

HOUSSEM SABRI



Graphes et Optimisation

2^{ème} année

LISI – LSI

Institut Supérieur des Sciences Appliquées et de Technologie de Sousse
A.U. 2023-2024

28 novembre 2023

Table des matières

1	Les graphes	3
1.1	Définitions	3
1.1.1	Graphes non orientés	3
1.1.2	Graphes orientés	6
1.2	Opérations et quelques types de graphes	7
1.3	Représentation d'un graphe	9
1.3.1	Listes d'adjacences (ou de succession)	9
1.3.2	Matrices d'adjacence	10
1.3.3	Matrice d'incidence	10
2	La connexité	11
2.1	Les chaînes et les cycles	11
2.2	Graphe connexe, eulérien, hamiltonien	13
2.3	Arbres	17
2.3.1	Codage et décodage d'un arbre – Prüfer code	18
2.3.2	Arbre couvrant de poids minimum – ACM	19
3	Les algorithmes de base	20
3.1	Parcours en profondeur (DFS) / largeur (BFS)	20
3.2	Recherche des chemins	22
3.2.1	Matrice d'adjacence et nombre de chemins	22
3.2.2	Plus court chemin et l'algorithme de Dijkstra	23
3.3	Problèmes d'ordonnancement et chemin critique	25
3.3.1	Algorithmes de tri topologique et de chemin critique	25
4	Programmation linéaire	27
4.1	La méthode simplexe	28
4.1.1	Solution de base admissible	28
4.1.2	L'algorithme du simplexe avec la méthode des tableaux	29
4.1.3	Initialisation - Phase I du simplexe	30
4.2	Dualité	32

CHAPITRE 1

Les graphes

L'exemple classique pour introduire la théorie des graphes est le problème des ponts de Königsberg, étudié par Euler en 1736. Quel itinéraire doit-on suivre pour traverser chaque pont exactement une fois? Une telle promenade n'existe pas, et c'est Euler qui donna la solution de ce problème en caractérisant les graphes que l'on appelle aujourd'hui **eulériens**.

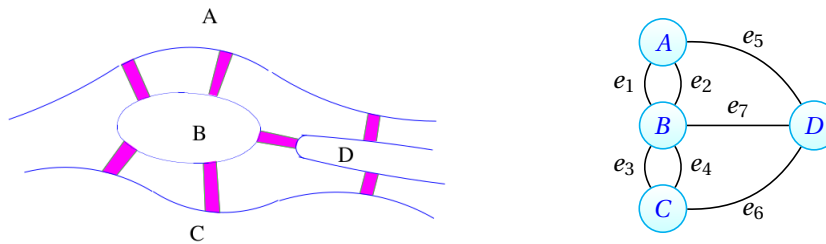


FIGURE 1.1 – Les ponts de Königsberg

Les graphes introduisent une simplification : les régions se transforment en des "points" (aussi appelés **noeuds** ou **sommet**), les ponts se transforment en des lignes (aussi appelées **arêtes** ou **arcs**), et ce principe s'applique dès lors qu'une relation existe entre des objets. Les graphes relèvent de la combinatoire plutôt que de la géométrie (les intersections, les longueurs, et la forme des arêtes n'ont aucune importance, sauf en ce qui concerne la lisibilité de la représentation).

1.1 Définitions

Dans la suite, il est toujours supposé que les graphes sont finis, sauf indication contraire.

1.1.1 Graphes non orientés

Définition 1.1: Graphe

Un graphe G est un couple d'ensembles (V, E) où l'ensemble E est une partie de $V^{\{2\}}$ des paires non ordonnées de V . V est l'ensemble des noeuds (Vertices en anglais) et E c'est l'ensemble des arêtes de G (Edges en anglais). Lorsque $e = \{x, y\}$ est une arête, on dit que x, y sont les extrémités de cette arête, que e est incidente en x et en y . On joint x et y par un trait et on dit que x, y sont adjacents (ou voisins). On note aussi $e = xy$ ou encore $e = (x, y)$ (**attention**, dans ce cas : $(x, y) = (y, x)$).

Si G est un graphe alors, on note par $V(G)$ l'ensemble de noeuds et par $E(G)$ l'ensemble des arcs.

Remarque 1.

- Une arête $e = \{x, x\}$ s'appelle **boucle**.
- Dans un graphe, il peut exister des arêtes parallèles (on parle donc de **multigraphe**). Dans l'exemple des ponts on a : $e_1 = e_2 = (A, B)$.

- Un graphe est dit **simple** s'il ne comporte ni arêtes multiples ni de boucles, c'est-à-dire chaque paire de nœuds distincts est reliée par au plus une arête, ou encore $E(G) \subseteq \mathcal{P}_2(G)$.

Dans la plupart des cas, les graphes sont considérés comme simples par défaut.

Définition 1.2: ordre – taille – degré

Soit $G = (V, E)$ un graphe.

1. On appelle **ordre** d'un graphe noté $v(G)$ le nombre de ses sommets :

$$v(G) = |V|$$

2. On appelle **taille** d'un graphe noté $e(G)$ le nombre de ses arêtes :

$$e(G) = |E|$$

3. Le **degré** (ou valence) d'un sommet x noté $d(x)$ est le nombre d'arêtes contenant x . (Attention! Une boucle sur un sommet compte double).

Si on note par $\Gamma(x)$ l'ensemble des voisins de x , i.e.

$$\Gamma(x) = \{y \in V : xy \in E\}$$

alors, si G est **simple**, on a :

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

Exemple 1.1 (Graphe complet)

Un graphe d'ordre n est dit **complet** noté K_n si chaque sommet du graphe est relié directement à tous les autres sommets, i.e. $E(K_n) = \mathcal{P}_2(V(K_n))$.

Représenter graphiquement K_3, K_4, K_5 . Calculer la taille de K_3, K_4, \dots, K_n .

Exemple 1.2 (Graphe biparti)

Un graphe $G = (V, E)$ est dit **biparti** si son ensemble de sommets $V = V_1 \cup V_2$, i.e. peut être divisé en deux sous-ensembles disjoints V_1 et V_2 tels que chaque arête ait une extrémité dans V_1 et l'autre dans V_2 .

Si $\forall x \in V_1, \forall y \in V_2; xy \in E$, alors G est dit **biparti complet** noté $K_{n,m}$ où $n = |V_1|$ et $m = |V_2|$.

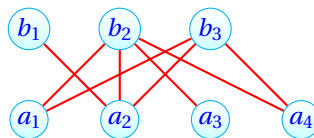


FIGURE 1.2 – Un graphe biparti

Théorème 1.1 Lemme des poignées de main – Handshaking lemma

Soit G un graphe alors, on a :

$$\sum_{x \in V(G)} d(x) = 2e(G)$$

Dans ce théorème $d(x)$ est le nombre de personnes à qui x serre la main et $e(G)$ le nombre total de poignées de mains.

Preuve.

- **Méthode simple** Il y en a $d(x)$ arêtes en un sommet x fixé. Pour tout les sommets cela fait donc $\sum_{x \in V(G)} d(x)$. Mais avec cette manière chaque arête est comptée deux fois, une fois à chacune de ses extrémités, d'où le résultat en divisant par 2.

- **Preuve par double dénombrement** On pose

$$Z = \{(x, e) \in V \times E : x \in e\} \subseteq V \times E$$

et les deux projections :

$$\begin{aligned} p: Z &\rightarrow V \quad \text{et} \quad q: Z \rightarrow E \\ (x, e) &\mapsto x \quad \quad (x, e) \mapsto e \end{aligned}$$

L'idée est de calculer $|Z|$ de deux manières différentes. D'abord, on rappelle que si $f: A \rightarrow B$ est une fonction quelconque, alors :

$$\sum_{b \in B} |f^{-1}[b]| = |A|$$

où $f^{-1}[b]$ désigne l'image réciproque de $\{b\}$ par f (pourquoi $\dot{\bigcup}_{b \in B} f^{-1}[b]$ est une union disjointe?).

Ainsi on obtient :

$$|Z| = \sum_{x \in V} |p^{-1}[x]| = \sum_{e \in E} |q^{-1}[e]|$$

Or, $|p^{-1}[x]| = d(x)$ et $|q^{-1}[e]| = |\{x \in V : x \in e\}| = 2$. Donc :

$$\sum_{x \in V} d(x) = \sum_{e \in E} 2 = 2|E|$$

■

Remarque 2.

- Un sommet de degré 0 est appelé **sommet isolé**.
- Un graphe tel que : $\forall x \in V, d(x) = k$ est dit graphe **k -régulier**.
- Le graphe complet K_n est un graphe $(n-1)$ -régulier.

Exercice 1.1

Montrer que dans un graphe le nombre de sommets d'ordre impairs est toujours pair. En particulier un graphe k -régulier avec k impaire est un graphe d'ordre pair.

Voici quelques exemples comme application :

1. Est-il possible de relier 15 ordinateurs en réseau de sorte que chacun soit relié à exactement 3 autres?
2. Une ligue de football comprenant 11 clubs organise un tournoi. Pour gagner du temps on décide que chaque équipe ne jouera que la moitié des matches possibles. Comment organiser le tournoi?

Proposition 1.1

Soit $G = (V, E)$ un graphe (simple) d'ordre $n \geq 2$. Alors :

$$\exists u, v \in V, u \neq v : d(u) = d(v)$$

Preuve. Supposons le contraire, c'est-à-dire $\forall u, v \in V : d(u) \neq d(v)$. Or dans un graphe (simple) on a toujours $d(x) \in \{0, 1, \dots, n-1\}$. Dans ce cas, puisque tous les n sommets ont des degrés distincts, alors il existe une bijection

$$\{d(x) : x \in V\} \longleftrightarrow \{0, 1, \dots, n-1\}$$

Donc, il existe un sommet u avec un degré 0, et un sommet v avec degré $n-1$, donc v est connecté à tous les autres sommets du graphe, ce qui signifie que u ne peut pas être un sommet isolé, d'où une contradiction. ■

1.1.2 Graphes orientés

Les arêtes d'un graphe peuvent être orientées, auquel cas une flèche indique la direction. De tels graphes sont appelés des graphes orientés ou des **digraphes**, et une arête (a, b) dans un tel graphe indique que cette arête est dirigée de sommet a vers b , i.e. $(a, b) = \overrightarrow{ab} \neq (b, a) = \overrightarrow{ba}$. D'où la définition :

Définition 1.3: Graphe orienté - digraphe

Un graphe orienté G (ou \vec{G}) est un couple d'ensembles (V, E) où l'ensemble E est une partie de V^2 des paires ordonnées de V . E est l'ensemble des arêtes (ou flèches, arcs) de G . Lorsque $e = (x, y)$ est une arête, on dit que x est l'**origine** (ou sommet initiale) et y l'**extrémité** (ou sommet terminale) de e , que e est sortant en x et incident en y , et que y est un successeur de x tandis que x est un prédécesseur de y . On note l'arête e par :

$$e = (x, y) = \overrightarrow{xy} = x \rightarrow y$$

De la même façon, une arête $x \rightarrow x$ est une boucle. On peut toujours supprimer les boucles pour obtenir des digraphes simples. Voici l'exemple d'un digraphe $G = (V, E)$ d'une fonction $f: E \rightarrow E$ où (x, y) est un arc si et seulement si $y = f(x)$. On l'appelle digraphe fonctionnelle.

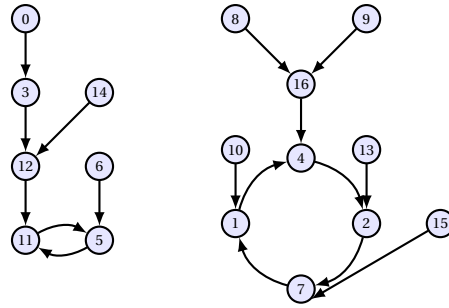


FIGURE 1.3 – Le digraphe fonctionnelle \mathcal{G}_f de $x \mapsto (x^2 + 3) \bmod 17$.

Comme dans les graphes on peut définir les degrés (entrant, sortant) d'un sommet dans un digraphe.

Définition 1.4: indegree - outdegree

Soit $G = (V, E)$ un digraphe. Pour $v \in V$, on note par :

$d^+(v)$ le degré extérieur (sortant - outdegree) du sommet v , c'est-à-dire le nombre d'arcs ayant v comme extrémité initiale.

$$d^+(v) = |\{v \rightarrow \bullet\}|$$

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

$$d^-(v) = |\{\bullet \rightarrow v\}|$$

On définit le degré : $d(v) = d^-(v) + d^+(v)$.

Un sommet de degré entrant non nul et de degré sortant nul est appelé **puits**, tandis qu'un sommet de degré entrant nul et de degré sortant non nul est appelé **source** ou **racine**.

Dans un graphe orienté, les voisins d'un sommet v sont soit des successeurs ou des prédécesseurs. On les notes par :

$$\Gamma^+(x) = \text{Succ}(x) = \{y \in V : x \rightarrow y \in E\} \quad \text{et} \quad \Gamma^-(x) = \text{Pred}(x) = \{y \in V : y \rightarrow x \in E\}$$

Si le digraphe est **simple** (ni boucle ni arcs parallèles) on a :

$$d^-(v) = |\Gamma^-(v)| \quad \text{et} \quad d^+(v) = |\Gamma^+(v)|$$

Proposition 1.2

Soit G un digraphe, alors :

$$\sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) = |E|$$

Preuve. On a

$$\begin{aligned} E &= \bigcup_{v \in V} \{\bullet \rightarrow v\} = \bigcup_{v \in V} \{v \rightarrow \bullet\} \\ \Rightarrow |E| &= \sum_{v \in V} |\{\bullet \rightarrow v\}| = \sum_{v \in V} |\{v \rightarrow \bullet\}| \Rightarrow |E| = \sum_{v \in V} d^-(v) = \sum_{v \in V} d^+(v) \end{aligned}$$

■

1.2 Opérations et quelques types de graphes

Une opération sur les graphes permet de construire un nouveau graphe comme résultat de l'opération. Les opérations courantes sur les graphes sont la suppression, la valuation, l'union, l'intersection et le produit cartésien.

Graphe valué

Tout d'abord, on peut vouloir attribuer des valeurs aux arcs ou arêtes pour tenir compte de contraintes : distance, coût ...

Définition 1.5: (di)graphe valué

Un (di)graphe valué $G = (V, E, w)$ est un (di)graphe muni d'une application $w: E \rightarrow \mathbb{R}$. L'application w est appelée application poids ou coût. On indique sur chaque arête son poids (ou son coût). On note $w(x, y)$ au lieu de $w((x, y))$. On convient (même que ceci n'a pas de sens!) que $w(x, y) = +\infty$ si $(x, y) \notin E$.

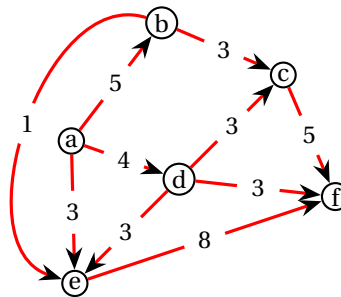


FIGURE 1.4 – Un digraphe valué

Python code :

Implémenter le digraphe valué précédent avec python. **Indice** – use chatGPT : how to build a weighted digraph with python

Sous graphes

Soit $G = (V, E)$ un graphe (orienté ou non). Un sous graphe de G est un graphe de la forme $G' = (V', E')$ où $V' \subseteq V$, $E' \subseteq E$ tels que toute arête de E' a ses extrémités dans V' (ou encore G' est aussi un graphe). On a :

- **Sous-graphe induit** Un sous-graphe $G' = (V', E')$ de $G = (V, E)$ est appelé un sous-graphe **induit** de G si :

$$\forall x, y \in V'; (x, y) \in E' \iff (x, y) \in E$$

En d'autres termes, deux sommets de G' qui sont connectés dans G , sont également connectés dans G' .

- **Sous-graphe couvrant** Un sous-graphe couvrant G' de G contient tous les sommets de G , i.e. $V' = V$.
- **$G - v$** Supprimer un sommet v d'un graphe G consiste à supprimer v et toutes ses arêtes incidentes. Le sous-graphe obtenu de cette opération est noté $G - v$, i.e.

$$V' = V - \{v\} \quad \text{et} \quad E' = E - \{(v, y) : (v, y) \in E\}$$

De manière similaire, une arête e peut être supprimée (resp. ajoutée) d'un graphe G , et le sous-graphe résultant est représenté par $G - e$ (resp. $G + e$). Notez que la suppression d'une arête ne retire pas de sommets du graphe.

- **$G - V' = G[V']$** On supprimer tous les sommets $v \in V'$ du graphe G avec les arêtes incidentes. Le sous-graphe obtenu de cette opération est noté $G[V']$, i.e.

$$V(G[V']) = V - V' \quad \text{et} \quad E(G[V']) = E - \{(v, y) \in E : v \in V'\}$$

- **Clique** Une clique C d'un graphe G est un sous-ensemble des sommets de G dont le sous-graphe induit $G' = (C, E')$ est un graphe complet, en d'autres termes, deux sommets quelconques de la clique C sont toujours adjacents.
- **Stable** Un stable – appelé aussi ensemble indépendant (independent set en anglais) est un ensemble de sommets deux à deux non adjacents. Donc le sous-graphe induit est sans arcs.

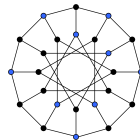


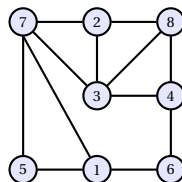
FIGURE 1.5 – L'ensemble des sommets en bleu dans ce graphe est un stable maximal du graphe.

Note :-

Le problème décisionnel de la clique (c'est à dire existe t-il une clique de taille k) est **NP-complet**. De même la recherche d'un stable de taille maximum dans un graphe est un problème NP-complet et difficile à approximer.

Exercice 1.2

Trouver un plus grand stable et une plus grande clique du graphe suivant :



Indice : vérifier le résultat avec `networkx.find_cliques(G)` et `.maximum_independent_set(G)` de python.

Opérations binaires

Union L'union de deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ est un graphe $F = G \cup H = (V_F, E_F)$ dans lequel

$$V_F = V_G \cup V_H \quad \text{et} \quad E_F = E_G \cup E_H$$

Intersection L'intersection de deux graphes $G = (V_G, E_G)$ et $H = (V_H, E_H)$ est un graphe $F = G \cap H = (V_F, E_F)$ dans lequel

$$V_F = V_G \cap V_H \quad \text{et} \quad E_F = E_G \cap E_H$$

1.3 Représentation d'un graphe

Il est important de savoir comment représenter les graphes au sein d'un ordinateur. Il ya plusieurs méthodes selon la nature des traitements que l'on souhaite appliquer au graphe considéré.

1.3.1 Listes d'adjacences (ou de succession)

Soit G un (di)graphe. On suppose que les sommets de G sont numérotés de S_1 à S_n . La représentation par listes d'adjacence de G consiste en un tableau (ou des listes chaînées) tel que la ligne $i = 1 \dots n$ correspond au sommet S_i et comporte **la liste** des voisins/successeurs (ou des prédécesseurs) de ce sommet notée $T[S_i]$. Les sommets dans chaque liste d'adjacence sont généralement listés selon un ordre arbitraire.

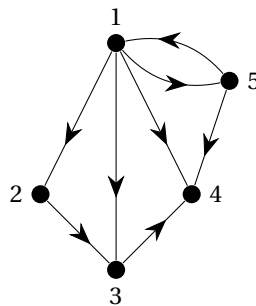


FIGURE 1.6 – Un digraphe

La liste d'adjacence de digraphe 1.6 est donc :

1	2	3	4	5
2	3			
3	4			
4	—			
5	1	4		

1.3.2 Matrices d'adjacence

Définition 1.6: Mat. Adjacence

Soit $G = (V, E)$ un graphe (orienté) d'ordre n . On peut numéroter $V = \{1, 2, \dots, n\}$. La **matrice d'adjacence** de G est la matrice carrée $A = (a_{ij})$ d'ordre n , définie par :

$$a_{ij} = \begin{cases} 1 & \text{si } (i, j) \in E; \\ 0 & \text{sinon.} \end{cases}$$

Remarque 3.

1. Si le graphe G est non orienté et simple, alors la matrice A est symétrique avec des 0 sur la diagonale. Pour un digraphe simple, on a toujours des 0 sur la diagonale, mais la matrice n'est pas symétrique en général.
2. Pour les multigraphes, il peut y avoir des termes diagonaux (pour les boucles) et s'il y a plusieurs arêtes de i à j , le coefficient de la matrice a_{ij} est le nombre d'arêtes de i à j .
3. Pour un (di)graphe valué $G = (V, E, w)$, on pose :

$$a_{ij} = \begin{cases} w(i, j) & \text{si } (i, j) \in E; \\ \infty & \text{sinon.} \end{cases}$$

La matrice d'adjacence de la figure 1.4 est :

$$\begin{pmatrix} \infty & 5 & \infty & 4 & 3 & \infty \\ \infty & \infty & 3 & \infty & 1 & \infty \\ \infty & \infty & \infty & \infty & \infty & 5 \\ \infty & \infty & 3 & \infty & 3 & 3 \\ \infty & \infty & \infty & \infty & \infty & 8 \\ \infty & \infty & \infty & \infty & \infty & \infty \end{pmatrix}$$

1.3.3 Matrice d'incidence

Définition 1.7: Mat. Incidence

Soit $G = (V, E)$ un graphe (simple) d'ordre n et de taille m . On pose $V = \{v_1, v_2, \dots, v_n\}$ et $E = \{e_1, e_2, \dots, e_m\}$. La **matrice d'incidence** de G est la matrice de type $n \times m$ $B = (b_{ij})$, définie par :

• Si G orienté

$$b_{ij} = \begin{cases} 1 & \text{si } e_j = v_i \rightarrow \bullet \text{ (i.e. } v_i \text{ est l'extrémité initiale de } e_j); \\ -1 & \text{si } e_j = \bullet \rightarrow v_i \text{ (i.e. } v_i \text{ est l'extrémité terminale de } e_j); \\ 0 & \text{sinon.} \end{cases}$$

• Si G non-orienté

$$b_{ij} = \begin{cases} 1 & \text{si } v_i \text{ est une extrémité de } e_j; \\ 0 & \text{sinon.} \end{cases}$$

Exercice 1.3

1. Donner la matrice d'incidence du graphe 1.6.
2. Montrer le théorème 1.1 avec la matrice d'incidence.

CHAPITRE 2

La connexité

Il est important de savoir si on peut suivre un chemin d'un sommet donné à un autre dans un graphe donné. Cette propriété (de connexité) doit être maintenue dans différents types de réseaux.

2.1 Les chaînes et les cycles

Définition 2.1: (chaîne – cycle) (chemin – circuit)

• Graphe non orienté

1. Une **chaîne (walk)** reliant x à y , notée $\mu(x, y)$, est définie par une suite finie d'arêtes consécutives, reliant x à y , ou encore une **séquence de sommets** (puisque non multigraphe)

$$\mu(x, y) = \langle v_0, v_1, \dots, v_k \rangle$$

tels que $v_0 = x, v_k = y$ et $(v_{i-1}, v_i) \in E(G)$ pour tout $i = 1 \dots k$. Si $x = y$ la chaîne est dite **fermée**. La **longueur du chaîne** $l(\mu(x, y))$ est le nombre d'arcs dans la chaîne, c'est-à-dire k .

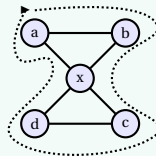
2. Une chaîne **élémentaire (a path)** si tous les sommets sont distincts. Une chaîne **simple (a simple walk, a trail)** est une chaîne ne passant pas deux fois par une même arête, c'est-à-dire dont toutes les arêtes sont distinctes. Remarquer que : élémentaire \implies simple.
3. Un **cycle** $v_0 v_1 \dots v_n$ de longueur $n \geq 3$ est une chaîne simple fermée ($v_0 = v_n$) tel que les sommets v_0, v_1, \dots, v_{n-1} sont tous distincts. Un graphe sans cycle est dit **acyclique**.

• Graphe orienté

Dans un graphe orienté, on parlera de **chemin** au lieu de chaîne, et de **circuit** au lieu de cycle.

Exemple 2.1

$abxcdxa$ est une chaîne simple et fermée (non cycle!) de longueur 6 (mais, c'est un graphe eulérien - voir plus loin). Y a-t-il un cycle de longueur 3, 4?



Remarque 4.

1. On peut pivoter une chaîne fermée (l'orienter de manière arbitraire), i.e.

$$\langle v_0, v_1, v_2, \dots, v_0 \rangle = \langle v_k, v_{k+1}, \dots, v_0, v_1, \dots, v_k \rangle$$

2. Soient $x, y \in V$. La **distance** de x à y est définie par :

$$d(x, y) = \begin{cases} k & \text{si le plus court chemin/chaîne de } x \text{ vers } y \text{ est de longueur } k \\ \infty & \text{sinon} \end{cases}$$

On convient que $d(x, x) = 0$.

3. Le **diamètre** du graphe la plus grande distance entre deux sommets.

L'observation suivante, bien que très facile à prouver, sera utile.

Théorème 2.1

S'il existe une chaîne du sommet y au sommet z dans le graphe G , où $y \neq z$, alors il existe une chaîne élémentaire (**path**) dans G avec pour premier sommet y et pour dernier sommet z .

Preuve. Soit la chaîne

$$W_1 = \mu(y, z) = x_0 x_1 \dots x_n; \quad \text{avec } x_0 = y \text{ et } x_n = z.$$

Si les sommets x_0, x_1, \dots, x_n sont tous distincts, alors W_1 est une chaîne élémentaire, et nous avons terminé. Sinon, il existe $x_i = x_j$ ($i < j$) et on écrit :

$$W_2 = x_0 x_1 \dots x_i x_{j+1} \dots x_n.$$

Alors, $l(W_2) < l(W_1)$. Si W_2 ne contient aucun sommet répété, alors c'est la chaîne recherchée. Sinon, par la même procédure, il existe W_3 (yz -chaîne) tel que

$$l(W_3) < l(W_2) < l(W_1).$$

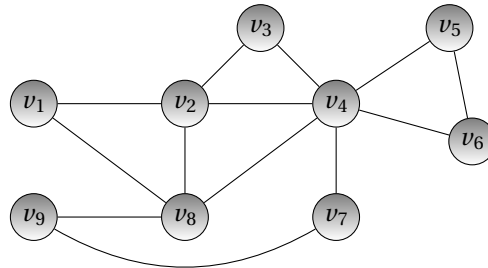
Ce processus doit s'arrêter à un certain stade, car chaque chaîne est plus courte que la précédente et la longueur ne peut jamais être inférieure à 1

$$1 < \dots < l(W_k) < \dots < l(W_3) < l(W_2) < l(W_1).$$

Ainsi, pour un certain k , W_k ne peut pas être réduit en longueur. Donc, la chaîne W_k ne contient aucun sommet répété et est donc une yz -chaîne élémentaire (path). ■

Exercice 2.1

Dans le graphe suivant, construire un path à partir de la chaîne $W = v_2 v_3 v_4 v_5 v_6 v_4 v_2 v_8 v_1$



Calculer $d(v_1, v_7)$ et le diamètre de G .

🐍 **python :** `>> G = nx.Graph([(1, 2), (2, 3), (2, 4), ...]) >> nx.diameter(G)`

Les **cycles** fournissent la caractérisation suivante pour les **graphes bipartis**.

Théorème 2.2

Un graphe est biparti si et seulement s'il ne contient aucun cycle de longueur impaire.

Preuve.

⇒ On suppose que $G = U \dot{\cup} V$ est un graphe biparti. Soit $W = \langle x_1, x_2, \dots, x_{2k+1}, x_1 \rangle$ un cycle de longueur impaire $(2k+1)$. Si $x_1 \in U$, alors

$$x_1 \in U \Rightarrow x_2 \in V \Rightarrow x_3 \in U \Rightarrow \dots \Rightarrow x_{2k+1} \in U \Rightarrow x_1 \in V \quad \blacktriangle$$

absurde, d'où le résultat.

⇐ On suppose que G est connexe (car l'union des composantes connexes bipartis est un graphe biparti). Soit x un sommet de G et on pose :

$$U = \{y \in V(G) : d(x, y) \text{ est paire}\} \quad \text{et} \quad V = \{z \in V(G) : d(x, z) \text{ est impaire}\}$$

S'il existe $y_1, y_2 \in U$ tel que $y_1 y_2 \in E(G)$, alors on a :

$W_1 = \langle x, \dots, y_1 \rangle$ est une chaîne de longueur paire $= 2k_1$

$W_2 = \langle x, \dots, y_2 \rangle$ est une chaîne de longueur paire $= 2k_2$

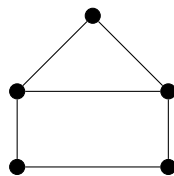
⇒ $W_3 = \langle x, \dots, y_1, y_2, \dots, x \rangle$ est un cycle de longueur impaire $= 2k_1 + 2k_2 + 1 \quad \blacktriangle$

Donc $G = U \dot{\cup} V$ est un graphe biparti.

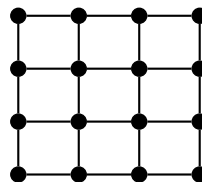
■

Exercice 2.2

Vérifier si le graphe maison est un graphe biparti. Faire de même pour la grille $G_{4,4}$.



maison



grille

2.2 Graphe connexe, eulérien, hamiltonien

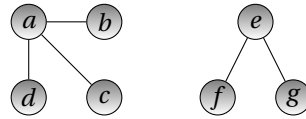
Définition 2.2: Graphe connexe

Un graphe est dit **connexe** si deux points peuvent toujours être joints par une chaîne, i.e.

$$\forall x, y \in V(G), d(x, y) < \infty$$

- Une **composante connexe** d'un graphe est un sous-graphe induit maximal connexe.
- Un **point d'articulation** d'un graphe est un sommet dont la suppression augmente le nombre de composantes connexes.

Par exemple, le graphe suivant :



n'est pas connexe car il n'existe pas de chaîne entre les sommets a et e . On a donc 2 composantes connexes. En revanche, le sous-graphe induit par les sommets $\{a, b, c, d\}$ est connexe dont a est le seul point d'articulation.

Remarque 5.

1. Un graphe connexe à n sommets possède au moins $n - 1$ arêtes (faite une preuve par récurrence).
2. L'algorithme de **parcours en profondeur** permet de déterminer si un graphe est connexe ou non.
3. Même définitions dans un **graphe orienté**. Graphe orienté (**fortement**) **connexe** et de **composante connexe**.

Définition 2.3: Graphe eulérien

- Une chaîne joignant deux sommets x et y est dite **chaîne eulérienne** si elle emprunte chaque arête de G une et une seule fois. On parle de chemin eulérien dans le cas d'un graphe orienté.
- Un **cycle eulérien** est une chaîne eulérienne dont les extrémités coïncident. Un graphe qui admet un cycle (resp. circuit) eulérien est dit **graphe eulérien**.

☺ Remarque qu'un cycle eulérien n'est pas forcément un cycle comme définit dans la définition 2.1

Si un graphe (ou multigraphe) G possède une chaîne eulérienne, alors soit G possède deux sommets impairs, le début et la fin de cette chaîne, soit G n'a aucun sommet impair, et la promenade d'Euler commence et se termine au même point (cycle eulérien). Une autre condition nécessaire évidente est que G doit être connexe. Ces deux conditions sont ensemble suffisantes.

Théorème 2.3 Euler

Soit G un graphe (multigraphe) connexe.

- Si G n'a pas de sommets impairs, alors il possède un cycle eulérien (commençant à n'importe quel point).

$$\text{i.e. } \forall x, d(x) \equiv 0 \pmod{2} \implies G \text{ est un graphe eulérien.}$$

- Si G a deux sommets impairs, alors il possède une chaîne eulérienne dont le début et la fin sont les sommets impairs.

La réciproque est aussi vraie.

Preuve.

- ⇐ On commence par la réciproque. S'il existe une chaîne eulérienne $\mu(x, y)$, alors on peut créer une arête fictive $e = (y, x)$ et par la suite on obtient un cycle eulérien. Donc il suffit de montrer que si G admet un cycle eulérien alors tous les sommets sont de degré pair.

Soit $W = \langle v_1, v_2, \dots, v_n, v_1 \rangle$ un cycle eulérien. Soit le sommet v_j avec $j \neq 1$, il est donc clair que chaque fois v_j apparaît dans W , il ya deux arcs distincts incidents en v_j ((\bullet, v_j) et (v_j, \bullet)). Comme toutes les arêtes sont utilisées, on voit que $d(v_j)$ est pair. Même chose pour v_1 car on ajoute deux arcs (v_1, v_2) et (v_1, v_n) .

- ⇒ On suppose que tous les sommets sont de degré pair. Montrons que G admet un cycle eulérien. Puisque tous les sommets sont de degré pair on peut construire d'abord une **chaîne simple fermée** (quelconque)

$$W_1 = \langle v_1, v_2, \dots, v_k, v_1 \rangle$$

Si W_1 contient chaque arête de G , alors c'est fini, c'est un cycle eulérien. Sinon (puisque G est connexe), il existe une arête $e = (v_i, c)$ qui n'est pas dans W_1 (avec v_i sommet de W_1). On pose le sous-graphe

$$G' = G - \{\text{arêtes de } W_1\}$$

Il est donc claire que les sommets de G' sont aussi de degré pair. Alors, par la même procédure on construit une **chaîne simple fermée** dans G' de la forme : $\langle v_i, c, d, \dots, v_i \rangle$ et par la suite on insère cette chaîne dans W_1 pour obtenir :

$$W_2 = \langle v_1, v_2, \dots, v_i, c, d, \dots, v_i, v_{i+1}, \dots, v_k, v_1 \rangle$$

Si $l(W_2) < |E(G)|$, on construit W_3 de longueur plus grand. Ainsi, on a :

$$l(W_1) < l(W_2) < l(W_3) < \dots \leq |E(G)|$$

Donc, nécessairement, il existe W_k (chaîne simple fermée de longueur $|E(G)|$) donc c'est un cycle eulérien.

Finalement, si $\exists a, b \in V(G) : d(a) \equiv d(b) \equiv 1 \pmod{2}$, et tous les autres sommets sont de degré pair, alors on pose :

$$G' = G + \{a, b\}$$

et donc tous les sommets de G' sont de degré pair. Avec la construction précédente, il existe un cycle eulérien dans G' :

$$W = \langle a, b, v_1, \dots, a \rangle$$

est donc la chaîne $\langle b, v_2, \dots, a \rangle$ est une chaîne eulérienne dans G .

■

Algorithme 1 : Euler

Entrées : G un graphe eulérien

Output : Un cycle eulérien W

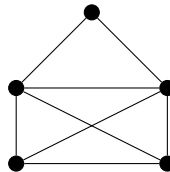
/* On suppose que G est eulérien, i.e. les sommets sont de degré pair */

```

1  $G_{aux} \leftarrow G$ ;  $W \leftarrow \mu(v, v)$  une chaîne simple (quelconque) de  $G$ ;           // Initialisation
2 tant que  $l(W) < e(G)$  faire
3    $G_{aux} \leftarrow G - \{\text{arêtes de } W\}$ ;
4   Choisir  $(x, c) \in E(G_{aux})$  tel que  $x \in W$ ;
5    $L \leftarrow \langle x, c, \dots, x \rangle$  une chaîne simple de  $G_{aux}$ ;
6   Dans  $W$ , remplacer (juste ce)  $x$  par la séquence  $L$ ;
7 fin
8 retourner  $W$ ;
```

Exercice 2.3

1. Montrer que le graphe de l'exercice 2.1 est un graphe eulérien et donner un cycle eulérien.
2. Tracer cette maison d'un seul trait.



Remarque 6. Le théorème d'Euler dans un graphe **orienté** connexe s'écrit :

Admet un circuit eulérien si, et seulement si, pour tout sommet, le degré entrant est égal au degré sortant.

$$\forall v \in V(G), d^+(v) = d^-(v)$$

Admet un chemin eulérien de a vers b si, et seulement si,

$$\begin{cases} d^+(v) = d^-(v) & ; \forall v \neq a, b, \\ d^-(a) = d^+(a) - 1 \\ d^-(b) = d^+(b) + 1 \end{cases}$$

Définition 2.4: Graphe hamiltonien

Soit G un graphe connexe d'ordre n . On appelle :

1. On appelle **cycle hamiltonien** de G un cycle passant (une et une seule fois) par chacun des sommets de G . Un graphe est dit **hamiltonien** s'il possède un cycle hamiltonien.
2. Une **chaîne hamiltonienne** est une chaîne passant une et une seule fois par chacun des sommets de G . Un graphe ne possédant que des chaînes hamiltoniennes est **semi-hamiltonien**.
3. Dans un digraphe, on change cycle par **circuit** et chaîne par **chemin**.

Remarque 7.

- Une chaîne hamiltonienne est une chaîne élémentaire de longueur $n - 1$.
- Un graphe peut être eulérien, hamiltonien, les deux à la fois, ou aucun des deux.
- De nombreux problèmes concrets peuvent être formulés en termes de recherche de parcours hamiltoniens : problème du **voyageur de commerce** (chercher un cycle hamiltonien de longueur totale minimale), l'**ordonnement de tâches**.
- Le problème du chaîne hamiltonienne est un problème **NP-complet**.

Exercice 2.4

Donner un graphe (connexe, simple) qui soit :

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

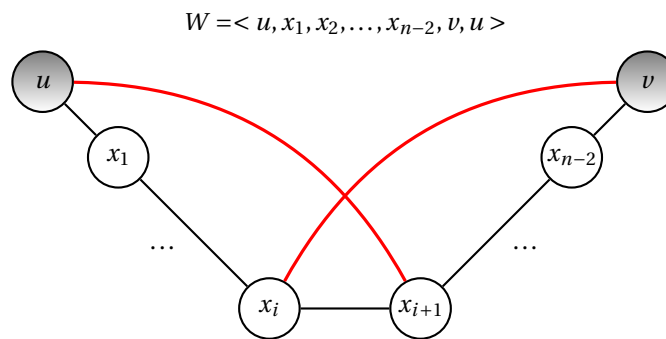
Aucune condition nécessaire et suffisante adéquate n'est connue pour l'existence de cycles hamiltoniens. Le résultat suivant constitue une condition suffisante utile.

Théorème 2.4 Ore

Soit G un graphe simple d'ordre $n \geq 3$. Si pour toute paire $\{x, y\}$ de sommets non adjacents, on a $d(x) + d(y) \geq n$, alors G est hamiltonien.

Preuve. On suppose que le théorème est faux. Alors, il existe un graphe G simple d'ordre n non hamiltonien tel que $d(u) + d(v) \geq n$ pour tout sommets non adjacents u et v . On prend G un tel graphe d'ordre n de taille maximale qui vérifie cette propriété (remarquer que le graphe complet K_n est hamiltonien).

Soient u et v deux sommets non adjacents de G . Par maximalité de G , le graphe $G + uv$ (d'ordre n) doit être hamiltonien. Donc il existe dans $G + uv$ un cycle hamiltonien :



Or, si $ux_{i+1} \in E(G)$ et $vx_i \in E(G)$ (les arcs en rouge), alors on peut construire dans G le cycle hamiltonien :

$$\langle u, x_1, x_2, \dots, x_i, v, x_{n-2}, \dots, x_{i+1}, u \rangle$$

Ceci est absurde car G est non hamiltonien. Donc si x_{i+1} est un sommet voisin à u , alors le sommet x_i n'est pas voisin de v :

$$x_{i+1} \in \Gamma(u) \implies x_i \notin \Gamma(v)$$

Donc :

$$d(u) = |\Gamma(u)| = p \implies d(v) \leq (n-1) - p$$

et par la suite

$$d(u) + d(v) \leq n-1$$

ce qui absurde par hypothèse. Donc G est hamiltonien. ■

Une conséquence immédiate est le théorème de Dirac :

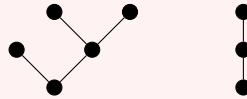
Proposition 2.1 Dirac

Soit G un graphe simple d'ordre $n \geq 3$. Si pour tout sommet x de G , on a $d(x) \geq \frac{n}{2}$, alors G est hamiltonien.

2.3 Arbres

Définition 2.5: Arbres et forêts

On appelle **arbre** tout graphe (non orienté) **connexe et acyclique**. Les composantes connexes dans un graphe acyclique sont des arbres et le graphe est dit **forêt**. Une **feuille** ou sommet pendant est un sommet de degré 1.



Un forêt avec deux arbres

Un graphe G est une **arborescence** s'il existe un sommet R appelé **racine** de G tel que, pour tout sommet S de G , il existe un chemin et un seul de R vers S . Donc, c'est un arbre orienté dans lequel on choisit un sommet R comme racine et on dirige les arcs vers le sens sortant de la racine.

Remarque 8.

- Un graphe G d'ordre n est un arbre ssi G est sans cycle et possède $n-1$ arêtes.
- Un graphe G d'ordre n est un arbre ssi G est connexe et possède $n-1$ arêtes.
- G est un arbre si pour chaque paire u, v de sommets distincts est reliée par une seule chaîne (donc chaîne élémentaire d'après le théorème 2.1).
- Un arbre (ou forêt) est nécessairement un graphe simple.

Pour démontrer les remarques précédentes, il suffit de prouver le théorème suivant :

Théorème 2.5

Soit G un graphe ayant m arêtes, n sommets et p composantes connexes, on définit $\nu(G)$:

$$\nu(G) = m - n + p$$

Alors, on a :

1. $\nu(G) \geq 0$.
2. $\nu(G) = 0 \iff G$ est acyclique.


Preuve. Par récurrence sur la taille m du graphe. Le résultat est évident pour $m = 0$ et $m = 1$. On suppose le résultat est vrai pour les graphes de taille m , et montrons que ceci est vrai pour un graphe de taille $m + 1$. Soit donc G un graphe d'ordre n de taille $m + 1$ ayant p composantes connexes.

- On suppose que G est un graphe acyclique. Soit uv une arête de G dans la composante C . Si on supprime uv , la composante C se transforme en deux composantes connexes (sinon on a un cycle dans G). Par la suite, le graphe $G' = G - uv$ est un graphe acyclique d'ordre n , de taille m et ayant $p + 1$ composantes connexes. Donc d'après l'hypothèse de récurrence :

$$v(G') = 0 = (m - 1) - n + (p + 1) = m - n + p = v(G)$$

- Maintenant, si G contient un cycle W dans la composante connexe C . Soit uv une arête de W , alors si on supprime l'arête uv , la composante C reste connexe. Donc le graphe $G' = G - uv$ est un graphe d'ordre n de taille $m - 1$ ayant p composantes connexes. Donc, d'après l'hypothèse de récurrence :

$$\begin{aligned} v(G') &= (m - 1) - n - p \geq 0 \\ \Rightarrow v(G) &= m - n - p \geq 1 > 0 \end{aligned}$$

D'où le résultat – car si $v(G) = 0$, G ne peut être que acyclique, sinon $0 = v(G) \geq 1$ 

■

2.3.1 Codage et décodage d'un arbre – Prüfer code

Soit T un arbre ayant n sommets numérotés $\{1, 2, \dots, n\}$. Le codage de Prüfer représente l'arbre T avec une suite $P = (x_1, x_2, x_3, \dots, x_{n-2})$ de $n - 2$ termes (les n° des sommets avec répétition possible). Une suite P donnée correspond à un et un seul arbre numéroté. Voici les algorithmes de codage et de décodage.

Algorithme 2 : Codage de Prüfer

Entrées : Un arbre T

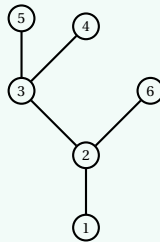
Output : Séquence de Prüfer P

```

1  $P \leftarrow ()$ ; // Initialisation de  $P$  comme séquence vide
2 tant que  $v(T) > 2$  faire
3    $x \leftarrow$  feuille de  $T$  ayant le  $n^\circ$  minimum ; // donc  $d(x) = 1$ 
4    $P \leftarrow (P, x)$  où  $x$  est le seul ( $n^\circ$ ) sommet adjacent à  $x$ ;
5    $T \leftarrow T - x$ ;
6 fin
7 retourner  $P$ ;
```

Exemple 2.2

Donner le code de Prüfer de l'arbre :



Indice : on trouve, $P = (2, 3, 3, 2)$

Pour faire la réciproque, étant donnée une séquence $P = (x_1, x_2, \dots, x_{n-2})$ où les $x_i \in \{1, 2, \dots, n\}$, on construit l'arbre T dont les sommets sont $\{1, 2, \dots, n\}$ avec l'algorithme suivant (le décodage) :

Algorithme 3 : Décodage de Prüfer**Entrées :** Une séquence P de taille $n - 2$ **Output :** L'arbre T correspondante

```

1  $I \leftarrow \{1, 2, \dots, n\};$ 
2  $T \leftarrow$  graphe tel que  $V(T) = I$  et  $E(T) = \emptyset$ ; //  $T$  graphe à  $n$  sommets isolés
3 tant que  $P \neq ()$  faire
4    $i \leftarrow \min \{x \in I : x \notin P\};$ 
5    $T \leftarrow T + (i, P[1]);$  // ajouter l'arc  $(i, P[1])$ 
6    $P \leftarrow P - P[1]; I \leftarrow I - \{i\};$ 
7 fin
8  $T \leftarrow T + (a, b);$  //  $a, b$  sont les deux éléments restant dans  $I$ 
9 retourner  $T$ ;
```

Exemple 2.3Donner l'arbre dont le code de Prüfer est : $P = (2, 3, 3, 3)$

Puisqu'il y a une bijection entre les arbres d'ordre n et les suites P de longueur $n - 2$ à coefficients dans $\{1, 2, \dots, n\}$, on déduit le théorème de Cayley :

Théorème 2.6 Formule de CayleyLe nombre d'arbres que l'on peut construire sur n ($n \geq 2$) sommets numérotés est égal à n^{n-2} .**2.3.2 Arbre couvrant de poids minimum – ACM**

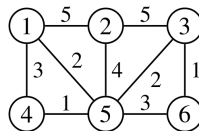
Soit G un graphe (non orienté). Un **arbre couvrant** (spanning tree) T de G est un sous graphe couvrant de G tel que T est un arbre. Remarquer que, G doit être connexe, sinon un arbre couvrant de G n'existe pas. Le nombre des arbres couvrant de G est noté $a(G)$. Remarquer que $a(K_n) = n^{n-2}$ (le nombre de Cayley). Un problème algorithmique classique est de trouver, dans **un graphe valué**, un **arbre couvrant de poids minimal**. On dispose de plusieurs algorithmes (l'algorithme de Prim, l'algorithme de Kruskal ...).

Algorithme 4 : Kruskal**Entrées :** Un graphe G valué d'ordre n et de taille m **Output :** Arbre (ou forêt) couvrant de poids minimum

```

1 Numéroter les arêtes de  $G$  par poids minimum :  $w(e_1) \leq w(e_2) \leq \dots \leq w(e_m)$ 
2  $T \leftarrow$  graphe tel que  $V(T) = V(G)$  et  $E(T) = \emptyset$ ; //  $T$  graphe à  $n$  sommets isolés
3  $k \leftarrow 0$ ;
4 tant que  $k < m$  et  $e(T) < n - 1$  faire
5   si  $T + e_{k+1}$  est acyclique alors
6      $T \leftarrow T + e_{k+1}$ ;
7    $k \leftarrow k + 1$ ;
8 fin
9 retourner  $T$ ;
```

Appliquer l'algorithme de Kruskal dans le graphe suivant :



CHAPITRE 3

Les algorithmes de base

Beaucoup de problèmes sur les graphes nécessitent que l'on parcoure l'ensemble des sommets et des arcs. Par exemple, pour vérifier que le graphe est connexe, chercher un sommet particulier, construire un arbre couvrant, trouver un plus court chemin ...etc.

3.1 Parcours en profondeur (DFS) / largeur (BFS)

Une manière simple de tester si un graphe G est connexe ou non est d'exécuter l'algorithme **DFS** ou **BFS** dans G en commençant par n'importe quel sommet et en enregistrant les sommets visités pendant l'exécution dans une liste. Si la liste contient tous les sommets de G , alors G est connexe. Une petite modification permet de donner les composantes connexes de G . L'algorithme **DFS_Component** suivant donne les composantes connexes de G . Il est donné sous forme d'implémentation récursive.

Algorithme 5 : DFS_Component

Entrées : Un graphe $G = (V, E)$

Output : Les composantes connexes $\mathcal{C} = (C_1, C_2, \dots, C_k)$ de G .

```
1 VisitedNodes  $\leftarrow \emptyset$ ;  
2  $k \leftarrow 0$ ; //  $k$  est le nombre des composantes connexes  
3 pour chaque  $u \in V$  faire  
4   si  $u \notin \text{VisitedNodes}$  alors  
5      $k \leftarrow k + 1$ ;  
6     DFS( $u$ );  
7   fin  
8 fin  
9 retourner  $(C_1, C_2, \dots, C_k)$ ;
```

```
1 Procédure DFS( $u$ )  
2   VisitedNodes  $\leftarrow \text{VisitedNodes} \cup \{u\}$ ;  
3    $C_k \leftarrow C_k \cup \{u\}$ ;  
4   pour chaque  $x \in \Gamma(u)$  faire  
5     si  $x \notin \text{VisitedNodes}$  alors  
6       DFS( $x$ );  
7   fin  
8 fin
```

Note :-

Il est possible de d'implémenter la procédure DFS **itérativement** à l'aide d'une **pile LIFO** (stack) contenant les sommets à explorer : on désempile un sommet et on empile ses voisins non encore explorés.

Pseudocode :

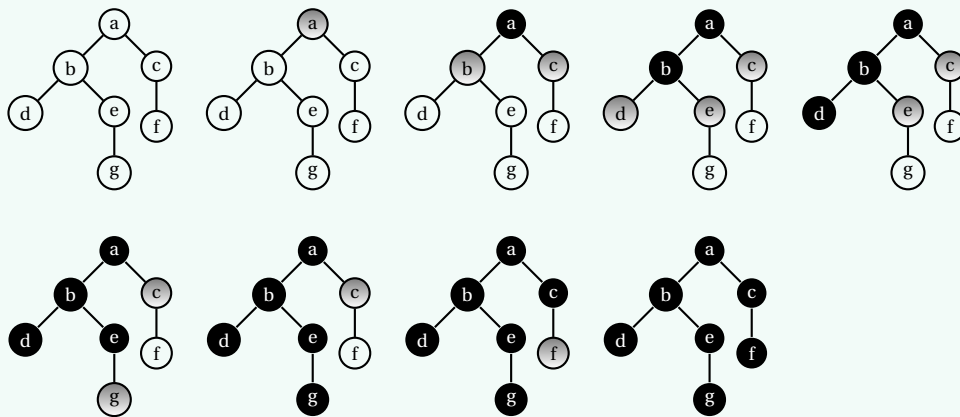
```

stack.push(root)
while stack.isEmpty()=false do
  node = stack.pop() //so mark it as visited
  for each x in node.childNodes do
    if x is not visited then
      stack.push(x)
  ...
endfor
endwhile

```

Example 3.1 (DFS)

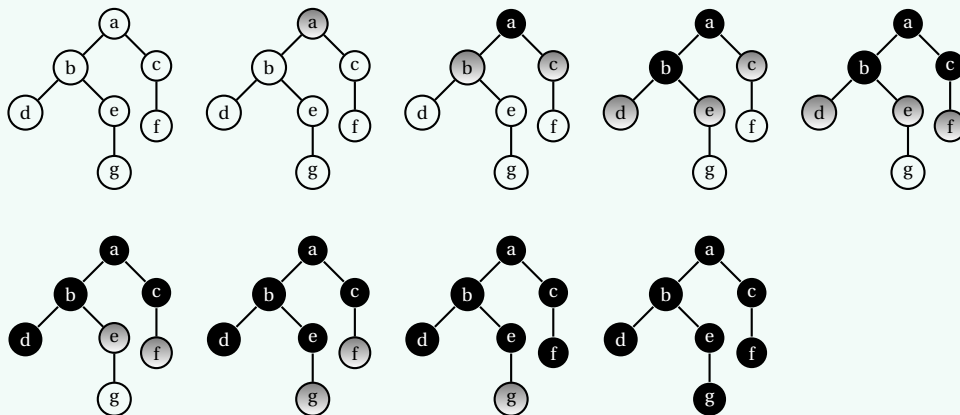
Ici on empile (gris) les sommets les plus à droite, alors l'ordre des sommets visité (noir) est (a, b, d, e, g, c, f) . Ici on prend $a = \text{root}$.



L'utilisation d'une **file** (FIFO) au lieu d'une **pile** transforme l'algorithme du parcours en profondeur en algorithme de parcours en largeur (**BFS**). Avec BFS l'exemple précédent devient (en enfile le plus à gauche) :

Example 3.2 (BFS)

Ici, avec BFS, on enfile (gris) les sommets les plus à gauche d'abord, alors l'ordre des sommets visité (noir) est (a, b, d, e, g, c, f) . Ici on prend $a = \text{root}$.



Exercice 3.1

1. Écrire la procédure **BFS**.
2. Utiliser DFS (ou BFS) pour la construction d'un arbre couvrant d'un graphe connexe. L'appliquer pour le graphe triangle.
3. Utiliser DFS pour écrire une procédure **DFS_Cycle** pour détecter un cycle dans un graphe.

On sait qu'un graphe orienté $G = (V, E)$ est **fortement connexe** si pour chaque paire de sommets $u, v \in V$, il existe un chemin de u à v et il existe un chemin de v à u . En d'autres termes, nous avons besoin d'une **connectivité dans les deux directions**.

Une procédure pour tester si un graphe orienté est fortement connexe ou non peut être conçue avec l'idée suivante. En partant d'un sommet arbitraire v du graphe G , on exécute l'algorithme **BFS** ou **DFS** et l'on note les sommets visités. Ensuite, on obtient la transposée de G , G^T , en **inversant la direction des arêtes de G** , et l'on exécute à nouveau l'algorithme BFS ou DFS à partir du sommet v . Si les sommets visités dans les deux cas sont égaux à V , alors le graphe orienté est fortement connexe. Cette méthode est illustrée dans l'algorithme 6.

Algorithme 6 : Fortement_connexe

Entrées : Un graphe orienté $G = (V, E)$

Output : **true** si G est fortement connexe et **false** sinon

```

1  $X \leftarrow \emptyset$ ;
2  $Y \leftarrow \emptyset$ ;
3  $v \leftarrow$  un sommet quelconque de  $V$ ;
4  $X \leftarrow \text{DFS}(G, v)$ ;                                // Enregistrez les sommets visités dans  $X$ 
5  $G^T \leftarrow G$  avec flèches inversées;
6  $Y \leftarrow \text{DFS}(G^T, v)$ ;
7 si  $X = Y = V$  alors
8   | retourner TRUE.
9 sinon
10  | retourner FALSE.
11 fin
```

3.2 Recherche des chemins

3.2.1 Matrice d'adjacence et nombre de chemins

La puissance de la matrice d'adjacence A montre qu'il y a des chemins (ou chaînes) de longueur k allant de x à y comme le montre le théorème suivant.

Théorème 3.1

Soit $G = (V, E)$ un (di)graphe. On note $V = \{1, 2, \dots, n\}$ et A sa matrice d'adjacence et A^k la puissance k -ème de A . Alors le terme d'indices i, j de A^k est le nombre de chemins (chaînes) de longueur k allant de i à j .

Preuve. Par récurrence sur k . Pour $k = 1$ c'est clair. On suppose le résultat est vrai à l'ordre k et montrons qu'elle est vrai pour $k + 1$. Soit $S = (s_{i,j}) = A^k$. Si on a un chemin de i à j de longueur $k + 1$, alors on va d'abord de i à un sommet r avec k pas et puis de r à j . Or d'après l'hypothèse de récurrence on a $s_{i,r}$ chemins possibles pour aller de i à r et finalement $a_{r,j}$ pas (1 ou 0) pour aller de r à j . Ainsi, le nombre totale possibles (i.e. tous les sommets r) est :

$$\sum_{r=1}^n s_{i,r} a_{r,j} = (A^k \times A)_{i,j}$$

ce qui est bien le terme (i, j) de la matrice A^{k+1} . ■

3.2.2 Plus court chemin et l'algorithme de Dijkstra

Dans cette partie on suppose que le (di)graphe $G = (V, E, w)$ est valué **positivement** (i.e. $\forall e \in E, w(e) > 0$). Notre objectif est de chercher un **plus court chemin** (i.e., un **chemin de poids minimal**) d'un sommet u fixé à un sommet quelconque de G .

Il est clair que l'on peut restreindre ce problème au cas d'un (di)graphe simple¹. On suppose aussi que G est connexe (sinon, certains trajets sont impossibles). Il est aussi claire (d'après théorème 2.1), qu'un plus court chemin est nécessairement un chemin élémentaire (path).

L'**algorithme de Dijkstra** (1959) permet de calculer le plus court chemin entre un sommet particulier u (**racine**) et tous les autres sommets. L'algorithme fonctionne de la manière suivante :

Pour tout sommet v de G , on associe une valeur $T(v)$, initialisée à $w(u, v)$ et une liste de sommets $C(v)$ qui correspond à un chemin de u à v . À la fin de l'algorithme, $T(v)$ contient le poids minimal des chemins joignant u à v (ou $T(v) = \infty$ si $u \nrightarrow v$) et la liste $C(v)$ réalise un tel chemin. Voici donc l'algorithme de Dijkstra :

Algorithme 7 : Algorithme de Dijkstra

Entrées : Un (di)graphe valué $G = (V, E, w)$ et un sommet **racine** u

Output : Le coût minimum d'un chemin de u à v ($\forall v \in V$) et un trajet pour chaque coût minimal obtenu

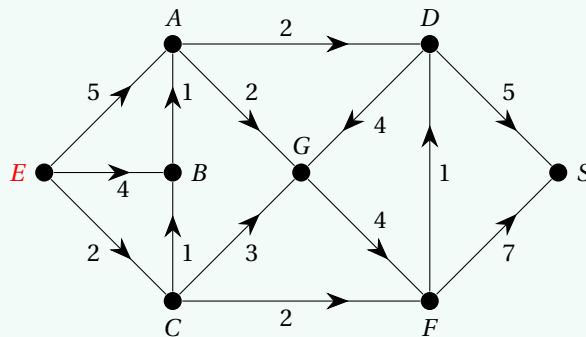
```

1   $\forall v \in V, T(v) \leftarrow w(u, v)$  et  $C(v) \leftarrow (u, v);$            // Initialisation de  $T(v)$  et la liste  $C(v)$ 
2   $X \leftarrow \{u\};$ 
3  tant que  $X \neq V$  faire
4      Choisir  $v \in V - X$  tel que  $\forall y \in V - X, T(v) \leq T(y);$ 
5       $X \leftarrow X \cup \{v\};$ 
6      pour chaque  $y \in V - X$  faire           // Faire les mises à jours pour les autres sommets
7          si  $T(y) > T(v) + w(v, y)$  alors
8               $T(y) \leftarrow T(v) + w(v, y);$ 
9               $C(y) \leftarrow [C(v), y];$            // On ajoute à la liste  $C(v)$  l'élément  $y$ 
10
11      fin
12  fin
13 fin

```

Exemple 3.3

Voici une application de l'algorithme de Dijkstra au digraphe suivant. On prend E comme sommet racine.



1. En effet, passer par une boucle ne ferait qu'augmenter inutilement le poids de ce chemin. De même, si plusieurs arcs joignent deux sommets, il suffit de conserver l'arc de poids minimal.

Ainsi, pour l'initialisation, on a : $X = \{E\}$ et on a les valeurs :

v	\cancel{E}	A	B	C	D	F	G	S
$T(v)$	0	5	4	2	∞	∞	∞	∞
$C(v)$	(E, E)	(E, A)	(E, B)	(E, C)	(E, D)	(E, F)	(E, G)	(E, S)

Maintenant, on choisit (un seul choix) le sommet C , donc $X = \{E, C\}$. Le tableau est mis à jour :

v	\cancel{E}	A	B	\cancel{C}	D	F	G	S
$T(v)$	0	5	3	2	∞	4	5	∞
$C(v)$	(E, E)	(E, A)	(E, C, B)	(E, C)	(E, D)	(E, C, F)	(E, C, G)	(E, S)

À l'étape suivante, on est forcé de choisir B . Ainsi, $X = \{E, C, B\}$ et la mise à jour est :

v	\cancel{E}	A	\cancel{B}	\cancel{C}	D	F	G	S
$T(v)$	0	4	3	2	∞	4	5	∞
$C(v)$	(E, E)	(E, C, B, A)	(E, C, B)	(E, C)	(E, D)	(E, C, F)	(E, C, G)	(E, S)

À présent, on a le choix entre A et F . On choisit A . Donc $X = \{E, C, B, A\}$ et on a :

v	\cancel{E}	\cancel{A}	\cancel{B}	\cancel{C}	D	F	G	S
$T(v)$	0	4	3	2	6	4	5	∞
$C(v)$	(E, E)	(E, C, B, A)	(E, C, B)	(E, C)	(E, C, B, A, D)	(E, C, F)	(E, C, G)	(E, S)

Puis, on prend le sommet F . D'où $X = \{E, C, B, A, F\}$ et :

v	\cancel{E}	\cancel{A}	\cancel{B}	\cancel{C}	D	\cancel{F}	G	S
$T(v)$	0	4	3	2	5	4	5	11
$C(v)$	(E, E)	(E, C, B, A)	(E, C, B)	(E, C)	(E, C, F, D)	(E, C, F)	(E, C, G)	(E, C, F, S)

On prend D et on obtient $X = \{E, C, B, A, F, D\}$ et :

v	\cancel{E}	\cancel{A}	\cancel{B}	\cancel{C}	\cancel{D}	\cancel{F}	G	S
$T(v)$	0	4	3	2	5	4	5	10
$C(v)$	(E, E)	(E, C, B, A)	(E, C, B)	(E, C)	(E, C, F, D)	(E, C, F)	(E, C, G)	(E, C, F, D, S)

Finalement, on prend G puis S et on voit que ceci ne change rien. D'où le tableau finale :

v	\cancel{E}	\cancel{A}	\cancel{B}	\cancel{C}	\cancel{D}	\cancel{F}	\cancel{G}	\cancel{S}
$T(v)$	0	4	3	2	5	4	5	10
$C(v)$	(E, E)	(E, C, B, A)	(E, C, B)	(E, C)	(E, C, F, D)	(E, C, F)	(E, C, G)	(E, C, F, D, S)

Ainsi le chemin minimum pour aller de E à S est de coût 10 et un trajet de coût minimal de E à S est : $ECFDS$.

3.3 Problèmes d'ordonnancement et chemin critique

Un problème d'ordonnancement peut-être représenté sous la forme d'un digraphe. Les sommets sont des événements (début et fin de la tâche) et les arcs des tâches. La longueur (ou poids) de chaque arc représente la durée d'exécution de la tâche. Ainsi, la tâche correspondant à l'arc (i, j) ne peut commencer que si toutes les tâches correspondant à des arcs (k, i) ont été complétées. On ajoute deux sommets pour le **début** (s) et de **fin** (t) du **projet**. Le digraphe peut contenir **des tâches fictives de durée nulle** afin de forcer certaines précédences (postériorité).

Ainsi, un **réseau PERT** est un digraphe valué $G = (V, E, w, s, t)$ où la fonction poids (ou longueur) $w: E \rightarrow \mathbb{R}_+$ et s le sommet source (début) et t le sommet puits (fin). Un **chemin critique** est un chemin de s à t de **longueur maximale**. S'il n'existe pas de chemin critique, le projet n'est pas réalisable (e.g. un circuit). Le temps minimal requis pour l'exécution d'un projet réalisable est égal à la longueur d'un chemin critique. Les tâches figurant sur un chemin critique sont aussi appelées critiques, ce qui s'explique par le fait que tout retard d'exécution d'une tâche critique retarde d'autant l'exécution du projet.

3.3.1 Algorithmes de tri topologique et de chemin critique

Il faut d'abord numérotés les sommets de 1 à n de manière compatible au pré-ordre (rang) avec l'algorithme du rang suivant (tri topologique) :

Algorithme 8 : Algorithme du rang

Entrées : Un digraphe $G = (V, E)$ sans circuit

Output : Le rang $r(v)$ de chaque sommet $v \in V$ de G

```

1  $r \leftarrow 0; X \leftarrow V;$ 
2  $R \leftarrow \{v \in X : \Gamma_X^-(v) = \emptyset\};$  // l'ensemble des sommets de  $X$  sans prédécesseur dans  $X$ 
3 tant que  $X \neq \emptyset$  faire
4    $r(v) \leftarrow r$  pour chaque sommet  $v \in R;$ 
5    $X \leftarrow X - R;$ 
6    $R \leftarrow$  l'ensemble des sommets de  $X$  sans prédécesseur dans  $X$ ;
7    $r \leftarrow r + 1;$ 
8 fin
```

On supposera dorénavant que les sommets ont déjà été numérotés de 1 à n de manière compatible avec leurs rangs, c'est-à-dire que $r(j) > r(i)$ implique $j > i$ (donc $1 = s$ et $n = t$). Enfin, on supposera éliminés les arcs parallèles par l'**introduction de tâches fictives de durée nulle**.

Pour un sommet i , on note t_i le **début au plus tôt** des tâches correspondant aux arcs $i \rightarrow j$ et par T_i la **fin au plus tard** des tâches correspondant aux arcs $j \rightarrow i$.

Algorithme 9 : Algorithme du chemin critique

Entrées : Un digraphe $G = (V, E)$ sans circuit trié topologiquement

Output : La durée du chemin critique et les dates t_i, T_i

```

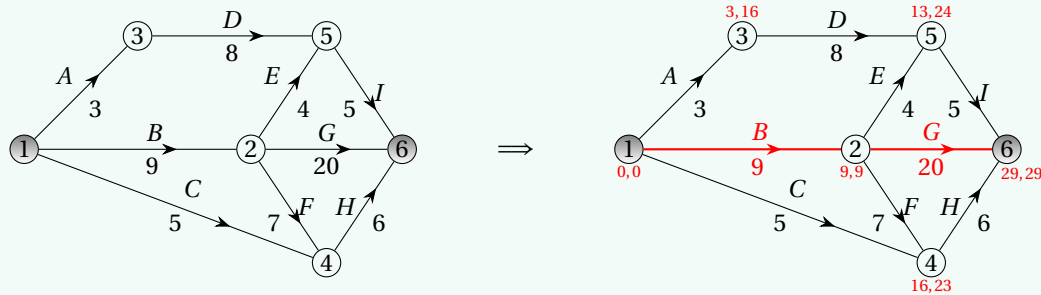
1  $t_1 \leftarrow 0;$ 
2 pour  $j = 2 \dots n$  faire // Calcul des dates de début au plus tôt - récurrence en avançant
3    $t_j \leftarrow \max_{i \in \Gamma^-(j)} \{t_i + w(i, j)\}$ 
4 fin
5  $T_n \leftarrow t_n;$ 
6 pour  $j = n - 1 \dots 1$  faire // Calcul des dates de fin au plus tard - récurrence en reculant
7    $T_j \leftarrow \min_{i \in \Gamma^+(j)} \{T_i - w(j, i)\}$ 
8 fin
```

Définition 3.1

- Un sommet i est critique si $t_i = T_i$.
- Un arc (i, j) est critique les sommets i et j sont critiques **et** $t_j - t_i = w(i, j)$.
- Un chemin critique est un chemin de 1 à n n'utilisant que des arcs critiques.
- La durée du chemin critique est donnée par T_n (ou par t_n). Elle correspond à la **durée minimale du projet**.

Exemple 3.4

L'application de l'algorithme du chemin critique au digraphe suivant (trié topologiquement) est :



Les tâches critiques sont B et G. La durée totale minimale du projet est donc 29.

Exercice 3.2

Les tâches (**durée et précédenes**) pour la construction d'un entrepôt sont données dans le tableau suivant :

Tâches	Nature	Précédenes	Durée (jours)
A	Acceptation des plans par le propriétaire	–	4
B	Préparation du terrain	–	2
C	Commande des matériaux	A	1
D	Creusage des fondations	A, B	1
E	Commande des portes et fenêtres	A	2
F	Livraison des matériaux	C	2
G	Coulage des fondations	D, F	2
H	Livraison des portes et fenêtres	E	10
I	Pose des murs, de la charpente et du toit	G	4
J	Mise en place des portes et fenêtres	H, I	1

1. Représentez le graphe des précédenes de ce projet (trié topologiquement).
2. Déterminez une durée totale minimale en exhibant un chemin critique dans ce graphe.

CHAPITRE 4

Programmation linéaire

La programmation linéaire est le processus de **minimiser ou maximiser** une fonction objective **linéaire** soumise à un nombre fini de **contraintes** d'égalités et d'inégalités linéaires.

La programmation linéaire a de nombreuses applications dans le domaine de l'informatique théorique. Elle peut être utilisée pour résoudre de nombreux problèmes combinatoires variés, tels que la recherche de flot maximum dans un réseau, la recherche d'un couplage maximal dans un graphe et la coloration d'un graphe parfait.

Exemple 4.1 (Problème de régime)

On suppose qu'on a quatre plats : de la truite (X_T), un sandwich au corned-beef (X_{CB}), un burrito (X_{BUR}) et un hamburger (X_{HB}). Le tableau suivant répertorie la valeur nutritionnelle de chaque plat (vitamines et calories) et notre besoin journalier des vitamines.

	Vit. A	Vit. C	Vit. D	Calories
Truite	203	92	100	600
Corned-beef	90	84	230	350
Burrito	270	80	512	250
Hamburger	500	90	210	500
Besoins	2000	300	430	

Notre objectif est de minimiser le nombre total de calories consommées : $600X_T + 350X_{CB} + 250X_{BUR} + 500X_{HB}$ tout en satisfaisant les trois besoins nutritionnels en vitamines. Le programme linéaire résultant s'écrit :

$$\begin{aligned} \min \quad & 600X_T + 350X_{CB} + 250X_{BUR} + 500X_{HB} \\ \text{s.t.} \quad & 203X_T + 90X_{CB} + 270X_{BUR} + 500X_{HB} \geq 2000, \quad (\text{Besoins de Vit. A}), \\ & 92X_T + 84X_{CB} + 80X_{BUR} + 90X_{HB} \geq 300, \quad (\text{Besoins de Vit. C}), \\ & 100X_T + 230X_{CB} + 512X_{BUR} + 210X_{HB} \geq 430, \quad (\text{Besoins de Vit. D}), \\ & X_T, X_{CB}, X_{BUR}, X_{HB} \geq 0 \end{aligned}$$

Définition 4.1: Forme canonique – Forme standard

On peut écrire un P.L. sous deux formes spéciales :

• Canonique

$$\begin{aligned} \max \quad & c^T X + c_0 \\ \text{s.t.} \quad & AX \leq b \quad (\geq b \text{ si min}), \\ & X \geq 0 \end{aligned}$$

• Standard

$$\begin{aligned} \max \quad & c^T X + c_0 \\ \text{s.t.} \quad & AX = b, \\ & X \geq 0 \end{aligned}$$

Avec :

$$\begin{aligned} c^T X &= (c_1 \quad \cdots \quad c_n) \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \\ c^T X + c_0 &= c_1 x_1 + c_2 x_2 + \cdots + c_n x_n + c_0 \end{aligned}$$

Remarque 9. Un PL. général peut être formulé selon une forme standard ou canonique par la méthode suivante :

générale \Rightarrow standard il faut éliminer les contraintes d'inégalité et les variables sans contrainte de positivité.

— Si $d_i^T X \leq b_i$ est une contrainte d'inégalité, alors créer une nouvelle variable (**d'écart**) s_i et remplacer l'inégalité $d_i^T X \leq b_i$ par $d_i^T X + s_i = b_i$ et $s_i \geq 0$.

Si $d_i^T X \geq b_i$, alors on peut écrire $d_i^T X - s_i = b_i$ et $s_i \geq 0$.

— Si une variable $x_i \leq 0$ est de signe quelconque, alors créez deux nouvelles variables x_i^+ et x_i^- et remplacez x_i partout par $x_i^+ - x_i^-$, et ajoutez les contraintes de signe : $x_i^+ \geq 0, x_i^- \geq 0$.

générale \Rightarrow canonique

— Si $d_i^T X = b_i$ est une contrainte d'égalité, alors remplacer cette égalité par une paire d'inégalités :

$$d_i^T X \geq b_i \quad \text{et} \quad -d_i^T X \geq -b_i$$

— Même chose que précédemment si la variable $x_i \leq 0$.

Il est facile de voir que les problèmes qui en résultent sont équivalents au problème d'origine. Pour l'algorithme **simplexe**, nous **utiliserons la forme standard**.

On remarque aussi qu'en utilisant la relation $\min f(x) = -\max(-f(x))$ dans laquelle $f(x)$ représente la fonctionnelle linéaire à optimiser, on peut toujours se ramener à un problème de minimisation (ou de maximisation).

Exemple 4.2

$$\begin{array}{ll} \max & 5x + 4y \\ \text{s.t.} & x \leq 6 \\ \text{Le PL :} & \frac{1}{4}x + y \leq 6 \\ & 3x + 2y \leq 22 \\ & x, y \geq 0 \end{array} \quad \text{s'écrit sous forme standard :} \quad \begin{array}{ll} \max & 5x_1 + 4x_2 \\ \text{s.t.} & x_1 + e_1 = 6 \\ & \frac{1}{4}x_1 + x_2 + e_2 = 6 \\ & 3x_1 + 2x_2 + e_3 = 22 \\ & x_1, x_2, e_1, e_2, e_3 \geq 0 \end{array}$$

Donc (sous la forme matricielle) on a :

$$c^T = (5 \quad 4 \quad 0 \quad 0 \quad 0) ; \quad A = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ \frac{1}{4} & 1 & 0 & 1 & 0 \\ 3 & 2 & 0 & 0 & 1 \end{pmatrix} ; \quad X = \begin{pmatrix} x_1 \\ x_2 \\ e_1 \\ e_2 \\ e_3 \end{pmatrix} \quad \text{et} \quad b = \begin{pmatrix} 6 \\ 6 \\ 22 \end{pmatrix}$$

4.1 La méthode simplexe

4.1.1 Solution de base admissible

On suppose que la matrice A du programme linéaire est de type $m \times n$ et **de rang m (pourquoi!)**. Soit B la matrice carrée extraite de A formée par les m colonnes j_1, j_2, \dots, j_m linéairement indépendants de A (les vecteurs de base). Alors, B est une matrice inversible. Ainsi si on pose $x_l = 0$ si $l \notin \{j_1, \dots, j_m\}$ et x_{j_k} la k -ième composante de $B^{-1}b$. Alors, on a $AX_B = b$. Si de plus $X_B \geq 0$ (i.e. chaque $x_i \geq 0$), alors :

X est une solution de base admissible (ou réalisable)

Dans l'exemple 4.2, la matrice extraite $B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ est inversible et $B^{-1}b = \begin{pmatrix} 6 \\ 6 \\ 22 \end{pmatrix}$. Donc $X_B = \begin{pmatrix} 0 \\ 0 \\ 6 \\ 6 \\ 22 \end{pmatrix} \geq 0$, par suite

X_B est une solution de base admissible. On appelle les variables x_{j_k} les **variables de base**. Les autres variables x_l sont les **variables hors-base**.

Parfois, **il n'est pas évident** de trouver une solution de base admissible, comme le montre l'exercice suivant :

Exercice 4.1

Écrire le programme linéaire suivant sous forme standard et montrer qu'on n'a pas une solution de base admissible. Résoudre le programme linéaire.

$$\begin{aligned} \max \quad & -x + y \\ \text{s.t.} \quad & x + y \geq 1, \\ & 3x + 2y = 6, \\ & x, y \geq 0 \end{aligned}$$

4.1.2 L'algorithme du simplexe avec la méthode des tableaux

Dans cette section, on suppose qu'on a déjà trouvé une solution de base admissible X (produite avec $B = I_m$) du programme linéaire (\mathcal{P} : $\max z = c^T X + c_0$ et $b \geq 0$ pour garantir que 0 est une solution de $AX \leq b$).

La méthode du simplexe est une méthode itérative qui génère une séquence de solutions de base réalisable (correspondant à différentes bases) et s'arrête finalement lorsqu'elle a trouvé une solution de base réalisable optimale. Remarquer qu'on peut toujours prendre $c_0 = 0$.

Algorithme 10 : Algorithme du simplexe

Entrées : Un PL. standard \mathcal{P} (max) avec une solution de base réalisable

Output : La solution optimale (ou non)

- 1 On commence à écrire le tableau initiale (avec les variables de base x_{j_1}, \dots, x_{j_m})

V_B	x_1	\dots	x_n	b
x_{j_1}	A			b_1
\vdots				\vdots
x_{j_m}				b_m
	c_1	\dots	c_n	$-c_0$

- 2 Si $c_j \leq 0$ ($\forall j$: x_j est hors base), alors solution optimale (STOP) // $\min \rightsquigarrow c_j \geq 0$
 3 Trouver la colonne s (hors base) tel que $c_s = \max\{c_j : c_j > 0\}$ // $\min \rightsquigarrow c_s = \min\{c_j : c_j < 0\}$
 4 Si $a_{i,s} \leq 0 \forall i = 1 \dots m$, alors pas de solution optimale (STOP)
 5 **Min Ratio Test** : trouver la **ligne** r tel que

$$\frac{b_r}{a_{r,s}} = \min_{i=1 \dots m} \left\{ \frac{b_i}{a_{i,s}} : a_{i,s} > 0 \right\}$$

- 6 Appliquer la procédure de **Gauss** autour de **pivot** $a_{r,s}$ ($a_{r,s} \rightsquigarrow 1$ et on annule les coefficients situés **sous et sur** le pivot dans la colonne s , par combinaisons de lignes) // Ne pas oublier la dernière ligne
 7 Remplacer (dans V_B) la variable de base (**sortante**) située à la ligne r par la variable (**entrante**) x_s
 8 Retourner à l'étape 2 // l'algorithme termine si le programme linéaire est non dégénéré

Remarque 10.

- La dernière ligne représente l'équation linéaire $-z + c_1 x_1 + \dots + c_n x_n = -c_0$.
- La valeur de la variable de base de V_B de la ligne k est égale à la valeur b_k (de la ligne k).
- La solution optimale du problème est la valeur finale (dernière cellule) ($\times - 1$).
- Si on n'a pas au départ $B = I_m$ (à l'aide des variables d'écart) alors, on ajoute les variables **artificielles** (et on applique d'abord la phase I - voir section suivante).

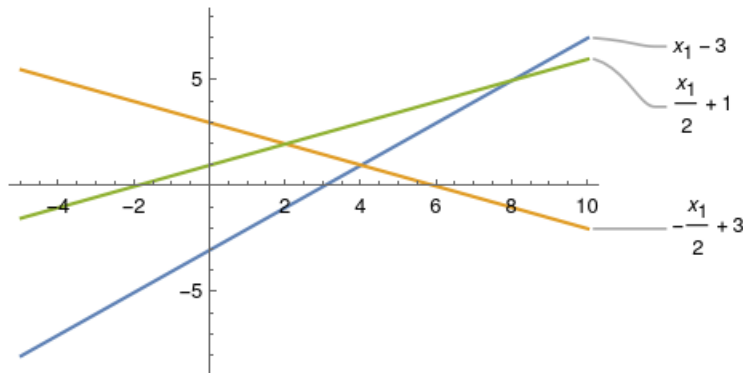
Exercice 4.2

Soit le programme linéaire :

$$\begin{aligned} \max \quad & 2x_1 + x_2 \\ \text{s.t.} \quad & x_1 - x_2 \leq 3, \\ & x_1 + 2x_2 \leq 6, \\ & -x_1 + 2x_2 \leq 2, \\ & x_1, x_2 \geq 0 \end{aligned}$$

Rajouter les variables d'écart. Puis résoudre le problème par l'algorithme du simplexe.

Remarque 11. Vérifier le résultat de l'exercice précédent, par la méthode graphique.

**4.1.3 Initialisation - Phase I du simplexe**

Un programme linéaire est dit sous forme **simpliciale** si la matrice A contient la matrice identité I_m comme sous matrice. On suppose que le programme linéaire \mathcal{P} , après la mise en forme $AX = b$, n'est pas simpliciale (possible dans le cas de contraintes du type \geq ou du type $=$). Soit B une matrice d'ordre m extraite de A inversible tel que $B^{-1}b \geq 0$ et soit X_B la solution de base réalisable ($X_B \geq 0$ et $AX_B = b$), alors on peut changer l'équation matricielle :

$$\begin{aligned} AX_B &= b \\ \Leftrightarrow (B^{-1}A)X_B &= B^{-1}b \\ \Leftrightarrow A^*X_B &= b^* \end{aligned}$$

et la matrice A^* contient la matrice identité I_m comme matrice extraite ($B^{-1}A = B^{-1}(H|B) = (B^{-1}H|I_m)$). Ainsi, on peut appliquer directement l'algorithme 10 en remplaçant A par A^* et b par b^* . Cependant, cette méthode n'est pas efficace, car il faut trouver une matrice extraite B réalisable correspondante. Pour une matrice A de type $m \times n$, il existe $\binom{m}{n}$ matrices extraites à tester!

L'idée est d'ajouter des variables, dites **artificielles** w_1, w_2, \dots, w_m , pour faire apparaître une forme simpliciale du programme linéaire \mathcal{P} : ($AX = b$). Mais, il faut que $w_1 = \dots = w_m = 0$. Donc, il faut minimiser d'abord le programme linéaire \mathcal{P}_0 suivant :

$$\begin{aligned} \min \quad & W = w_1 + \dots + w_m \\ \mathcal{P}_0 = \quad \text{s.t.} \quad & (A|I_m) \times (x_1 \dots x_n w_1 \dots w_m)^T = b \\ & x_1, \dots, x_n, w_1, \dots, w_m \geq 0 \end{aligned}$$

On exprime W à l'aide des autres variables (hors base) et puis on applique l'algorithme du simplexe 10 pour \mathcal{P}_0 (phase I). À la fin, **si on trouve que les $w_i = 0$** , alors les variables de bases x_{j_1}, \dots, x_{j_m} forment une première base admissible pour démarrer le simplexe pour le programme d'origine \mathcal{P} .

Remarque 12. On ajoute la ligne de Z : $\begin{bmatrix} c_1 & \dots & c_n & -c_0 \end{bmatrix}$ dans le tableau du simplexe de \mathcal{P}_0 (phase I) et on lui applique les opérations élémentaire comme avec les autres lignes. À la fin de la phase I, on **supprime** les colonnes des w_i et la ligne de la fonction W . Finalement, on utilise ce tableau pour résoudre le problème initiale \mathcal{P} .

Exemple 4.3

Soit le programme linéaire \mathcal{P} :

$$\begin{array}{ll} \min & 6x_1 + 3x_2 \\ \text{s.t.} & x_1 + x_2 \geq 1, \\ & 2x_1 - x_2 \geq 1, \\ & 3x_2 \leq 2, \\ & x_1, x_2 \geq 0 \end{array} \quad \xrightarrow{\text{sous forme standard (max)}} \quad \begin{array}{ll} \max & Z = -6x_1 - 3x_2 \\ \text{s.t.} & x_1 + x_2 - e_1 = 1, \\ & 2x_1 - x_2 - e_2 = 1, \\ & 3x_2 + e_3 = 2, \\ & x_1, x_2, e_1, e_2, e_3 \geq 0 \end{array}$$

La matrice A est :

$$A = \begin{pmatrix} x_1 & x_2 & e_1 & e_2 & e_3 \\ 1 & 1 & -1 & 0 & 0 \\ 2 & -1 & 0 & -1 & 0 \\ 0 & 3 & 0 & 0 & 1 \end{pmatrix}$$

est par suite \mathcal{P} n'est pas sous forme simpliciale (il n'y a pas de sous matrice I_3). Donc on peut ajouter 2 variables artificielles pour construire le problème initiale \mathcal{P}_0 :

$$\begin{array}{ll} \max & -w_1 - w_2 = 3x_1 - e_1 - e_2 - 2 \\ \text{s.t.} & x_1 + x_2 - e_1 + w_1 = 1, \\ & 2x_1 - x_2 - e_2 + w_2 = 1, \\ & 3x_2 + e_3 = 2, \\ & x_1, x_2, e_1, e_2, e_3, w_1, w_2 \geq 0 \end{array} \quad \text{dont le vecteur } X = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 2 \\ 1 \\ 1 \end{pmatrix} \begin{matrix} x_1 \\ x_2 \\ e_1 \\ e_2 \\ e_3 \\ w_1 \\ w_2 \end{matrix} \text{ est une solution de base réalisable.}$$

Ainsi, on commence notre algorithme 10 pour le programme \mathcal{P}_0 . Le tableau initial est :

$$\begin{array}{c} w_1 \\ w_2 \\ e_3 \\ \mathbf{W}: \\ Z: \end{array} \begin{pmatrix} x_1 & x_2 & e_1 & e_2 & e_3 & w_1 & w_2 & b \\ 1 & 1 & -1 & 0 & 0 & 1 & 0 & 1 \\ 2 & -1 & 0 & -1 & 0 & 0 & 1 & 1 \\ 0 & 3 & 0 & 0 & 1 & 0 & 0 & 2 \\ 3 & 0 & -1 & -1 & 0 & 0 & 0 & 2 \\ -6 & -3 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Le pivot c'est $a_{2,1}$ donc x_1 est la variable entrante et w_2 est sortante. On obtient après pivotement le tableau :

$$\begin{array}{c} w_1 \\ x_1 \\ e_3 \\ \mathbf{W}: \\ Z: \end{array} \begin{pmatrix} x_1 & x_2 & e_1 & e_2 & e_3 & w_1 & w_2 & b \\ 0 & \frac{3}{2} & -1 & \frac{1}{2} & 0 & 1 & \frac{-1}{2} & \frac{1}{2} \\ 1 & \frac{-1}{2} & 0 & \frac{-1}{2} & 0 & 0 & \frac{1}{2} & \frac{1}{2} \\ 0 & 3 & 0 & 0 & 1 & 0 & 0 & 2 \\ 0 & \frac{3}{2} & -1 & \frac{1}{2} & 0 & 0 & \frac{-3}{2} & \frac{1}{2} \\ 0 & -6 & 0 & -3 & 0 & 0 & 3 & 3 \end{pmatrix}$$

$\xrightarrow{\begin{matrix} L_2 \leftarrow \frac{1}{2}L_2 \\ L_1 \leftarrow L_1 - L_2 \\ L_4 \leftarrow L_4 - 3L_2 \\ L_5 \leftarrow L_5 + 6L_2 \end{matrix}}$

Le pivot est $a_{1,2}$ (**Min ratio test** = $\frac{1}{3}$). On obtient alors :

$$\begin{array}{l}
L_1 \leftarrow \frac{2}{3}L_1 \\
L_2 \leftarrow L_2 + \frac{1}{2}L_1 \\
L_3 \leftarrow L_3 - 3L_1 \\
L_4 \leftarrow L_4 - \frac{3}{2}L_1 \\
L_5 \leftarrow L_5 + 6L_1
\end{array}
\begin{array}{l}
x_2 \\
x_1 \\
e_3 \\
\mathbf{W}: \\
\mathbf{Z}:
\end{array}
\begin{array}{c}
\begin{pmatrix} 0 & 1 & \frac{-2}{3} & \frac{1}{3} & 0 & \frac{2}{3} & \frac{-1}{3} & \frac{1}{3} \end{pmatrix} \\
\begin{pmatrix} 1 & 0 & \frac{-1}{3} & \frac{-1}{3} & 0 & \frac{1}{3} & \frac{1}{3} & \frac{2}{3} \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 2 & -1 & 1 & -2 & 1 & 1 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 \end{pmatrix} \\
\begin{pmatrix} 0 & 0 & -4 & -1 & 0 & 4 & 1 & 5 \end{pmatrix}
\end{array}$$

Ainsi, la phase I se termine avec $W = 0$ et $x_1 = \frac{2}{3}, x_2 = \frac{1}{3}, e_3 = 1$ et $e_1 = e_2 = w_1 = w_2 = 0$. Le tableau pour la phase II est donc :

$$\begin{array}{c}
x_2 \\
x_1 \\
e_3 \\
\mathbf{Z}:
\end{array}
\begin{pmatrix}
0 & 1 & \frac{-2}{3} & \frac{1}{3} & 0 & \frac{1}{3} \\
1 & 0 & \frac{-1}{3} & \frac{-1}{3} & 0 & \frac{2}{3} \\
0 & 0 & 2 & -1 & 1 & 1 \\
0 & 0 & -4 & -1 & 0 & 5
\end{pmatrix}$$

On remarque donc que les $c_j \leq 0$, donc (STOP) la solution est optimale et le max est -5 . Ainsi la solution optimale pour le programme $\mathcal{P} = \min 6x_1 + 3x_2$ est $-(-5) = 5 = 6 \times \frac{2}{3} + 3 \times \frac{1}{3}$.

4.2 Dualité

Soit \mathcal{P} un P.L. sous forme canonique. Le programme **dual** \mathcal{D} de \mathcal{P} est défini par :

$$\begin{array}{ll}
\max & z = c^T X \\
\mathcal{P} = \text{s.t.} & AX \leq b \\
& X \geq 0
\end{array}
\begin{array}{c}
\text{le dual est} \\
\Longrightarrow
\end{array}
\begin{array}{ll}
\min & w = b^T Y \\
\mathcal{D} = \text{s.t.} & A^T Y \geq c \\
& Y \geq 0
\end{array}$$

Le programme \mathcal{P} est dit **primal**. On remarque que si A de type $m \times n$, alors $X \in \mathbb{R}^n$ et $Y \in \mathbb{R}^m$ pour le dual.

Théorème 4.1 Théorème de dualité

Si un problème primal ou dual admet une solution, alors l'autre problème admet aussi une solution et on a :

$$\begin{array}{ll}
\max & z = c^T X \\
\text{s.t.} & AX \leq b \\
& X \geq 0
\end{array}
=
\begin{array}{ll}
\min & w = b^T Y \\
\text{s.t.} & A^T Y \geq c \\
& Y \geq 0
\end{array}$$

Exercice 4.3

Soit le problème suivant de type régime :

$$\begin{array}{ll}
\min & w = 340x_1 + 2400x_2 + 560x_3 \\
\text{s.t.} & x_1 + 2x_2 + x_3 \geq 1200 \\
\mathcal{D} = & x_1 + 3x_2 + 2x_3 \geq 1400 \\
& x_1 + x_2 + 3x_3 \geq 1500 \\
& x_1, x_2, x_3 \geq 0
\end{array}$$

Écrire le programme primal \mathcal{P} de \mathcal{D} . Résoudre le programme \mathcal{P} et déduire la valeur minimale de w , a-t-on besoin de faire les deux phases du simplexe ?