bellman-Ford pour la recherche des plus courts chemins

Data: G = (X, U, W), s : sommet source
Result: D : tableau des distances, P : tableau des prédécesseurs

foreach $x \in X$ **do**
    $D[x] \leftarrow \infty$
    $P[x] \leftarrow \emptyset$
$D[s] \leftarrow 0$
`/* Étape de relaxation */`
for $i \leftarrow 1$ **to** $|X|-1$ **do**
    foreach $(u,v) \in U$ **do**
        if $D[v] > D[u] + w(u,v)$ **then**
            $D[v] \leftarrow D[u] + w(u,v)$
            $P[v] \leftarrow u$
`/* Détection de circuit absorbant */`
foreach $(u,v) \in U$ **do**
    if $D[u] + w(u,v) < D[v]$ **then**
        return *“Il existe un circuit absorbant”*
return *“Pas de circuit absorbant”*

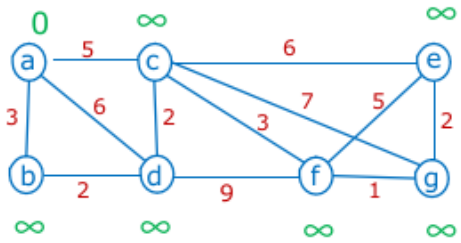
Exemple 1

Appliquer l'algorithme de Bellman-Ford pour retrouver les plus courts chemins entre le sommet a et les autres sommets du graphe suivant :



Solution :
Initialisation : On initialise :

$$D(a) = 0 \quad \text{et} \quad D(x) = \infty \quad \text{pour tout autre sommet } x \neq a.$$



À chaque itération, le parcours séquentiel des arcs du graphe doit se faire dans l'ordre suivant :

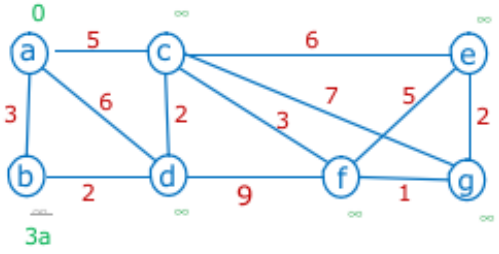
```
ab, ac, ad,
ba, bd,
ca, cd, ce, cf, cg,
da, db, dc, df,
ec, ef, eg,
fc, fd, fe, fg,
gc, ge, gf
```

1iere approche : Graphique

Itération 1 :
On parcourt toutes les arêtes du graphe pour voir si on peut améliorer certaines distances :

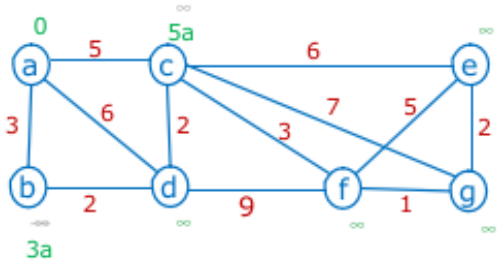
- Avec l'arête (a,b), on améliore $D(b)$:

$$D(b) = 3$$



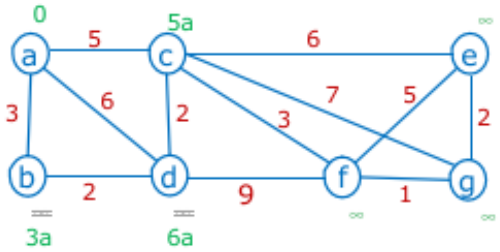
- Avec l'arête (a,c), on améliore $D(c)$:

$$D(c) = 5$$



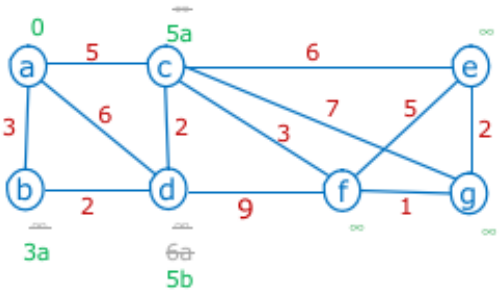
- Avec l'arête (a,d), on améliore $D(d)$:

$$D(d) = 6$$



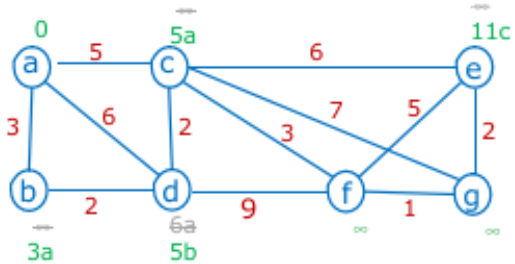
- Arête (b,a), aucune amélioration.
- Avec l'arête (b,d), on améliore $D(d)$:

$$D(d) = 3+2 = 5$$



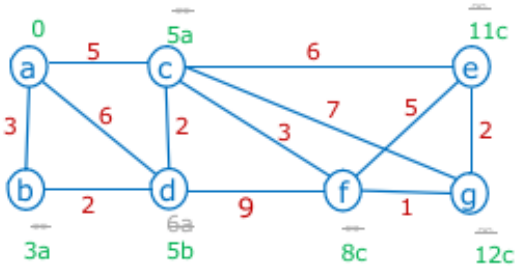
- Arête (c,a), aucune amélioration.
- Arête (c,d), aucune amélioration.
- Avec l'arête (c,e), on améliore $D(e)$:

$$D(e) = 5+6 = 11$$



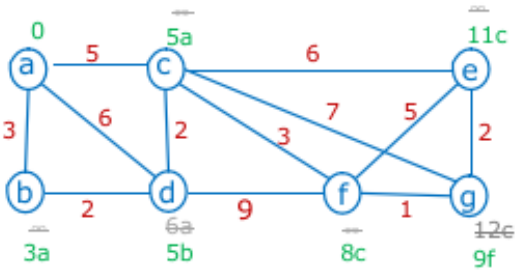
- Avec l'arête (c,g), on améliore $D(g)$:

$$D(g) = 5+7 = 12$$



- Les arêtes $(d,a),(d,b),(d,c),(d,f)$, aucune amélioration.
- Les arêtes $(e,c),(e,f),(e,g)$, aucune amélioration.
- Les arêtes $(f,c),(f,d),(f,e)$, aucune amélioration.
- Avec l'arête (f,g), on améliore $D(g)$:

$$D(g) = 8+1 = 9$$



- Les arêtes $(g,c),(g,e),(g,f)$, aucune amélioration.

Itération 2 :
On parcourt à nouveau toutes les arêtes du graphe pour voir si on peut améliorer certaines distances :

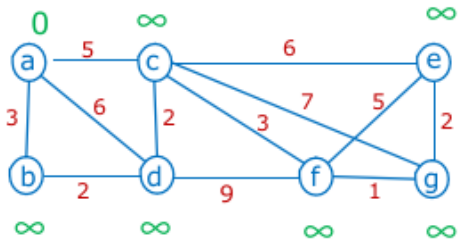
- Les arêtes (a,b), (a,c), (a,d) : aucune amélioration.
- Les arêtes (b,a), (b,d) : aucune amélioration.
- Les arêtes (c,a), (c,d), (c,e), (c,f), (c,g) : aucune amélioration.
- Les arêtes (d,a), (d,b), (d,c), (d,f) : aucune amélioration.
- Les arêtes (e,c), (e,f), (e,g) : aucune amélioration.
- Les arêtes (f,c), (f,d), (f,e), (f,g) : aucune amélioration.
- Les arêtes (g,c), (g,e), (g,f) : aucune amélioration.

L'itération 2 n'améliore aucune distance $D(x)$, on arrête donc l'algorithme à ce niveau.
Comme le nombre d'itérations $i = 2 < |\mathscr{X}| - 1 = 7 - 1 = 6$, la solution obtenue est **optimale**.

Les chemins obtenus sont :

	a	b	c	d	e	f	g
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f

2ieme approche : Tableau
On affiche seulement les arcs qui améliorent les distances.



Initialisation :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞

Itération 1 :

	a	b	c	d	e	f	g
ab		3					
ac			5				
ad				6			
bd				5			
ce					11		
cf						8	
cg							12

Itération 2 :
On parcourt à nouveaux tous les 24 arêtes du graphe. On constate qu'aucun d'eux n'améliore la distance d'un sommet. On arrête donc à cette itération.
Cette solution est optimale.

	a	b	c	d	e	f	g
D	0	3	5	5	11	8	9
P	∅	a	a	b	c	c	f

Exemple 2 (avec circuit absorbant)

Utiliser l'algorithme de Bellman pour retrouver les plus courts chemins partant du sommet *a* pour le graphe ci-dessous :
Solution :
Initialisation :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞

Itération 01 :

	a	b	c	d	e	f	g
	0	∞	∞	∞	∞	∞	∞
ab		4					
ac			3				
bd				9			
ce					11		
df						12	
eg							13
gd				3			

Itération 02 :

	a	b	c	d	e	f	g
	0	4	3	3	11	12	13
df						6	
fg							11
gd				1			

Itération 03 :

	a	b	c	d	e	f	g
	0	4	3	1	11	6	11
df						4	
fg							9
gd				-1			

Itération 04 :

	a	b	c	d	e	f	g
	0	4	3	-1	11	4	9
df						2	
fg							7
gd				-3			

Itération 05 :

	a	b	c	d	e	f	g
	0	4	3	-3	11	2	7
df						0	
fg							5
gd				-5			

Itération 06 :

	a	b	c	d	e	f	g
	0	4	3	-5	11	0	5
df						-2	
fg							3
gd				-7			

L'itération 6 doit être la dernière ($|\mathscr{X}|-1=7-1=6$). Faisons une itération de plus pour voir s'il y aurait un circuit absorbant.
Itération 07 :

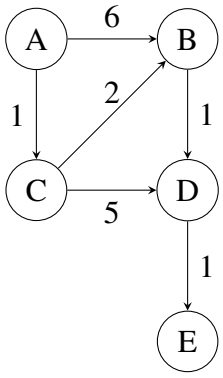
	a	b	c	d	E	f	g
	0	4	3	-7	11	-2	3
df						-4	
fg							1
gd				-9			

Cette 7^{e} itération améliore les distances, donc il y a un **circuit absorbant** dans ce graphe.

3.3 Exercice

Considérez le graphe orienté pondéré suivant représenté par les sommets A, B, C, D, E et les arêtes pondérées :

$$\{(A,B,6),\ (A,C,1),\ (C,B,2),\ (B,D,1),\ (C,D,5),\ (D,E,1)\}$$



1. Utilisez l'algorithme de Dijkstra pour trouver le chemin le plus court de A vers tous les autres sommets.

2. Si le poids de l'arête (A,B) devient -6, l'algorithme de Dijkstra fonctionnera-t-il toujours correctement ? Si non, expliquez pourquoi et proposez une solution alternative.

Solution :

1. **Algorithme de Dijkstra avec poids positifs :**
   - Initialisation :
     $d(A) = 0, \quad d(B) = \infty, \quad d(C) = \infty, \quad d(D) = \infty, \quad d(E) = \infty$
   - Itérations :
     - Visitez A, mettez à jour : $d(B) = 6$, $d(C) = 1$
     - Visitez C, mettez à jour : $d(B) = 3$, $d(D) = 6$
     - Visitez B, mettez à jour : $d(D) = 4$
     - Visitez D, mettez à jour : $d(E) = 5$
   - **Résultat final :**
     $d(A) = 0, \quad d(B) = 3, \quad d(C) = 1, \quad d(D) = 4, \quad d(E) = 5$
2. **Poids négatif :**
   Si le poids de l'arête (A,B) devient -6, l'algorithme de **Dijkstra** ne fonctionne plus correctement car il suppose que les poids des arêtes sont tous positifs. En présence de poids négatifs, il pourrait conclure prématurément qu'un chemin est optimal, alors qu'un meilleur chemin existe.

   Solution alternative : Utiliser l'algorithme de **Bellman-Ford**, qui gère les poids négatifs et peut également détecter les circuits absorbants (poids total négatif).

3.4 Conclusion

Le choix de l'algorithme dépend des caractéristiques du graphe. Pour des graphes sans poids négatif, l'algorithme de **Dijkstra** est plus rapide et efficace. **Bellman-Ford** est nécessaire pour les graphes avec des arêtes de poids négatif, bien qu'il soit plus lent. Enfin, pour les plus courts chemins entre toutes les paires de sommets, l'algorithme de **Floyd-Warshall** est souvent utilisé, bien qu'il soit plus coûteux en termes de complexité temporelle.

CHAPITRE

4

PROBLÈME DU FLOT MAXIMUM

4.1 Introduction

Le problème du flot maximum cherche à déterminer la quantité maximale d'une matière (marchandise, eau, électricité, données, etc.) que l'on peut faire transiter d'un sommet **source** s vers un sommet **puits** t, à travers un réseau orienté et pondéré, tout en respectant :

— la capacité maximale de chaque arc,

— la conservation du flot dans les sommets intermédiaires.

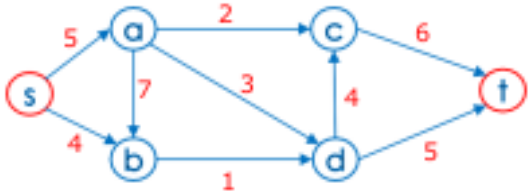
4.2 Notions de Base

4.2.1 Réseau

Un **réseau** est un graphe orienté et valué $G = (X, U, W)$ ayant deux sommets particuliers :

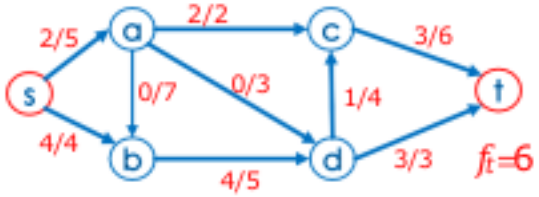
— la **source** s

— le **puits** t



4.2.2 Flot

Un **flot** est une quantité de matière (marchandise, eau, électricité, information, etc.) envoyée par le sommet s, qui traverse certains arcs du graphe pour atteindre le puits t.



Contraintes sur un flot réalisable

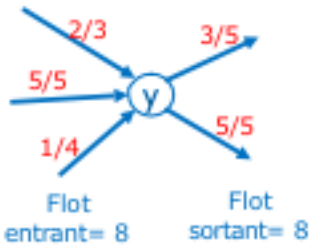
Un flot réalisable f respecte les trois contraintes suivantes :

1. **Contrainte de capacité :** Pour tout arc $u \in U$:

$$0 \leq f(u) \leq w(u)$$

2. **Contrainte de conservation :** Pour tout sommet $x \in X \setminus \{s,t\}$, le flot est conservé :

$$\sum_{y \in d^-(x)} f(y,x) = \sum_{y \in d^+(x)} f(x,y)$$



3. **Flot total :** Le flot total f est la quantité envoyée par la source s et reçue par le puits t :

$$f = \sum_{y \in d^+(s)} f(s,y) = \sum_{y \in d^-(t)} f(y,t)$$

4.2.3 Objectif du problème

Le **problème du flot maximum** consiste à déterminer les valeurs des flots élémentaires $f(u)$ pour tout $u \in U$ qui maximisent le flot total f envoyé de s à t.

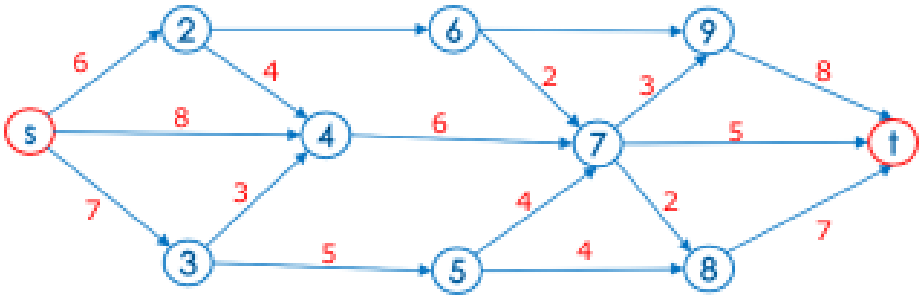
Notation matricielle :

La **matrice des flots élémentaires** est notée $F = [f_{ij}]$ où f_{ij} représente le flot sur l'arc $(i,j) \in U$.
Le flot total f peut aussi être exprimé par une double somme :

$$f = \sum_{i=1}^{n} \sum_{j=1}^{n} f_{ij}$$

4.2.4 Exemple de Flot

Soit le réseau de distribution d'eau suivant :



— Il est composé des conduites $(s,2),(s,4),\ldots$

— La valeur de chaque arc représente la quantité maximale que la conduite peut transporter.

— La source de l'eau est le sommet s, et la destination de l'eau est le sommet t.

Question

Trouver la quantité maximale d'eau qu'on peut transporter de la source vers la destination en respectant les capacités des conduites.

Réseau avec plusieurs Sources/Puits

Si le graphe possède plusieurs sources $s_1, \ldots, s_k$, on ajoute une source fictive s_0 avec :

$$w(s_0, s_i) = \sum_{y \in d^+(s_i)} w(s_i, y)$$

Si le graphe possède plusieurs puits $t_1, \ldots, t_k$, on ajoute un puits fictif t_0 avec :

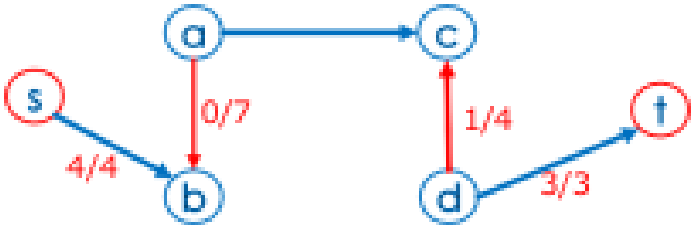
$$c(t_i, t_0) = \sum_{y \in d^-(t_i)} c(y, t_i)$$

4.2.5 Chaîne Améliorante

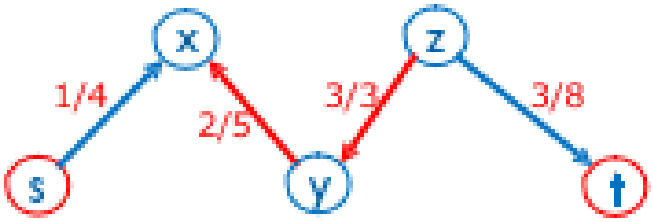
Une chaîne C est dite **améliorante** si elle lie la source s au puits t et satisfait les conditions suivantes :

— Les arcs u directs vérifient : $f(u) < w(u)$

— Les arcs u inversés vérifient : $f(u) > 0$



Par exemple, les arcs $(s,b),(a,c),(d,t)$ sont des arcs directs, tandis que $(a,b),(d,c)$ sont des arcs inversés.
Cette chaîne **n'est pas améliorante** car $f(a,b) \leq 0$.



Cette chaîne **est améliorante** car :

— Les arcs directs vérifient :

$$f(s,x) < w(s,x), \quad f(z,t) < w(z,t)$$

— Les arcs inversés vérifient :

$$f(y,x) > 0, \quad f(z,y) > 0$$

Sur cette chaîne, on peut augmenter le flot de :

$$\varepsilon = \min(\varepsilon_1, \varepsilon_2)$$

— $\varepsilon_1 = \min\{w(u) - f(u) \mid u \in C^+\}$, où C^+ est l'ensemble des arcs directs de la chaîne C.
— $\varepsilon_2 = \min\{f(u) \mid u \in C^-\}$, où C^- est l'ensemble des arcs indirects de la chaîne C.

Les nouveaux flots élémentaires $f'(u)$ sur cette chaîne seront :

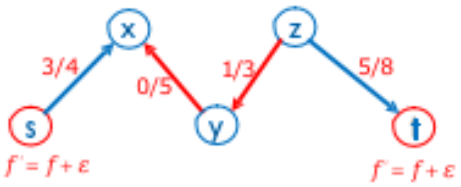
$$f'(u) = \begin{cases} f(u) + \varepsilon, & \text{si } u \in C^+ \\ f(u) - \varepsilon, & \text{si } u \in C^- \end{cases}$$

On vérifie facilement que le nouveau flot est réalisable.

Calcul

$$\varepsilon_1 = \min(4-1, 8-3) = \min(3,5) = 3$$
$$\varepsilon_2 = \min(2,3) = \min(3,5) = 2$$
$$\Rightarrow \varepsilon = \min(3,2) = 2$$

Les nouveaux flots élémentaires sur cette chaîne sont :



$$f(s,x) = 1+2 = 3$$
$$f(y,x) = 2-2 = 0$$
$$f(z,y) = 3-2 = 1$$
$$f(z,t) = 3+2 = 5$$

4.3 Coupe Minimale et Flot Maximum

4.3.1 Définition d'une coupe

Une **coupe** $(Y, \bar{Y})$ est définie par l'ensemble $Y \subset X$ et son complément $\bar{Y}$, tel que :

$$Y \cup \bar{Y} = X \quad \text{et} \quad Y \cap \bar{Y} = \emptyset$$

avec : $s \in Y$ et $t \in \bar{Y}$.
La **capacité d'une coupe** est définie par :

$$w(Y, \bar{Y}) = \sum_{\substack{x \in Y \\ y \in \bar{Y}}} w(x,y) - \sum_{\substack{x \in \bar{Y} \\ y \in Y}} w(y,x)$$

4.3.2 Théorème de Ford-Fulkerson

Soit $G = (X, U, W)$ un graphe orienté valué. Pour tout flot réalisable F de valeur f, et toute coupe $(Y, \overline{Y})$, on a :

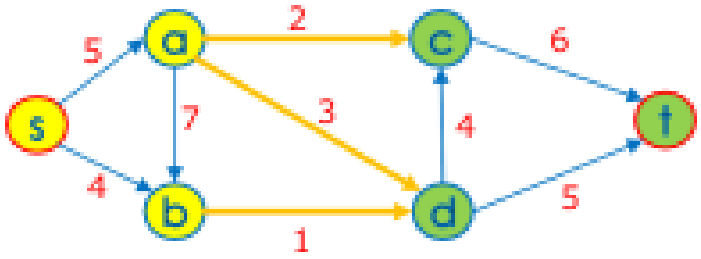
$$f \leq w(Y, \overline{Y})$$

Corollaire : S'il existe un flot F^* et une coupe Y^* tels que :

$$f^* = w(Y^*, \overline{Y^*})$$

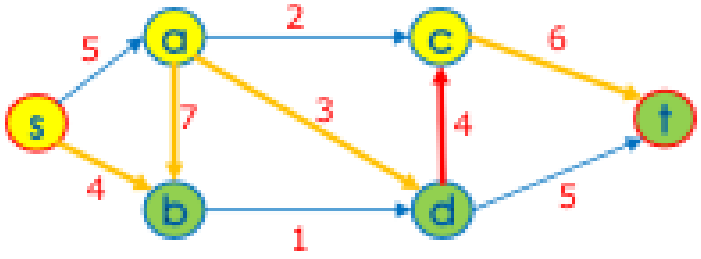
alors F^* est un **flot maximal**, et Y^* est appelée une **coupe minimale**.

Exemples :



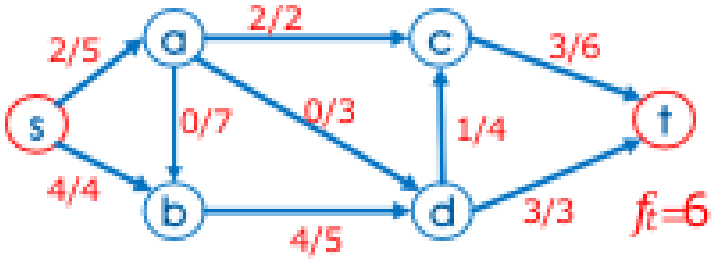
— Soit $Y_1 = \{s,a,b\}$ et $\overline{Y_1} = \{c,d,t\}$. La capacité de cette coupe est :

$$w(Y_1,\overline{Y_1}) = w(a,c) + w(a,d) + w(b,d) = 2+3+1 = 6$$



— Soit $Y_2 = \{s,a,c\}$ et $\overline{Y_2} = \{b,d,t\}$. La capacité de cette coupe est :

$$w(Y_2,\overline{Y_2}) = w(s,b) + w(a,b) + w(a,d) - w(d,t) + w(c,t) = 4+7+3-4+6 = 16$$



On peut vérifier que la coupe $(Y_1,\overline{Y_1})$ est **minimale**, donc le flot associé est **maximal**.

4.4 Parcours d'un Graphe

On parcourt un graphe pour de nombreuses raisons :

— Énumérer tous les sommets du graphe.

— Énumérer tous les arcs du graphe.

— Trouver un chemin, ou tous les chemins, entre deux sommets x et y.

— Détecter les composantes connexes d'un graphe.

— Etc.

4.4.1 Algorithme DFS (Depth-First Search)

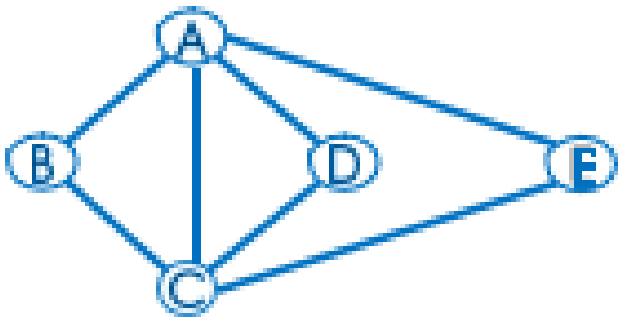
Algorithm 6: Parcours en profondeur (DFS)

Input: Graphe $G(X,U)$, sommet de départ x
Output: Chaîne C des sommets parcourus

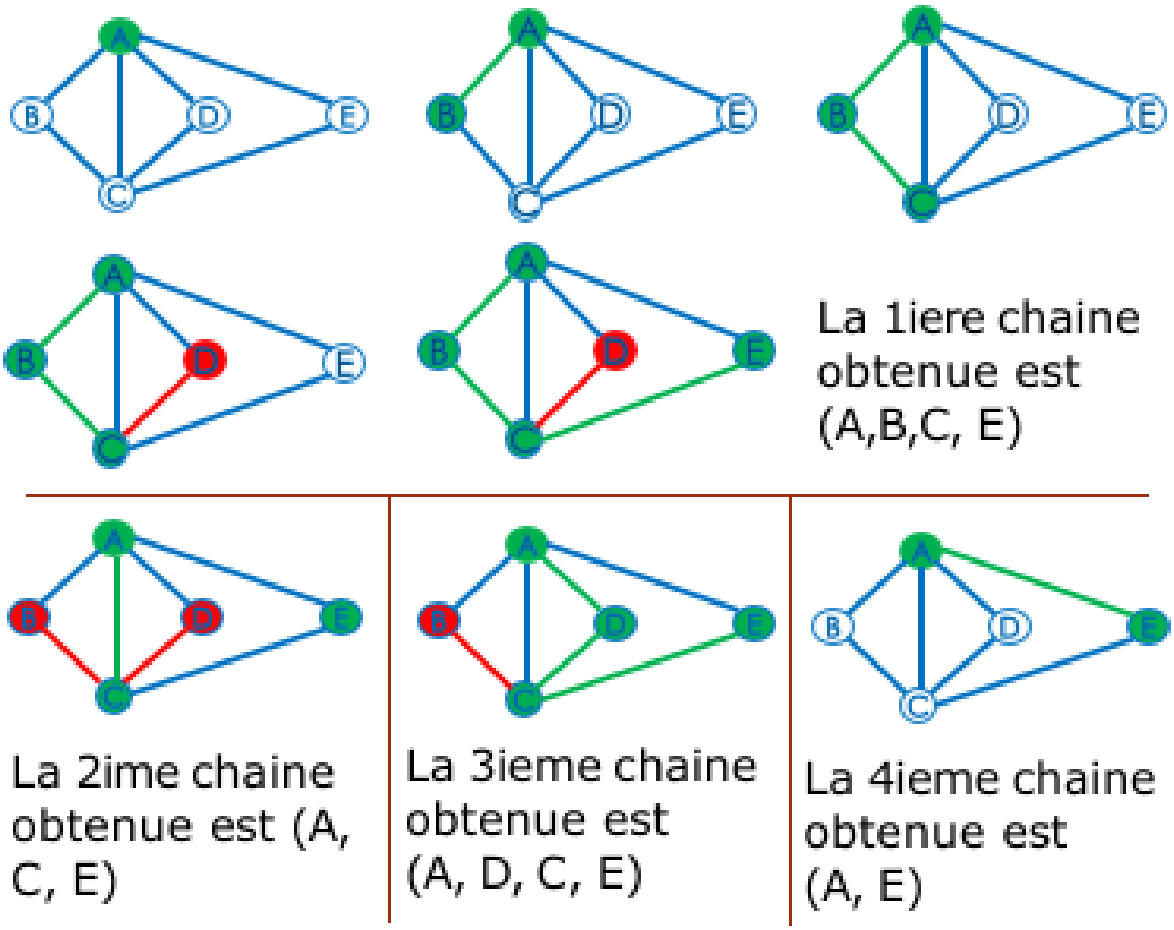
```
1 Marquer x comme visité
2 foreach arc (x,y) sortant de x do
3     if y n'est pas encore visité then
4         Marquer y
5         Ajouter y à C comme poursuivant de x
6         DFS(y)
```

Exemple :

Utiliser DFS pour trouver toutes les chaines liant le sommet A au sommet E du graphe suivant :



Solution :



4.4.2 Algorithme de Ford-Fulkerson

Algorithm 7: Algorithme de Ford-Fulkerson

Input: Graphe $G(X,U,W)$, source s, puits t
Output: Flot $F = \{f(u), u \in U\}$ et valeur totale f

foreach $u \in U$ **do**
  $f(u) \leftarrow 0$
$f \leftarrow 0$
while *il existe une chaîne améliorante* C **do**
  $\varepsilon_1 \leftarrow \min\{w(u) - f(u), u \in C^+\}$
  $\varepsilon_2 \leftarrow \min\{f(u), u \in C^-\}$
  $\varepsilon \leftarrow \min(\varepsilon_1, \varepsilon_2)$
  foreach $u \in C^+$ **do**
    $f(u) \leftarrow f(u) + \varepsilon$
  foreach $u \in C^-$ **do**
    $f(u) \leftarrow f(u) - \varepsilon$
  $f \leftarrow f + \varepsilon$

Remarque : La recherche des chaînes améliorantes peut être faite avec un parcours en profondeur (DFS).

Complexité :

— Complexité de Ford-Fulkerson : $\mathcal{O}(mf)$, où $m = |U|$ et f est le flot maximal.

4.4.3 Autres algorithmes de flot maximum

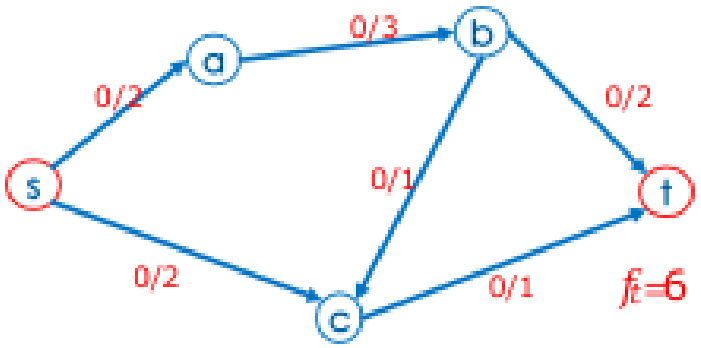
— **Algorithme d'Edmonds-Karp** : Complexité $\mathcal{O}(m^2 n)$, utilise BFS pour choisir le plus court chemin augmentant.

— **Algorithme de Goldberg-Tarjan** : Complexité $\mathcal{O}(n^3)$, utilise une approche par préflux.

4.5 Exercices

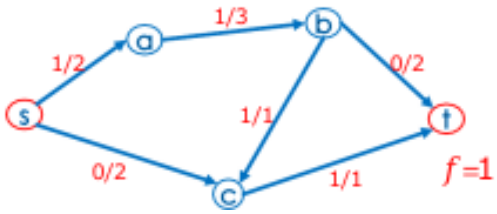
4.5.1 Exercice 01

Maximiser le flot dans le graphe suivant :

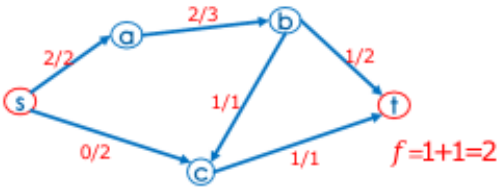


Solution :
Itération 1 :
On démarre par le sommet s et on marque les sommets en profondeur (DFS) pour construire une chaîne améliorante.

Sommet	s	a	b	c	t
Origine	-	s	a	b	c
ε	∞	2	2	1	ε=1



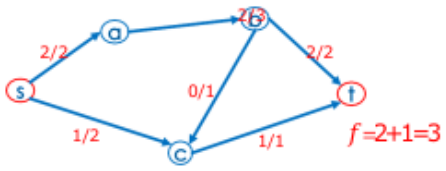
Itération 2 :
Sur la dernière chaine parcourue, le plus proche sommet ayant un successeur non visité est b.



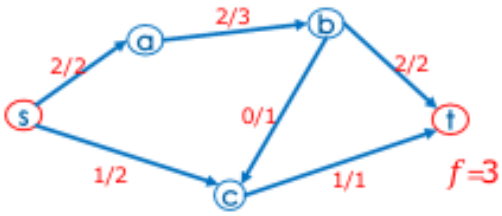
Sommet	s	a	b	t
Origine	-	s	a	b
ε	∞	1	1	ε=1

Itération 3 :
Sur la dernière chaine parcourue, le plus proche sommet ayant un successeur non visité est s.

Sommet	s	c	b	a	t
Origine	-	s	-c	-b	b
ε	∞	2	1	1	ε=1



Itération 4 :
Sur la dernière chaine parcourue, le plus proche sommet ayant un successeur non visité est c



Sommet	s	c
Origine	-	s
ε	∞	2

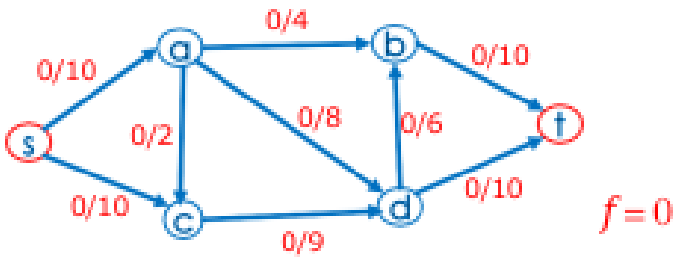
Il n y a plus de chaine améliorante, le flot obtenu est maximal.
Remarque :
On a la coupe $Y = \{s,a,b,c\}$ et $\overline{Y} = \{t\}$,
avec

$$w(Y,\overline{Y}) = w(b,t) + w(c,t) = 2 + 1 = 3.$$

Puisque $f = w(Y,\overline{Y})$, ceci confirme encore une fois que ce flot est maximal.

4.5.2 Exercice 02

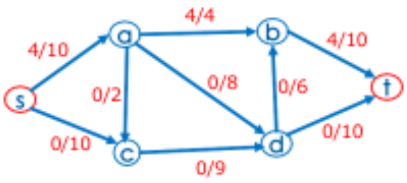
Calculer le flot max dans le réseau suivant :



Solution :

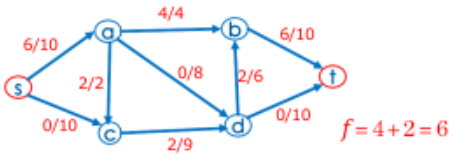
Itération 1 :

Sommet	s	a	b	t
Origine	-	s	a	b
ε	∞	10	4	$\varepsilon = 4$



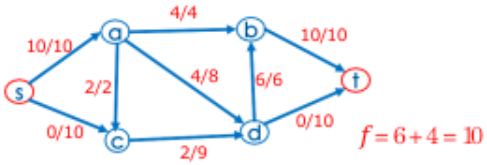
Itération 2 :

Sommet	s	a	c	d	b	t
Origine	-	s	a	c	d	b
ε	∞	6	2	2	2	$\varepsilon = 2$



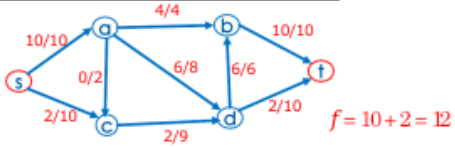
Itération 3 :

Sommet	s	a	d	b	t
Origine	-	s	a	d	b
ε	∞	4	4	4	ε=4



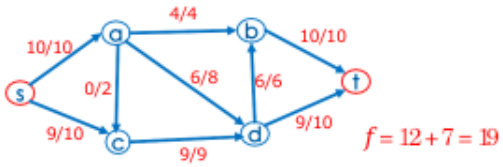
Itération 4 :

Sommet	s	c	a	d	t
Origine	-	s	c	a	d
ε	∞	10	2	2	ε=2



Itération 5 :

Sommet	s	c	d	t
Origine	-	s	c	d
ε	∞	8	7	ε=7



— Il n'y a plus de chaîne améliorante, ce flot est donc maximal.
D'ailleurs, la coupe $Y = \{s, a, c\}$ est de capacité $\boxed{19}$.

4.6 Conclusion

La résolution du problème du flot maximum repose sur l'algorithme de Ford-Fulkerson, qui consiste à chercher de manière itérative des chaînes améliorantes reliant la source au puits. Lorsque plus aucune chaîne de ce type n'existe, le flot obtenu est réalisable et maximal.

Ce flot est d'autant plus significatif qu'il atteint la capacité d'une coupe minimale du graphe, ce qui confirme l'optimalité de la solution selon le théorème de Ford-Fulkerson.

Pour les grands graphes ou les applications exigeant des performances accrues, des variantes optimisées comme les algorithmes d'Edmonds-Karp (utilisant BFS) ou de Goldberg-Tarjan (approche préflux) offrent des alternatives plus efficaces.

CHAPITRE

5

PROBLÈMES D'ORDONNANCEMENT

5.1 Introduction

Les problèmes d'ordonnancement sont courants en informatique et en ingénierie, car ils consistent à organiser des tâches ou des opérations en fonction de contraintes de temps et de ressources. L'objectif est de trouver une séquence optimale pour accomplir les tâches tout en respectant les contraintes. Ces problèmes se posent dans des domaines tels que la planification de production, la gestion de projet, et la répartition de ressources dans des systèmes informatiques.

5.2 Méthode PERT

La méthode **PERT** (Program Evaluation and Review Technique) est une technique développée en 1958 aux États-Unis. Elle a été conçue pour optimiser la gestion des tâches dans le cadre de projets de grande envergure, impliquant des milliers de personnes et un très grand nombre de tâches. Son principe repose sur le découpage du projet en tâches précises, l'estimation de la durée de chaque tâche, ainsi que la détermination des tâches précédentes nécessaires à leur exécution. Historiquement, la première application de la méthode PERT a permis de réduire la durée de réalisation d'un projet de 7 à 4 ans, démontrant ainsi son efficacité dans la planification et le pilotage de projets complexes.

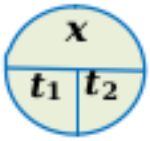
5.3 Le graphe PERT

- Représente les tâches et les dépendances entre elles
- Affiche une date de début au plus tôt et une date au plus tard pour chaque tâche
- Détermine un chemin critique qui conditionne la durée minimale du projet
- Met en évidence les tâches critiques qu'il faut impérativement réaliser dans les délais sous peine de retarder tout le projet
- Les tâches non critiques peuvent être retardées dans la limite de la marge calculée sans impacter la durée totale du projet

5.4 Conventions de la Méthode PERT

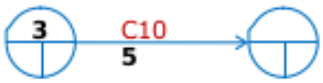
Une tâche est représentée par une étiquette sur une flèche. L'étiquette indique le code de la tâche et sa durée. Par exemple :

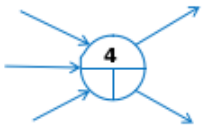
La tâche B ci-dessous dure 30 jours. On note cette durée par $w(B) = 30$.
Une **étape** représente le début ou la fin d'une tâche dans un graphe PERT. Elle est visualisée par un cercle divisé en trois portions :



— x : le numéro de l'étape (facultatif) ;
— t_1 : la date au plus tôt à laquelle l'étape peut être atteinte ;
— t_2 : la date au plus tard à laquelle l'étape peut être atteinte sans retarder le projet.

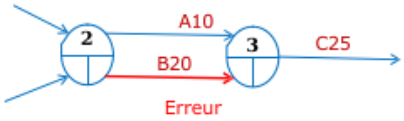
La tâche C10 démarre à l'étape 3 et se termine à l'étape 5.





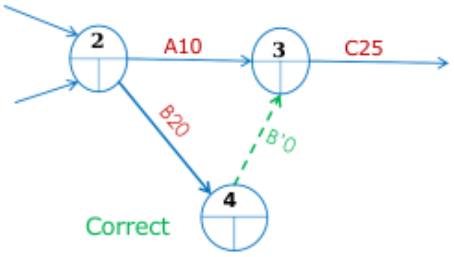
Une Etape peut recevoir et émettre plusieurs tâches.

Le graphe PERT démarre toujours par l'étape *D* (ou Début) et se termine par l'étape *F* (Fin).



Deux tâches, ou leurs successeurs, ne peuvent pas être raccordées aux mêmes étapes de départ et de fin.

Pour y remédier, il faut ajouter une tâche fictive en pointillés de durée 0.

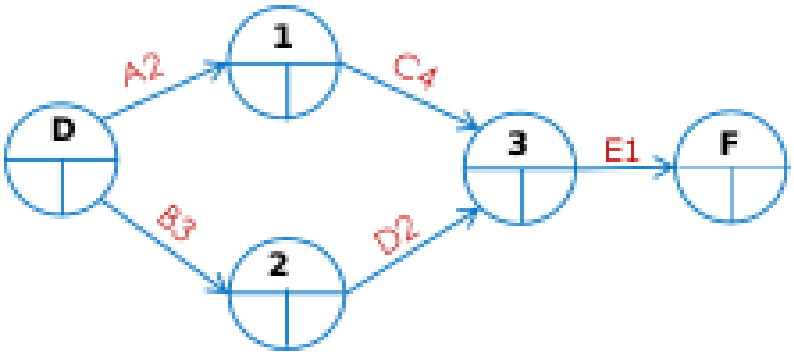


Exemple 1

Soit les tâches suivantes d'un projet :

Tâches	Prédécesseurs	Durées
A	-	2
B	-	3
C	A	4
D	B	2
E	C, D	1

Le graphe PERT correspondant est :



Sur le graphe, on constate bien que :

— Chaque tâche (arc) est précédée et suivie par une étape (sommet).

— Les tâches A et B n'ont pas de prédécesseurs.

— La tâche C est précédée par A.

— La tâche D est précédée par B.

— La tâche E est précédée par C et D.

Calcul des dates au plus tôt :

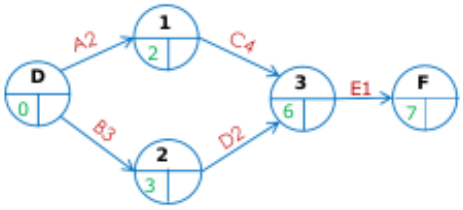
Pour l'étape D, on pose :

$$t_1(D) = 0$$

Pour une étape y, on a :

$$t_1(y) = \max_{x \in d^-(y)} (t_1(x) + w(x,y))$$

où $d^-(y)$ est l'ensemble des sommets prédécesseurs de y, et $w(x,y)$ est la durée de la tâche allant de x à y.



— **Étape 1 :** $0+2=2$

— **Étape 2 :** $0+3=3$

— **Étape 3 :** $\max\{2+4\ ;\ 3+2\}=6$

— **Étape F :** $6+1=7$

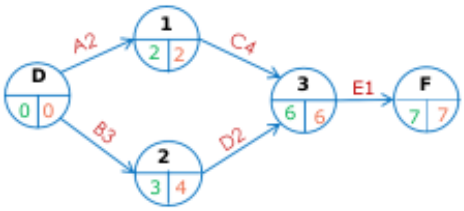
Calcul des dates au plus tard :

Pour l'étape F (finale), on pose :

$$t_2(F) = t_1(F) \quad \text{(ici, } F = 7\text{)}$$

Pour une étape x, on a :

$$t_2(x) = \min_{y \in d^+(x)} (t_2(y) - w(x,y))$$



— **Étape 3 :** $t_2 = 7 - 1 = 6$
— **Étape 2 :** $6 - 4 = 2$
— **Étape 1 :** $6 - 4 = 2$
— **Étape D :** $\min\{2-2\ ;\ 4-3\} = \min\{0\ ;\ 1\} = 0$

5.5 Construction d'un Réseau PERT

5.5.1 Exercice 01

Tableau des tâches :

Exemple d'un projet de 7 tâches :

Tâches	Prédécesseurs	Durées
A	-	5
B	D	3
C	D	6
D	-	2
E	B, H	3
F	C	1
G	A	2
H	C	2

Remarque : A priori, ce projet peut durer 24 jours (somme des durées).

Solution

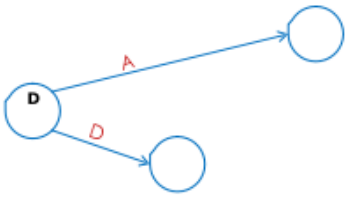
D'abord, on détermine les niveaux et les successeurs de chaque tâche.

- Les tâches de Niveau 1 sont celles qui n'ont pas de prédécesseurs.
- Les tâches de Niveau 2 sont celles dont les prédécesseurs sont ceux de Niveau 1.
- Les tâches de Niveau 3 sont celles dont les prédécesseurs sont ceux de Niveau 2.
- Etc.

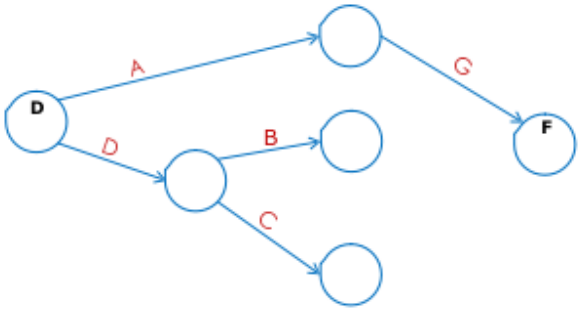
Tâches	Prédécesseurs	Niveaux	Successeurs
A	-	Niveau 1	G
B	D^1	Niveau 2	E
C	D^1	Niveau 2	F, H
D	-	Niveau 1	B, C
E	B^2, H^3	Niveau 4	-
F	C^2	Niveau 3	-
G	A^1	Niveau 2	-
H	C^2	Niveau 3	E

Construction progressive du graphe PERT

1. On commence par les tâches de **niveau 1**, soient A et D, qu'on raccorde au sommet Départ (D).

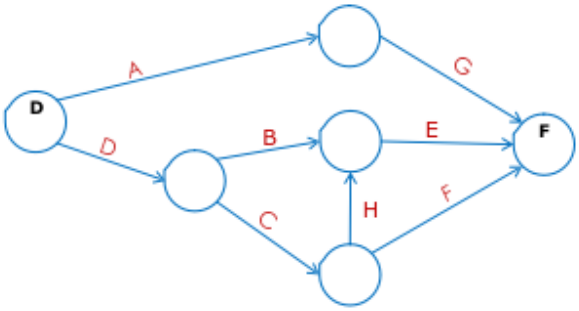


2. On ajoute les tâches de **niveau 2** : B, C et G. G est le successeur de A ; B et C sont les successeurs de D.

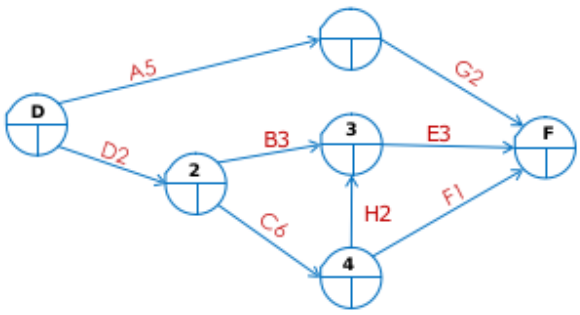


Attention : A chaque fois, on doit s'assurer que les tâches de niveau k-1 soient connectées à leurs successeurs de niveau k.

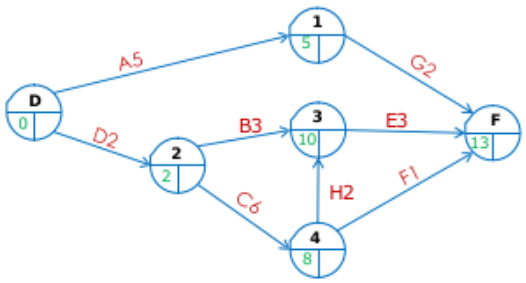
3. On ajoute les tâches de **niveau 3** : F et H comme successeurs de C. Ensuite E (niveau 4), successeur de B et H.



4. On affiche les durées de chaque tâche, et on numérote les étapes.

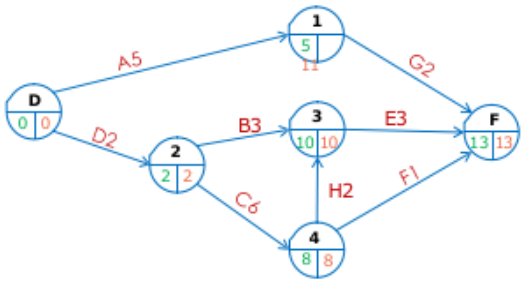


Calcul des dates au plutôt :



— **Étape D :** $t_1 = 0$
— **Étape 1 :** $0+5=5$
— **Étape 2 :** $0+2=2$
— **Étape 4 :** $2+6=8$
— **Étape 3 :**
$\max\{2+3\,;\,8+2\} = \max\{5\,;\,10\} = 10$
— **Étape F :**
$\max\{5+2\,;\,10+3\,;\,8+1\} =$
$\max\{7\,;\,13\,;\,9\} = 13$

Calcul des dates au plus tard (t_2) :



— **Étape F :** $t_2 = 13$
— **Étape 1 :** $13-2=11$
— **Étape 3 :** $13-3=10$
— **Étape 4 :**
$\min\{13-1\,;\,10-2\} = \min\{12\,;\,8\} = 8$
— **Étape 2 :**
$\min\{10-3\,;\,8-6\} = \min\{7\,;\,2\} = 2$
— **Étape D :**
$\min\{11-5\,;\,2-2\} = \min\{6\,;\,0\} = 0$

▶ Une **étape critique** x vérifie :

$$t_1(x) = t_2(x)$$

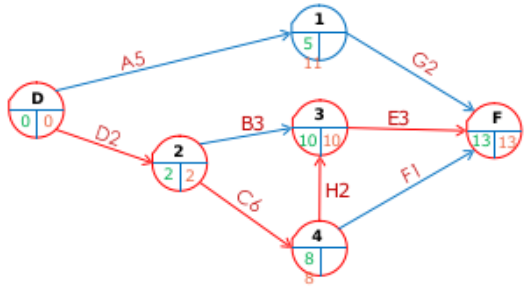
▶ La **marge** $m(u)$ d'une tâche u comprise entre les étapes x et y est donnée par :

$$m(u) = t_2(y) - w(u) - t_1(x)$$

où $w(u)$ est la durée de la tâche u.

▶ La tâche u peut commencer :
— au plus tôt à $t_1(x)$,
— au plus tard à $t_2(y) - w(u)$.

▶ Elle se termine :
— au plus tôt à $t_1(y)$,
— au plus tard à $t_2(x) + w(u)$.



Calcul des marges :

$m(A) = 11-5-0 = 6$
$m(D) = 2-2-0 = 0$
$m(G) = 13-2-5 = 6$
$m(B) = 10-3-2 = 5$
$m(C) = 8-6-2 = 0$
$m(H) = 10-2-8 = 0$
$m(E) = 13-3-10 = 0$
$m(F) = 13-1-8 = 4$

Interprétation :

— La durée minimale du projet est de **13 jours**.
— Le **chemin critique** est composé des tâches **D, C, H, E**.
— Un retard sur l'une de ces tâches entraînerait un retard du projet complet.
— Les tâches non critiques **A**, **G**, **B** et **F** peuvent être retardées respectivement de **6**, **6**, **5** et **4 jours** sans affecter la durée totale du projet.

5.5.2 Exercice 02

Utiliser la méthode PERT pour optimiser la gestion du projet suivant :

Tâches	Prédécesseurs	Durées
A	–	6
B	–	2
C	A	3
D	C	4
E	A	1
F	C, E	7
G	A, B	2
H	G	3
I	D, F	4

Somme des durée = 32 jours

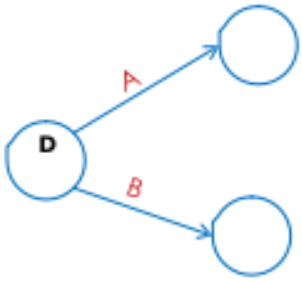
Solution

Tableau des niveaux et des successeurs :

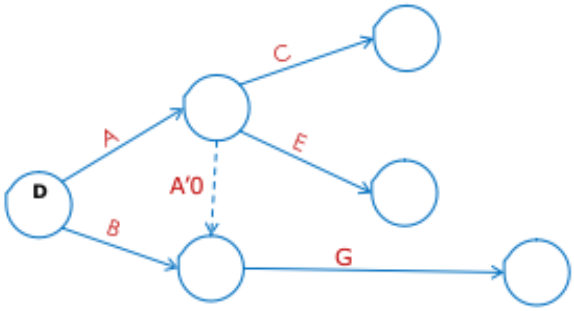
Tâches	Prédécesseurs	Niveaux	Successeurs
A	–	Niveau 1	C, E, G
B	–	Niveau 1	G
C	A^1	Niveau 2	D, F
D	C^2	Niveau 3	I
E	A^1	Niveau 2	F
F	C^2, E^2	Niveau 3	I
G	A^1, B^1	Niveau 2	H
H	G^2	Niveau 3	–
I	D^3, F^3	Niveau 4	–

Graphe PERT :

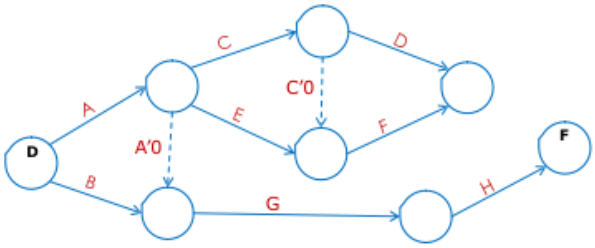
1. On commence par les tâches de niveau 1, c-à-d **A** et **B**.



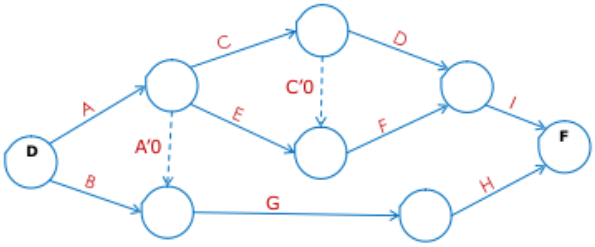
2. On passe aux tâches de niveau 2, c'est-à-dire **C**, **E** et **G**. La tâche **G** est le successeur de **A**, mais comme **A** est déjà reliée à une autre tâche, on les connecte à l'aide d'une **tâche fictive** notée $\mathbf{A}'_0$.



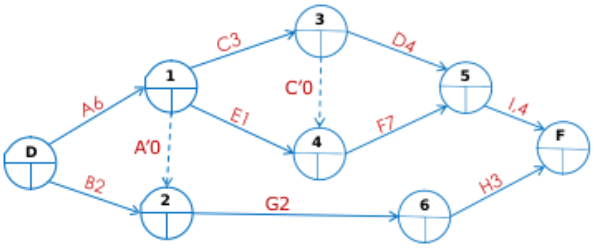
3. On passe aux tâches de niveau 3, c'est-à-dire **D**, **F** et **H**. La tâche **F** est le successeur de **C**, mais **C** étant déjà engagée ailleurs, on ajoute une **tâche fictive** notée $\mathbf{C}'_0$ pour représenter ce lien correctement dans le graphe.



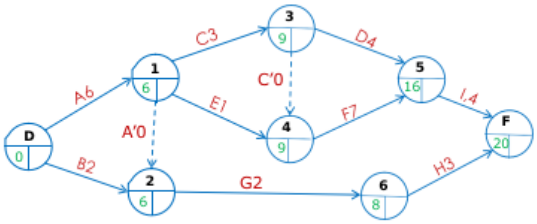
4. On passe à la tâche I de niveau 4, qui est le successeur de **D** et **F**.



5. On affiche les durées de chaque tâche, et on numérote les étapes.

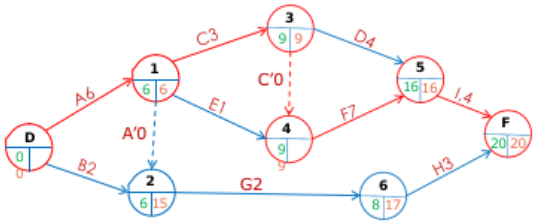


Calcul des dates au plutôt :



— **Étape D :** 0
— **Étape 1 :** $0+6=6$
— **Étape 2 :** $\max\{6+0\ ;\ 0+2\}=6$
— **Étape 3 :** $6+3=9$
— **Étape 6 :** $6+2=8$
— **Étape 4 :** $\max\{6+1\ ;\ 9+0\}=9$
— **Étape 5 :** $\max\{9+4\ ;\ 9+7\}=16$
— **Étape F :** $\max\{16+4\ ;\ 8+3\}=20$

Calcul des dates au plus tard



— **Étape F :** 20
— **Étape 6 :** $20-3=17$
— **Étape 5 :** $20-4=16$
— **Étape 4 :** $16-7=9$
— **Étape 2 :** $17-2=15$
— **Étape 3 :** $\min\{16-4\ ;\ 9-0\}=9$
— **Étape 1 :**
$\min\{9-3\ ;\ 9-1\ ;\ 15-0\}=6$
— **Étape D :** $\min\{6-6\ ;\ 15-2\}=0$

Calcul des marges :
$m(A)=6-6-0=0$
$m(B)=15-2-0=13$
$m(C)=9-3-6=0$
$m(D)=16-4-9=3$
$m(E)=9-1-6=2$
$m(F)=16-7-9=0$
$m(G)=17-2-6=9$
$m(H)=20-3-8=9$

$m(I) = 20 - 4 - 16 = 0$

Interprétation :

— La durée minimale du projet est de **20 jours**.

— Le **chemin critique** est composé des tâches : A, C, F, I.

— Les **tâches non critiques** sont :

— B : marge = 13 jours

— D : marge = 3 jours

— E : marge = 2 jours

— G : marge = 9 jours

— H : marge = 9 jours

5.6 Diagramme de Gantt (1910)

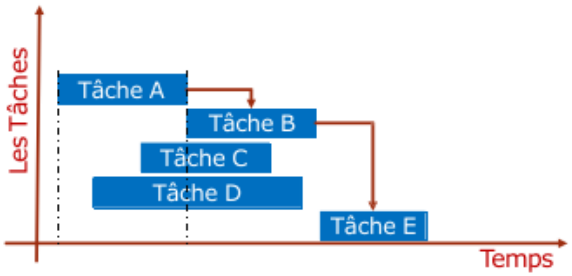
Le diagramme de Gantt permet de visualiser sur un plan à deux dimensions le déroulement des tâches d'un projet.

– Cette méthode suppose connus :

— les dates de début de chaque tâche,

— les durées de chaque tâche.

– Parfois, on peut ajouter les dépendances entre les tâches.



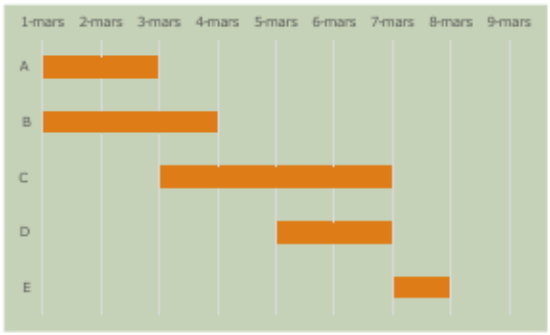
Sur ce diagramme, la tâche B ne peut commencer avant la fin de A, et E commence après la terminaison de B.

5.6.1 Exercice

Soit un projet dont les dates de débuts et les durées sont comme suit :

Tâche	Début	Durée (jours)	Fin
A	01/03/2017	2	03/03/2017
B	01/03/2017	3	04/03/2017
C	03/03/2017	4	07/03/2017
D	05/03/2017	2	07/03/2017
E	07/03/2017	1	08/03/2017

Le diagramme de Gantt correspondant est :



5.7 Conclusion

Les problèmes d'ordonnancement jouent un rôle central dans la planification de projets complexes où les ressources et le temps sont limités. Grâce à des méthodes comme **PERT** et les outils visuels tels que le **diagramme de Gantt**, il est possible de représenter, d'analyser et d'optimiser la séquence des tâches nécessaires à la réalisation d'un projet.

La méthode PERT, en particulier, permet d'identifier les tâches critiques, de calculer les marges de manœuvre et de déterminer la durée minimale du projet. Elle offre ainsi une aide précieuse à la prise de décision, à la gestion des délais, et à l'allocation optimale des ressources.

En somme, la maîtrise des techniques d'ordonnancement permet non seulement d'améliorer l'efficacité opérationnelle, mais aussi de réduire les risques liés aux retards et aux goulots d'étranglement dans la gestion de projets complexes.