Théorie de la Complexité – Feuille de cours intégré n° 5
Complexité en temps

Rappelons que l'objectif de ce cours est d'établir une **classification des problèmes** selon leur difficulté algorithmique. Nous avons déjà dégagé certaines classes de problèmes, dont les problèmes décidables et les problèmes indécidables. Notre classification est en fait un peu plus fine : nous avons défini les problèmes semi-décidables et les problèmes co-semi-décidables (ceux dont le complémentaire est semi-décidable).

Exercice 1 – ☆ – Rappeler les définitions de ces termes et donner les inclusions entre ces classes.

À partir de cette semaine, nous nous intéressons uniquement à des problèmes **décidables**. Nous allons classer les problèmes décidables selon les ressources qu'ils requièrent : le **temps de calcul** et **l'espace mémoire nécessaire**. Nous sommes intéressés par des bornes supérieures (*i.e.*, par des énoncés du type « tel problème peut être résolu en temps polynomial »), mais aussi par des bornes inférieures sur les ressources nécessaires.

1 Complexité déterministe en temps

On commence par définir les classes de complexité en temps. Nous en verrons deux versions : déterministe et non-déterministe. Pour cela, il nous faut d'abord définir le temps d'exécution d'une machine de Turing. Étant donné un alphabet Σ et un mot $w \in \Sigma^*$, on note $|w|$ la taille de w (c'est-à-dire son nombre de lettres, par exemple $|a\,a\,b| = 3$).

Definition 1 - Temps d'exécution d'une machine de Turing déterministe.

On considère un alphabet Σ et une machine de Turing déterministe M dont Σ est l'alphabet d'entrée. Étant donnée une fonction $f : \mathbb{N} \to \mathbb{N}$, on dit que le temps d'exécution de M est borné par f lorsque pour toute entrée $w \in \Sigma^*$, l'exécution de M sur w s'arrête après **au plus** $f(|w|)$ transitions.

Remarquez que la définition ci-dessus s'applique à tout type de machine de Turing déterministe (machines classiques, machines de décision, machines à plusieurs bandes). On peut maintenant définir les classes de complexité en temps déterministe. Ce sont des classes de langages (donc aussi, de problèmes) : on se sert des machines de Turing de décision.

Definition 2 - Classes de complexité déterministes en temps.

On considère une fonction $f : \mathbb{N} \to \mathbb{N}$. On note **DTIME**$(f)$ la classe de tous les langages L pour lesquels il existe des constantes $h, c \in \mathbb{N}$ telle que L est décidé par une machine de Turing dont le temps d'exécution est borné par la fonction $n \mapsto h \times f(n) + c$.

Remarque 1 - Pourquoi « D » ?. La lettre « D » dans la notation **DTIME**(f) réfère au mot « déterministe ». On l'utilise par opposition aux classes de complexité non-déterministes qu'on verra plus tard.

Il est standard de simplifier les notations et de remplacer la fonction f par sa définition. Par exemple, quand f est la fonction $f : n \mapsto n^2$, on écrira directement **DTIME**(n^2) pour **DTIME**(f).

Nous n'avons par posé de conditions particulières sur les fonctions f que nous pouvons utiliser. Pour cette raison, il est possible de définir des classes très exotiques. On va maintenant présenter les plus standard.

Definition 3 - Temps polynomial.

On note **PTIME** (ou simplement **P**), la classe suivante :

$$\textbf{PTIME} = \bigcup_{k\in\mathbb{N}} \textbf{DTIME}(n^k).$$

Definition 4 - Temps exponentiel.

On note **EXPTIME** la classe suivante :

$$\textbf{EXPTIME} = \bigcup_{k\in\mathbb{N}} \textbf{DTIME}(2^{n^k}).$$

Il est facile de comparer les classes **PTIME** et **EXPTIME**. On le fait dans le théorème suivant.

Théorème 2

On a l'inclusion **PTIME** $\subseteq$ **EXPTIME**. De plus, cette inclusion est stricte : on a **PTIME** $\neq$ **EXPTIME**.

Le Théorème 2 énonce une inclusion stricte entre les classes **PTIME** et **EXPTIME**. Intuitivement, il exprime qu'en augmentant significativement la ressource « temps » qu'une machine est autorisée à utiliser, on peut résoudre plus de problèmes. Cependant, ce type de résultat est rare en complexité : on est souvent capable de prouver qu'une classe est incluse dans une autre mais pas que l'inclusion est stricte (ni, à l'inverse, que les deux classes sont égales).

Exercice 2 – ❗★ – On va prouver le Théorème 2. On va commencer par la partie « facile » :

1. Montrer qu'on a bien l'inclusion **PTIME** $\subseteq$ **EXPTIME**.

Il nous faut maintenant montrer que cette inclusion est stricte. On va donc devoir trouver un langage qui appartient à **EXPTIME** mais pas à **PTIME**. On s'inspire du premier langage indécidable **DIAG** qu'on a rencontré dans la partie indécidabilité. On définit :

$$\textbf{DEXP} = \left\{ c(M) \;\middle|\; c(M) \notin \mathscr{L}(M) \text{ ou l'exécution de } M \text{ sur } c(M) \text{ dure plus de } 2^{|c(M)|} \text{ étapes} \right\}.$$

2. Montrer que **DEXP** $\in$ **EXPTIME**.
3. Montrer que **DEXP** $\notin$ **PTIME**.

Remarque 3 - Hiérarchie de classes exponentielles. Il est standard de définir des classes encore plus grandes que **EXPTIME**. Par exemple, on définit la classe 2–**EXPTIME** de la façon suivante :

$$2\text{–}\textbf{EXPTIME} = \bigcup_{k\in\mathbb{N}} \textbf{DTIME}(2^{2^{n^k}})$$

De la même façon, on peut définir 3–**EXPTIME**, 4–**EXPTIME**,... En adaptant les arguments servant à prouver le Théorème 2, on peut montrer qu'on obtient une hiérarchie stricte et infinie de classes :

$$\textbf{P} \subsetneq \textbf{EXPTIME} \subsetneq 2\text{–}\textbf{EXPTIME} \subsetneq 3\text{–}\textbf{EXPTIME} \subsetneq 4\text{–}\textbf{EXPTIME} \subsetneq 5\text{–}\textbf{EXPTIME} \subsetneq \cdots$$

Il est aussi classique de noter **ELEMENTARY** l'union de toutes ces classes :

$$\textbf{ELEMENTARY} = \bigcup_{k\geqslant 1} k\text{–}\textbf{EXPTIME}$$

On représente la situation graphiquement dans la Figure 1.

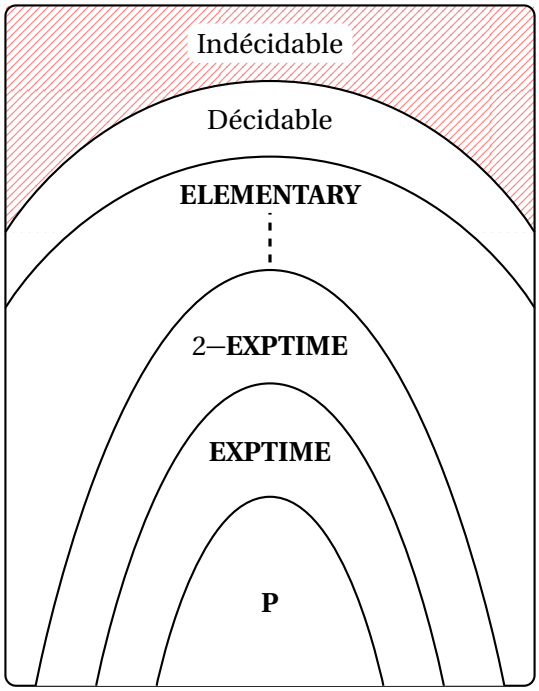


FIGURE 1 – Classes de complexité déterministes en temps

On va maintenant donner un exemple naturel de problème qui est dans **EXPTIME**, mais dont nous pensons qu'il n'est pas dans **P** : **SAT**. Ici le terme « nous » désigne la communauté scientifique qui croit fermement que **SAT** $\notin$ **P**, mais est jusqu'ici incapable de le prouver : c'est seulement une conjecture.

Definition 5 - Satisfaisabilité d'une formule propositionnelle (SAT).

Une formule propositionnelle est définie à l'aide d'un ensemble fini de variables Booléennes et de connecteurs logiques $\neg, \wedge, \vee$. Par exemple, $\varphi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (x_2 \vee x_3 \vee \neg(x_1 \wedge x_2))$ est une formule propositionnelle sur l'ensemble de variables $\{x_1, x_2, x_3\}$. Chaque variable peut prendre les valeurs de vérité *Vrai* ou *Faux*. Une formule propositionnelle est dite *satisfaisable* si il existe une affectation de ses variables telle que la formule s'évalue à *Vrai*. Le problème **SAT** est le suivant :

ENTRÉE : φ une formule propositionnelle.
QUESTION : φ est-elle satisfaisable ?

Remarque 4 - Attention. Si φ est une formule propositionnelle, on ne peut pas dire, en général, qu'elle est vraie ou fausse : cela dépend de la valeur de ses variables. Autrement dit, le problème **SAT** est un problème de résolution d'une équation, dont les variables sont à chercher dans les Booléens.

Exercice 3 – ☆ – Montrer que **SAT** $\in$ **EXPTIME**.

L'existence du problème **SAT** nous pose un problème. Nous sommes convaincus qu'il n'est pas dans **P**, mais nous sommes incapables de le prouver. Une autre conjecture est que **SAT** ne fait pas partie des « problèmes les plus durs de **EXPTIME** » (qu'on appelle problèmes **EXPTIME**-complets, nous les définirons plus tard). Au contraire, **SAT** est caractéristique d'une classe que nous n'avons pas encore définie : la classe **NP**. On pense que cette classe est strictement plus grande que **P**, mais strictement plus petite que **EXPTIME** (encore une fois ces deux affirmations sont des conjectures : nous sommes incapables de les prouver). Pour définir **NP**, nous devons changer notre modèle de machine de Turing et passer au **non-déterminisme**.

2 Complexité non-déterministe en temps

La définition des classes de complexité comme **NP** est basée sur les machines de Turing non-déterministes.

Remarque 5 - Rappel. La terminologie **non-déterministe** peut être trompeuse : une machine non-déterministe **peut** avoir plusieurs transitions possibles dans une configuration donnée, mais ce n'est pas obligatoire. Autrement dit, une machine de Turing déterministe est un cas particulier de machine de Turing non-déterministe. Dit encore autrement, une machine non-déterministe peut être déterministe.

Du point de vue de la calculabilité, les machines non-déterministes ont peu d'intérêt : ce qu'on peut calculer grâce au non-déterminisme peut aussi être calculé par machine déterministe. Par contre, les machines non-déterministes sont centrales en complexité. Définissons d'abord le temps d'exécution d'une machine non-déterministe.

Definition 6 - Temps d'exécution d'une machine de Turing non-déterministe.

On considère un alphabet Σ et une machine de Turing non-déterministe M dont Σ est l'alphabet d'entrée. Étant donnée une fonction $f : \mathbb{N} \to \mathbb{N}$, on dit que le temps d'exécution de M est borné par f si pour toute entrée $w \in \Sigma^*$, **toutes les exécutions** de M sur w terminent après **au plus** $f(|w|)$ transitions.

On peut maintenant définir les classes de complexité non-déterministes en temps.

Definition 7 - Classes de complexité non-déterministes en temps.

On considère une fonction $f : \mathbb{N} \to \mathbb{N}$. On note **NTIME**$(f)$ la classe de tous les langages L pour lesquels il existe des constantes $h, c \in \mathbb{N}$ telle que L est décidé par une machine de Turing non-déterministe dont le temps d'exécution est borné par la fonction $n \mapsto h \times f(n) + c$.

De façon similaire à ce que nous avons vu pour les classes déterministes, nous sommes surtout intéressés par quelques classes non-déterministes particulières.

Definition 8 - Temps non-déterministe polynomial.

On note **NPTIME** (ou simplement **NP**), la classe suivante :

$$\textbf{NPTIME} = \bigcup_{k \in \mathbb{N}} \textbf{NTIME}(n^k).$$

Definition 9 - Temps non-déterministe exponentiel.

On note **NEXPTIME**, la classe suivante :

$$\textbf{NEXPTIME} = \bigcup_{k \in \mathbb{N}} \textbf{NTIME}(2^{n^k}).$$

Exercice 4 – ❗☆ – Montrer que le problème **SAT** est dans **NP**.

On va maintenant comparer nos classes non-déterministes avec les classes déterministes vues précédemment.

Théorème 6

On a les inclusions suivantes : **P** $\subseteq$ **NP** $\subseteq$ **EXPTIME** $\subseteq$ **NEXPTIME**.

Soulignons que dans le Théorème 6, nous savons montrer que les deux inclusions **P** $\subseteq$ **EXPTIME** et **NP** $\subseteq$ **NEXPTIME** sont strictes. Par contre, le problème est **ouvert** pour toutes les autres inclusions : **P** $\subseteq$ **NP**, **NP** $\subseteq$ **EXPTIME** et **EXPTIME** $\subseteq$ **NEXPTIME**. On **conjecture** généralement que toutes ces inclusions sont strictes

et en conséquence, on représente souvent ces classes comme étant différentes (voir la Figure 2 ci-dessous par exemple). Cependant, il est important de garder en tête que ce ne sont que des conjectures. En particulier, savoir si $\mathbf{P} = \mathbf{NP}$ ou $\mathbf{P} \neq \mathbf{NP}$ est un des problèmes ouverts les plus célèbres en informatique.

Exercice 5 – ❗★ – Montrer le Théorème 6.

Enfin on termine par une remarque importante. Les classes non-déterministes sont (a priori) asymétriques. Par exemple, **SAT** $\in$ **NP**. Par contre, puisque dans une machine non-déterministe, les conditions d'acceptation (il existe une exécution qui accepte) et de rejet (toutes les exécutions rejettent) sont asymétriques, il n'est pas évident que le problème complémentaire « non-**SAT** » (étant donné une formule, est-ce qu'elle n'est **pas** satisfaisable) est dans **NP**. En fait, on conjecture que ce n'est pas le cas (mais on ne sait pas le prouver comme souvent en complexité). Cette discussion nous amène à la définitions des classes complémentaires.

Definition 10 - Classes complémentaires.

Soit $\mathscr{C}$ une classe de complexité. On note co-$\mathscr{C}$ la classe telle que pour tout alphabet Σ et tout langage $L \subseteq \Sigma^*$, on a $L \in$ co-$\mathscr{C}$ si et seulement si $\Sigma^* \setminus L \in \mathscr{C}$.

On représente la situation dans la Figure 2 ci-dessous (attention, elle est remplie de conjectures).

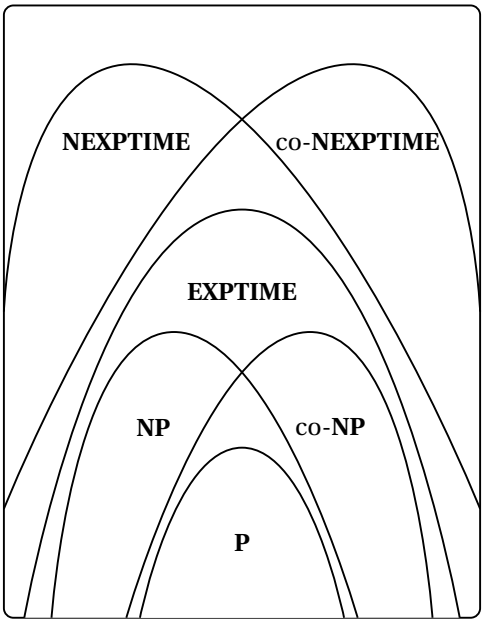


FIGURE 2 – Classes de complexité non-déterministes en temps

3 Bornes inférieures : problèmes difficiles et problèmes complets

On s'intéresse maintenant aux bornes inférieures. En théorie de la complexité, on souhaite relier la complexité aux langages (ou de manière équivalente aux problèmes de décision qu'ils codent), pas aux machines de Turing qui les décident. Par exemple, considérons un langage L et supposons que nous avons réussi à montrer que $L \in$ **NP** (**SAT** par exemple). C'est une propriété intéressante, mais elle n'est peut-être pas significative pour L : il se peut très bien que L appartienne à une classe plus petite (comme par exemple **P**). En d'autres termes, le fait que $L \in$ **NP** ne nous permet pas de dire que « **NP** est la complexité de L ».

Pour arriver à caractériser précisément la complexité d'un langage, il faut donner des *bornes inférieures* : on veut pouvoir énoncer en même temps que $L \in$ **NP et** « L n'est dans aucune classe plus petite que **NP** ».

Quand ces deux propriétés sont satisfaites simultanément, il est naturel de dire « **NP** est la complexité de L » (formellement on dira que L est **NP**-*complet*). Bien sûr, il faut définir formellement ce que veut dire « L n'est dans aucune classe plus petite que **NP** ». Pour cela nous recourrons à nouveau aux *réductions*.

Definition 11 - Réductions polynomiales.

Considérons un alphabet Σ et deux langages $K, L \subseteq \Sigma^*$. Une réduction polynomiale de K vers L est une fonction **calculable en temps polynomial** $f : \Sigma^* \to \Sigma^*$ telle que :

$$\text{Pour tout } w \in \Sigma^*, \quad w \in K \text{ si et seulement si } f(w) \in L.$$

Exercice 6 – ☆ – Prouver que les réductions polynomiales sont transitives : si il existe des réductions polynomiales de K vers L et L vers H, alors il en existe une de K vers H.

On peut maintenant définir ce qu'est un langage complet pour une classe de complexité donnée.

Definition 12 - Langages difficiles et langages complets pour les réductions polynomiales.

Soit $\mathscr{C}$ une classe de complexité et L un langage.

- On dit que L est $\mathscr{C}$*-difficile* pour les réductions polynomiales si et seulement si pour tout langage $K \in \mathscr{C}$, il existe une réduction polynomiale de K vers L.
- On dit que L est $\mathscr{C}$*-complet* pour les réductions polynomiales si et seulement si $L \in \mathscr{C}$ **et** L est $\mathscr{C}$-difficile.

Remarque 7 - Attention. Par souci de concision, nous omettrons souvent la mention « pour les réductions polynomiales » quand nous parlons de problèmes $\mathscr{C}$-difficiles et $\mathscr{C}$-complets. Cependant, cette condition est importante : il est possible de considérer des types de réduction plus restrictifs et changer ainsi ce que veut dire $\mathscr{C}$-difficile et $\mathscr{C}$-complet.
Par exemple, utiliser les réductions polynomiales n'a pas de sens pour définir **P**-difficile et **P**-complet : tous les langages non-triviaux dans **P** sont **P**-complets pour les réductions polynomiales (voir l'Exercice 9). Bien sûr, la notion de langage **P**-complet existe. Cependant elle est définie avec un type de réduction plus restrictif : les réductions en espace logarithmique que nous verrons plus tard.

Les problèmes complets pour une classe sont caractéristiques de celle-ci. En particulier, prouver un résultat pour un problème complet spécifique entraîne des conséquences pour toute la classe. On donne des exemples dans l'exercice suivant.

Exercice 7 – ☆ – On considère un langage L qui est **NP**-complet. Montrer que les deux propriétés suivantes sont équivalentes :

1. $L \in \mathbf{P}$.
2. $\mathbf{P} = \mathbf{NP}$.

Exercice 8 – ☆ – Montrer que tout langage L qui est **EXPTIME**-difficile, n'appartient pas à **P**.

Exercice 9 – ❗☆ – Soit Σ un alphabet et L un langage sur Σ. Montrer si L est non-trivial (c'est-à-dire différent de $\emptyset$ et Σ^*), il est **P**-difficile pour les réductions polynomiales.

4 Le théorème de Cook-Levin

Le théorème de Cook-Levin dit que **SAT** est un problème **NP**-complet. Historiquement, c'est le premier résultat majeur en théorie de la complexité et il reste aujourd'hui parmi les plus importants. En particulier, il nous fournit un premier exemple de problème **NP**-complet.

Théorème 8 — **Cook-Levin**

Le problème **SAT** est **NP**-complet (pour les réductions polynomiales).

Il nous reste maintenant à montrer le Théorème 8. Nous savons depuis l'exercice 4 que **SAT** est dans **NP**. D'après la définition, il nous reste nous à montrer que **SAT** est **NP**-difficile. C'est-à-dire que nous devons prouver que tout problème appartenant à **NP** se réduit polynomialement à **SAT**.

Remarque 9 - Réductions polynomiales génériques. Ici, nous allons devoir partir d'un problème *quelconque* dans **NP** et le réduire à **SAT**. Informellement, on parle d'une réduction « générique » vers **SAT**. À cet stade, nous n'avons pas le choix car nous ne connaissons encore problème **NP**-difficile. Plus tard, lorsque nous connaîtrons des problèmes **NP**-difficiles, nous disposerons d'une stratégie alternative consistant à réduire un problème **NP**-complet de notre choix parmi ceux que nous connaissons déjà (voir l'exercice 11 ci-dessous).
Par exemple, une fois le Théorème 8 prouvé, il suffira de réduire **SAT** (avec une réduction polynomiale) à un problème pour prouver que ce dernier est **NP**-difficile.

🎓 **Exercice 10 – ❗★★ – Preuve du théorème de Cook-Levin** On considère un alphabet Σ et un langage $L \subseteq \Sigma^*$ dans **NP**. Décrire une réduction polynomiale de L vers **SAT**.

5 La jungle des problèmes NP-complets

Les exercices de cette section consistent chacun à prouver qu'un problème particulier est **NP**-complet. Commençons par poser une question simple : comment prouve-t-on qu'un problème est **NP**-complet (et en particulier **NP**-difficile) ? On peut bien sûr procéder comme nous l'avons fait pour le Théorème 8 avec une réduction « générique ». Cependant maintenant que nous connaissons un premier problème **NP**-complet, il y a plus simple.

🎓 **Exercice 11 – ❗☆ –** On considère une classe de complexité $\mathscr{C}$. Soit L un langage quelconque et K un langage $\mathscr{C}$-complet. Montrer que les deux conditions suivantes sont équivalentes :

1. L est $\mathscr{C}$-difficile (pour les réductions polynomiales).
2. Il existe une réduction polynomiale de K vers L.

Remarque 10. La situation est très similaire à ce que nous avons rencontré en étudiant l'indécidabilité. Nous avons commencé par montrer « manuellement » qu'un premier problème était indécidable (**DIAG**). Ensuite, nous avons prouvé que d'autres problèmes étaient aussi indécidables en utilisant des réductions de **DIAG** vers ces autres problèmes.
Ici, nous avons commencé par montrer « manuellement » qu'un premier problème était **NP**-difficile (c'est-à-dire **SAT**). Nous allons maintenant prouver que d'autres problèmes sont aussi **NP**-difficiles en utilisant des réductions de **SAT** vers ces autres problèmes.
Il est cependant important de noter que le type de réduction utilisé n'est pas le même. Quand nous avons étudié l'indécidabilité, nous nous servions de réductions *calculables*. Ici, nous devons utiliser des réductions *polynomiales*.

5.1 Le problème 3-SAT

On va d'abord considérer une variante « plus restrictive » du problème **SAT**. C'est-à-dire qu'on restreint son entrée à des formules propositionnelles ayant une forme particulière.

Nous allons définir le problème **3-SAT** en deux étapes. Tout d'abord, on définit une variante plus générale : **CNF-SAT**. Il s'agit de la restriction de **SAT** aux formules propositionnelles en *forme normale conjonctive*. Commençons par définir cette notion.

Definition 13 - Littéraux et clauses.

Un *littéral* est une formule propositionnelle réduite à une variable (c'est à dire de la forme « x ») ou à la négation d'une variable (c'est à dire de la forme « $\neg x$ »).
Pour tout nombre $k \geqslant 1$, on appelle *k-clause* une disjonction de k littéraux : une formule propositionnelle de la forme $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_k$ où $\ell_1, \ldots, \ell_k$ sont des littéraux. Par exemple « $x \vee y \vee \neg z$ » est une 3-clause.
Enfin, on appelle *clause* une formule propositionnelle qui est une k-clause pour un certain $k \geqslant 1$.

Remarque 11. Le même littéral peut être répété dans une même clause. Par exemple « $\neg x \vee \neg x \vee y$ » est une 3-clause.

Definition 14 - Forme normale conjonctive.

On définit deux notions :

- Une formule propositionnelle en *forme normale conjonctive* (CNF) est une conjonction finie de clauses, c'est-à-dire de la forme $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ où $C_1, \ldots, C_n$ sont des clauses et $n \geqslant 1$.
- Pour tout $k \geqslant 1$, une formule propositionnelle en k-CNF est une conjonction finie de k-clauses, c'est-à-dire de la forme $C_1 \wedge C_2 \wedge \cdots \wedge C_n$ où $C_1, \ldots, C_n$ sont des k-clauses et $n \geqslant 1$.

Nous pouvons maintenant définit les problèmes **CNF-SAT** et **3-SAT**. Le premier est une restriction de **SAT** et le second est une restriction du premier.

Definition 15 - CNF-SAT.

Le problème **CNF-SAT** est le suivant :

ENTRÉE : φ une formule propositionnelle en CNF.
QUESTION : φ est-elle satisfaisable ?

Definition 16 - 3-SAT.

Le problème **3-SAT** est le suivant :

ENTRÉE : φ une formule propositionnelle en 3-CNF.
QUESTION : φ est-elle satisfaisable ?

Exercice 12 – ❗★ – NP-complétude de 3-SAT Montrer que **3-SAT** est **NP**-complet. Que peut-on en déduire vis-à-vis de **CNF-SAT** ?

Remarque 12. Maintenant que nous savons que **3-SAT** est également **NP**-complet, il suffit de réduire **3-SAT** à un problème pour montrer que ce dernier est **NP**-difficile. C'est un point important. Intuitivement, **3-SAT** est « plus simple » que **SAT** et on préfère donc que notre réduction parte de **3-SAT**.

5.2 Les problèmes de graphes

De nombreux exemples importants de problèmes **NP**-complets prennent des graphes en entrées. Sauf indication contraire, nous considérerons toujours des graphes *orientés*.

Definition 17 - Graphes orientés.

Un graphe orienté G est une paire $G = (V, E)$ où V est un ensemble fini appelé l'ensemble des *sommets* du graphe et $E \subseteq V \times V$ est un ensemble de paires de sommets appelé l'ensemble des *arêtes* du graphe.

Exercice 13 – ❶★ – Couverture par sommets Pour tout graphe $G = (V, E)$, un sous-ensemble de sommets $C \subseteq V$ est *couvrant* si et seulement si pour toute arête $(u, v) \in E$ du graphe au moins une de ses extrémités u ou v appartient à C. Le problème de couverture par sommets (noté **COUV**) est défini ainsi :

ENTRÉE : Un graphe $G = (V, E)$ et un entier $p \in \mathbb{N}$.
QUESTION : Existe-t-il un sous-ensemble couvrant $C \subseteq V$ tel que $|C| = p$.

1. Justifiez que **COUV** est dans **NP**.
2. On veut maintenant montrer que **COUV** est **NP**-difficile. On va réduire **3-SAT** à **COUV**. Il faut donc donner un algorithme polynomial qui, étant donné une formule 3-CNF φ, construit un graphe G et un entier p qui satisfont **COUV** si et seulement si φ est satisfaisable. Soient $x_1, \ldots, x_n$ les variables de φ.
   a) Soit G un graphe à $2n$ sommets $\{q_1, \ldots, q_n, r_1, \ldots, r_n\}$. Donner un ensemble d'arêtes pour lequel tout ensemble couvrant contient au moins r_i ou q_i pour tout $i \leqslant n$.
   b) On reprend le graphe G de la question précédente et on distingue trois sommets s_1, s_2, s_3. Compléter G avec de nouveaux sommets et arêtes et donner un entier p tel que tout ensemble couvrant de taille p du graphe résultant contient au moins l'un des trois sommets s_1, s_2, s_3.
3. Utiliser les questions précédentes pour donner une réduction polynomiale de **3-SAT** à **COUV**. Que peut-on en conclure sur **COUV**?

Pour les problèmes suivants nous aurons besoin de la notion de *sous-graphe* qu'on commence par définir.

Definition - Sous-graphe d'un graphe.

Soit $G = (V, E)$ un graphe. Une *sous-graphe* de G est un second graphe $H = (W, F)$ tel que $W \subseteq V$ et $F \subseteq E$. En d'autres termes, H est construit à partir de G en retirant (possiblement) des sommets et des arêtes.

Remarque 13. Cette notion diffère de celle de sous-graphe **induit** qui requiert de garder toutes les arêtes qui relient les sommets restants. Ici, on peut retirer autant de sommets et d'arêtes que l'on veut.

Exercice 14 – ❶★ – Clique et stable Soit $G = (V, E)$ un graphe. On dit que G est une *clique* (on parle aussi de graphe complet) si et seulement si tous ses sommets sont deux-à-deux adjacents (on a $(u, v) \in E$ pour tous $u, v \in V$). De façon symétrique, on dit que G est un *stable* (on parle aussi de graphe indépendant) si et seulement si tous ses sommets sont deux-à-deux non-adjacents (on a $(u, v) \notin E$ pour tous $u, v \in V$). On étudie le problème **CLIQUE** suivant :

ENTRÉE : Un graphe G et un entier $q \in \mathbb{N}$.
QUESTION : Existe-t-il un sous-graphe de taille q de G qui est une *clique*?

et le problème **STABLE** suivant :

ENTRÉE : Un graphe G et un entier q.
QUESTION : Existe-t-il un sous-graphe de taille q de G qui est un *stable*?

1. Montrer que **CLIQUE** et **STABLE** se réduisent mutuellement l'un à l'autre.
2. Montrer que **COUV** et **STABLE** se réduisent mutuellement l'un à l'autre.
3. Montrer que **CLIQUE** et **STABLE** sont **NP**-complets.

Pour l'exercice suivant nous allons avoir besoin de la notion d'*isomorphisme* entre deux graphes.

Definition 19 - Isomorphisme de graphes.

Étant donné deux graphes $G=(V,E)$ et $H=(W,F)$, on dit que G et H sont *isomorphes* si et seulement si il existe une bijection $\alpha : V \to W$ telle que pour tout $u, v \in V$, on a $(u,v) \in E \Leftrightarrow (\alpha(u), \alpha(v)) \in F$.

Exercice 15 – ❗★ – Sous-graphe On considère le problème **SGRAPHE** suivant :

ENTRÉE : Deux graphes G et H.
QUESTION : Existe-t-il un sous-graphe de G isomorphe à H ?

1. Montrer que **SGRAPHE** est **NP**-complet.

On considère maintenant le problème **ISO** suivant :

ENTRÉE : Deux graphes G et H.
QUESTION : Ces deux graphes sont-ils isomorphes l'un à l'autre ?

2. Montrer que **ISO** se réduit à **SGRAPHE**.
3. Que peut-on en déduire sur **ISO** ?

Remarque 14. Savoir si **ISO** est **NP**-complet ou non est un problème *ouvert*. Une conviction générale est que ce problème est de *complexité intermédiaire* : plus compliqué que polynomial mais moins compliqué que **NP**-complet (tout cela, bien sûr, si **P** $\neq$ **NP**).

Exercice 16 – ❗★ – 3-Colorabilité Étant donnés un graphe $G=(V,E)$ et un entier $k \in \mathbb{N}$, un *k-coloriage* de G est une partition de l'ensemble de sommets V en k ensembles (intuitivement, chaque ensemble de la partition est associée à une couleur utilisée pour colorier tous ses sommets) telle que pour toute arête $(u,v) \in E$, les sommets u et v sont dans deux ensembles distincts de la partition (autrement dit, deux sommets adjacent doivent avoir une couleur différente).
Pour tout $k \in \mathbb{N}$, on définit le problème de k-colorabilité (noté k-**COL**) de la façon suivante :

ENTRÉE : Un graphe G.
QUESTION : Existe-t-il un k-coloriage de G ?

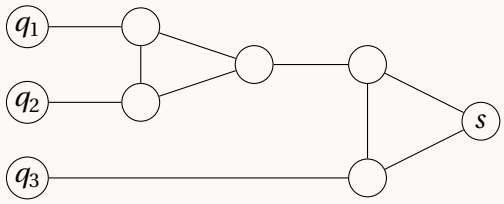
Dans l'exercice, on va surtout s'intéresser à **3-COL**. Commençons d'abord par regarder **1-COL** et **2-COL**.

1. Montrer que **1-COL** et **2-COL** sont dans **P**.
2. Montrer que **3-COL** est dans **NP**.
3. On va maintenant montrer que **3-COL** est **NP**-difficile. Pour cela, on va réduire **3-SAT** à **3-COL**. Soit φ une formule 3-CNF. On note $x_1, \dots, x_n$ les variables de φ.

   a) On considère un graphe G, qui, pour chaque variable contient un sommet x_i et un sommet $\neg x_i$. Compléter le graphe pour que tout 3-coloriage de celui-ci colorie les sommets $x_1, \neg x_1, \dots, x_n, \neg x_n$ avec seulement deux couleurs et ne donne jamais la même couleur à x_i et $\neg x_i$ pour tout i.

   Note : Intuitivement, on est en train de coder les affectations de variables $x_1, \dots, x_n$ par les 3-coloriages de notre graphe.

b) On doit maintenant modifier le graphe pour que les 3-coloriages possibles sont uniquement ceux qui codent une affectation de variables qui satisfait la formule 3-CNF φ. Considérons le graphe suivant :



Montrer que dans tout 3-coloriage de ce graphe, si q_1, q_2, q_3 ont la même couleur, alors s est aussi de cette couleur. Ensuite, montrer que si q_1, q_2, q_3 ont au moins deux couleurs associées, alors s peut avoir aussi une de ces couleurs associées.

c) Donner une réduction de **3-SAT** à **3-COL**. En déduire que **3-COL** est **NP**-complet.

4. Montrer que k-**COL** est **NP**-complet pour tout $k \geqslant 3$.

🎓 **Exercice 17 – ❶★★☆ – Chemin Hamiltonien** Un *chemin Hamiltonien* dans un graphe G est un chemin qui passe une et une seule fois par chaque sommet du graphe. On appelle **CHEMIN** le problème associé :

ENTRÉE	:	Un graphe orienté G.
QUESTION	:	Existe-t-il un chemin Hamiltonien dans G ?

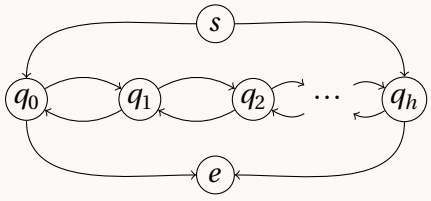
Un *circuit Hamiltonien* est un chemin Hamiltonien fermé, c'est-à-dire qu'il existe une arrête entre le premier sommet du chemin et le dernier. On appelle **CIRCUIT** le problème associé :

ENTRÉE	:	Un graphe orienté G.
QUESTION	:	Existe-t-il un circuit Hamiltonien dans G ?

1. Donner un exemple d'un graphe qui contient un chemin Hamiltonien mais pas de circuit Hamiltonien.
2. Montrer que **CHEMIN** se réduit à **CIRCUIT**.
3. Montrer que **CIRCUIT** se réduit à **CHEMIN**.

On cherche maintenant à montrer que **CHEMIN** est **NP**-complet. On va réduire **3-SAT**, soit φ un formule 3-CNF et $x_1, \ldots, x_n$ les variables et h le nombre de clauses.

1. On considère le graphe G suivant à $h+3$ sommets :



Montrer qu'il n'existe que deux chemins Hamiltoniens possibles et les donner.

2. En utilisant la question précédente donner un graphe tel que chaque chemin Hamiltonien dans celui-ci code une affectation des variables $x_1, \ldots, x_n$.
3. Complétez le graphe avec de nouveaux sommets et arrêtes pour interdire les chemins Hamiltoniens qui codent des affectations qui ne satisfont pas φ.
4. En déduire une réduction de **3-SAT** à **CHEMIN** et conclure que **CHEMIN** et **CIRCUIT** sont **NP**-complets.
5. On considère **CHEMIN** et **CIRCUIT** pour des graphes non-orientés. Montrez que les deux problèmes restent **NP**-complets.

5.3 Somme d'entiers

Exercice 18 – ❶★★ – Somme d'entiers On considère le problème **SOMME** suivant :

ENTRÉE : Des entiers positifs $v_1, v_2, \ldots, v_n \in \mathbb{N}$ et un entier $S \in \mathbb{N}$.
QUESTION : Existe-t-il une sous-suite $1 \leqslant i_1 < i_2 < \ldots < i_p \leqslant n$ telle que $v_{i_1} + v_{i_2} + \ldots + v_{i_p} = S$?

1. Proposer un algorithme qui résout le problème **SOMME** avec un temps de calcul en $O(n \cdot S)$.
2. Il est connu que le **SOMME** est **NP**-complet (propriété que nous allons montrer ensuite). Vient-on de prouver avec la première question que **P** = **NP** ? Justifier.
3. Justifier que **SOMME** est dans **NP**.
4. On veut maintenant montrer que **SOMME** est en effet **NP**-difficile. On va réduire **3-SAT** à **SOMME**. Nous devons décrire un algorithme polynomial qui, étant donné une formule **3-SAT**,

$$\varphi = (\ell_{1,1} \vee \ell_{1,2} \vee \ell_{1,3}) \wedge \ldots \wedge (\ell_{m,1} \vee \ell_{m,2} \vee \ell_{m,3})$$

produit une suite d'entiers $v_1, \ldots, v_n$ et un entier S tels que S est la somme d'un sous-ensemble d'entiers parmi $v_1, \ldots, v_n$ si et seulement si φ est satisfaisable. Nous présentons une idée de construction par un exemple. Vous devrez ajouter les détails manquants pour le cas général.

On considère la formule

$$\varphi = \underbrace{(x_1 \vee x_2 \vee \neg x_3)}_{c_1} \wedge \underbrace{(\neg x_1 \vee x_2 \vee \neg x_3)}_{c_2}.$$

D'abord, nous introduisons des entiers $v_1, \bar{v}_1, v_2, \bar{v}_2, v_3, \bar{v}_3$ qui représentent les occurrences de chaque littéral (positive ou négative) dans les clauses c_1 et c_2 de φ. On donne les valeurs de $v_1, \bar{v}_1, v_2, \bar{v}_2, v_3, \bar{v}_3$ en utilisant leur représentation décimale, comme indiqué dans la table suivante :

	x_1	x_2	x_3	c_1	c_2
v_1	1	0	0	1	0
$\bar{v}_1$	1	0	0	0	1
v_2	0	1	0	1	1
$\bar{v}_2$	0	1	0	0	0
v_3	0	0	1	0	0
$\bar{v}_3$	0	0	1	1	1

Que peut-on dire à propos de tout sous-ensemble de $\{v_1, \bar{v}_1, v_2, \bar{v}_2, v_3, \bar{v}_3\}$ dont la somme est de la forme $S = 1\,1\,1 - -$, où $--$ sont deux chiffres différents de 0 (c'est-à-dire $\geqslant 1$, on est en base 10) ?

Enfin, nous ajoutons deux autres entiers pour chaque clause, c'est-à-dire :

	x_1	x_2	x_3	c_1	c_2
w_1	0	0	0	1	0
w_2	0	0	0	1	0
w_3	0	0	0	0	1
w_4	0	0	0	0	1

Montrez qu'il existe un sous-ensemble de $\{v_1, \bar{v}_1, v_2, \bar{v}_2, v_3, \bar{v}_3, w_1, w_2, w_3, w_4\}$ dont la somme est l'entier $S = 1\,1\,1\,3\,3$ si et seulement si φ est satisfaisable.

5. Généraliser la construction de la question précédente afin de montrer que **SOMME** est **NP**-complet.