Théorie de la Complexité – Feuille de cours intégré n° 5 à 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 $|a\,a\,b| = 3$).

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.

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 $n \mapsto h \times f(n) + c$.

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 celles qui sont les plus standard.

Temps polynomial

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

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

Temps exponentiel

On note **EXPTIME** la classe suivante :

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

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

Théorème 1

On a les propriétés suivantes : $\mathbf{P} \subseteq \mathbf{EXPTIME}$ et $\mathbf{P} \neq \mathbf{EXPTIME}$.

Le Théorème 1 énonce une inclusion stricte entre les classes **P** 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 Prouver le Théorème 1. Pour montrer que $\mathbf{P} \neq \mathbf{EXPTIME}$, il « suffit » de trouver un problème dans **EXPTIME** et de prouver qu'il n'est pas dans **P**. On pourra s'inspirer de ce que nous avons fait en calculabilité et du premier problème indécidable que nous avons vu. ▪

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 :

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

De la même façon, on peut définir 3–**EXPTIME**, 4–**EXPTIME**,… En adaptant les arguments servant à prouver le Théorème 1, on peut montrer qu'on obtient une hiérarchie stricte et infinie de classes :

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

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

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

On représente la situation graphiquement dans la Figure 1.

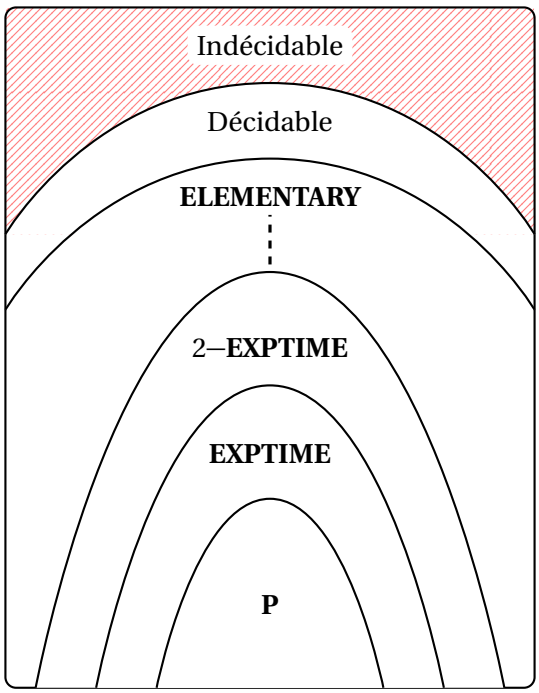


FIGURE 1 – Classes de complexité déterministes en temps

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.

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 ?

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.

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.

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.

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.

Temps non-déterministe polynomial

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

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

Temps non-déterministe exponentiel

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

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

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 2

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

Soulignons que dans le Théorème 2, 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 **P** $=$ **NP** ou **P** $\neq$ **NP** est un des problèmes ouverts les plus célèbres en informatique.

Exercice 5 Montrer le Théorème 2. ∎

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.

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).

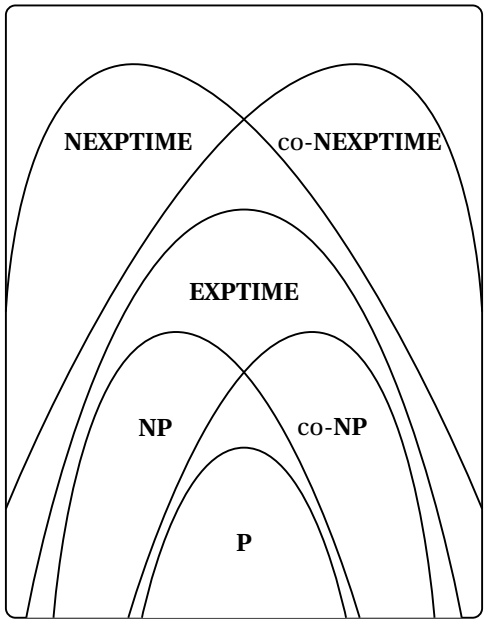


FIGURE 2 – Classes de complexité non-déterministes en temps

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 les propriétés $L \in$ **NP et** « L n'est dans aucune classe plus petite que **NP** ». Quand ces deux propriétés sont satisfaites simultanément, il devient 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 nouveau aux *réductions*.

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.

Langages difficiles, langages complets

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.

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 réductions 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 Montrer que tout langage L non-trivial est **P**-difficile pour les réductions polynomiales. ∎

On va maintenant montrer le Théorème de Cook-Levin qui dit que **SAT** est un problème **NP**-complet. En fait, on va même montrer qu'un problème plus restreint est **NP**-complet : CNF-**SAT**.

CNF-SAT

CNF-**SAT** est la variante de **SAT** restreinte aux formules en normale conjonctive. On commence par définir cette notion.

Un *littéral* est soit une variable (x), soit la négation d'une variable ($\neg x$). On appelle *clause* une formule propositionnelle qui est une disjonction de littéraux ($p_1 \vee p_2 \vee \cdots \vee p_n$ où $p_1, \ldots, p_n$ sont des littéraux). Enfin une formule propositionnelle en *forme normale conjonctive* (CNF) est une conjonction de clauses ($C_1 \wedge C_2 \wedge \cdots \wedge C_k$ où $C_1, \ldots, C_k$ sont des clauses). Le problème CNF-**SAT** est le suivant :

ENTRÉE : φ une formule propositionnelle en forme normale conjonctive.
QUESTION : φ est-elle satisfaisable ?

Théorème 3 (Cook-Levin)

Le problème CNF-**SAT** est **NP**-complet.

Exercice 10 Prouver le Théorème 3. Puisqu'on sait déjà que CNF-**SAT** est dans **NP**, il s'agit de prouver qu'il est **NP**-difficile. Étant donné un problème quelconque dans **NP**, on doit donner une réduction polynomiale de celui-ci vers CNF-**SAT**. ∎

4 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 3. 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.
2. il existe une réduction polynomiale de K vers L.

∎

Remarque

La situation est très similaire à ce que nous avons rencontré en étudiant l'indécidabilité. Nous avons dû commencer par montrer manuellement qu'un premier problème était indécidable (**HALT**). Ensuite, nous en avons trouvé d'autres en utilisant des réductions de **HALT** vers ces problèmes.

Ici, nous avons dû commencer par montrer manuellement qu'un premier problème était **NP**-complet (CNF-**SAT**). Nous pouvons maintenant montrer que de nouveaux problèmes sont **NP**-complets en montrant qu'ils sont dans **NP** et que CNF-**SAT** se réduit polynomialement à ces problèmes.

Exercice 12 — SAT. Montrer que le problème **SAT** est **NP**-complet (pour l'instant nous n'avons le résultat que pour CNF-**SAT**). ∎

Exercice 13 — 3-SAT. On considère une variante de **SAT** encore plus restrictive que CNF-**SAT**. On dit qu'une formule propositionnelle est 3-**SAT** si elle est en forme normale conjonctive et que toutes ses clauses contiennent au plus trois littéraux. Le problème 3-**SAT** est le suivant :

ENTRÉE : φ une formule propositionnelle 3-**SAT** en forme normale conjonctive.
QUESTION : φ est-elle satisfaisable ?

Montrer que 3-**SAT** est **NP**-complet. ∎

Exercice 14 — Couverture par sommets. Soit $G=(V,E)$ un graphe, un sous-ensemble de sommets $C \subseteq V$ est dit *couvrant* si pour toute arête $(u,v) \in E$ au moins une de ses extrémités u ou v appartient à C. Le problème **Couverture par sommets** est défini ainsi :

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

On veut montrer que **Couverture par sommets** est **NP**-complet.

1. Justifiez que **Couverture par sommets** est dans **NP**.
2. Il faut maintenant montrer que **Couverture par sommets** est **NP**-difficile. On va réduire 3- **SAT** (à **Couverture par sommets**). Il faut donc donner un algorithme polynomial qui, étant donné une formule 3-**SAT** φ, construit un graphe G et un entier p qui satisfont **Couverture par sommets** si et seulement si φ est satisfaisable. On appelle $x_1,\dots,x_n$ les variables de φ.
   (a) Soit G un graphe à $2n$ sommets $\{q_1,\dots,q_n,r_1,\dots,r_n\}$, donner un ensemble d'arêtes qui garantit que 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 **Couverture par sommets**. Conclure que **Couverture par sommets** est **NP**-complet. ∎

Exercice 15 En théorie des graphes, une **clique** (appelée aussi graphe complet) est un ensemble de sommets deux à deux *adjacents*. De façon similaire, un **stable** (appelé aussi ensemble indépendant) est un ensemble de sommets deux à deux *non adjacents*.

On étudie le problème **Clique** :

ENTRÉE : Un graphe G et un entier q.
QUESTION : Existe-t-il un ensemble de q sommets de G dont le graphe induit est une *clique*?

et le problème **Stable** :

ENTRÉE : Un graphe G et un entier q.
QUESTION : Existe-t-il un ensemble de q sommets de G dont le graphe induit est un *stable*?

1. Montrer que **Clique** et **Stable** se réduisent mutuellement l'un à l'autre.
2. Montrer que **Couverture par sommets** et **Stable** se réduisent mutuellement l'un à l'autre.
3. Montrer que **Clique** et **Stable** sont **NP**-complets. ∎

Exercice 16 Le problème **Sous-graphe** est le suivant :

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

Le problème **Isomorphisme de graphes** est le suivant :

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

1. Montrer que **Clique** se réduit à **Sous-graphe**.

2. Montrer que **Sous-graphe** est **NP**-complet.
3. Montrer que **Isomorphisme de graphes** se réduit à **Sous-graphe**.

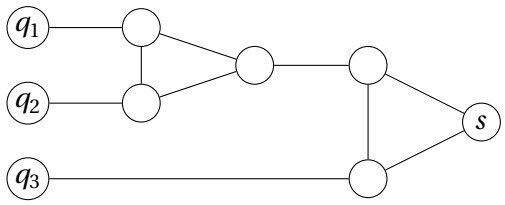
Remarque. Savoir si **Isomorphisme de graphes** est **NP**-complet ou non est un problème ouvert. Une conviction générale est que ce problème est de *complexité intermédiaire* : plus compliqué que polynomial mais moins compliqué que **NP**-complet (tout cela, bien sûr, si $\mathbf{P} \neq \mathbf{NP}$).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

Exercice 17 Considérons le problème **3-Colorabilité** suivant :

ENTRÉE	:	Un graphe $G = (V, E)$.
QUESTION	:	Existe-t-il un coloriage des sommets de G en 3 couleurs tel que les sommets adjacents ont des couleurs différentes ?

Il est possible de définir de manière analogue des problèmes de k-coloration pour tout entier k. Le but de l'exercice est de montrer que **3-Colorabilité** est **NP**-complet.

1. Montrer que **2-Colorabilité** est dans **P**.
2. Montrer que **3-Colorabilité** est dans **NP**.
3. On cherche à réduire **3-SAT** à **3-Colorabilité**, soit φ une instance de **3-SAT** et $x_1, \ldots, x_k$ les variables des clauses de φ.
   (a) On créé 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 $x_1, \neg x_1, \ldots, x_n, \neg x_n$ avec seulement deux couleurs et ne donne jamais la même couleur à x_i et $\neg x_i$ pour tout i. (i.e. un 3-coloriage donne une affectation des variables de φ).
   (b) On doit maintenant modifier le graphe pour qu'il n'accepte que des 3-coloriages qui correspondent à des affectations qui satisfont 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-Colorabilité**. En déduire que **3-Colorabilité** est **NP**-complet.
4. soit $k \geqslant 3$, montrer que k-**Colorabilité** est **NP**-complet. ▪

Exercice 18 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 Hamiltonien** le problème associé :

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

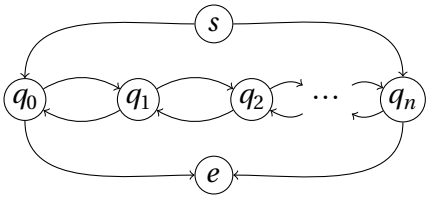
Un *circuit Hamiltonien* est un chemin Hamiltonien fermé, c'est-à-dire qu'il existe une arrête entre le premier sommet du chemin et le dernier. On appelle **Chemin Hamiltonien** le problème suivant :

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 ne contient pas de circuit Hamiltonien.
2. Montrer que **Chemin Hamiltonien** se réduit à **Circuit Hamiltonien**.
3. Montrer que **Circuit Hamiltonien** se réduit à **Chemin Hamiltonien**.

On cherche maintenant à montrer que **Chemin Hamiltonien** est **NP**-complet. On va réduire **3-SAT**, soit φ une instance de **3-SAT**, $x_1, \ldots, x_k$ les variables et n le nombre de clauses.

1. On considère le graphe G suivant à $n+3$ 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 correspond à une affectation des variables $x_1, \ldots, x_k$ à une valeur de vérité.
3. Complétez le graphe avec de nouveaux sommets et arrêtes pour interdire les chemins Hamiltoniens qui correspondent à des affectations qui ne satisfont pas φ.
4. En déduire que **Chemin Hamiltonien** et **Circuit Hamiltonien** sont **NP**-complets.
5. On considère **Chemin Hamiltonien** et **Circuit Hamiltonien** pour des graphes non-orientés. Montrez que les deux problèmes restent **NP**-Complets. ▪

Exercice 19 On considère le problème **Somme d'Entiers** suivant :

ENTRÉE : Des entiers positifs $v_1, v_2, \ldots, v_n$ et un entier S.
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. On suppose que la séquence $v_1, \ldots, v_n$ est déjà triée $v_1 \leqslant v_2 \leqslant \ldots \leqslant v_n$. Pour le problème **Somme d'Entiers** proposer un algorithme qui utilise une mémoire de taille $\mathcal{O}(S)$ et dont le temps de calcul est $\mathcal{O}(n \cdot S)$.
2. Il est connu que le problème **Somme d'Entiers** est **NP**-complet. Pourquoi l'existence de l'algorithme de la question 1. ne contredit pas la **NP**-complétude du problème ?
3. On veut maintenant montrer que le problème **Somme d'Entiers** est en effet **NP**-difficile. On va réduire 3-**SAT** à **Somme d'Entiers**.

   Il faut 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 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 $S = 1\,1\,1\,3\,3$ si et seulement si φ est satisfaisable. ∎