

Décidabilité et classes de complexité

Quentin Fortier

November 30, 2025

Définition

Un algorithme est une suite d'instructions déterministes (sans aléatoire) avec une entrée finie.

On suppose que cet algorithme s'exécute sur un ordinateur avec une quantité illimitée de mémoire.

Sur une entrée, un algorithme peut :

- renvoyer un résultat en temps fini
- ne pas renvoyer de résultat, soit parce qu'il ne termine pas (boucle infinie...), soit parce qu'il plante (dépassement de tableau...).

Définition

Une machine universelle est un programme U prenant en entrée le code source d'un programme f et un argument x , et simulant l'exécution de f sur x . Autrement dit :

- si $f(x)$ renvoie un résultat y en temps fini, alors $U(f, x)$ renvoie y en temps fini.
- si $f(x)$ déclenche une erreur, $U(f, x)$ déclenche une erreur.
- si $f(x)$ ne termine pas, $U(f, x)$ ne termine pas.

Exemple : l'interpréteur OCaml (utop) est une machine universelle pour les programmes OCaml.

Attention : `let universal f x = f x` n'est pas une machine universelle, car ce n'est pas le code source de `f` qui est donné en argument mais une fonction. Cependant, par simplicité de notation, on se permettra parfois d'écrire `f x` au lieu de `universal f x`.

Définition

Un problème de décision est un couple (I, I^+) tel que :

- I est l'ensemble des instances du problème
- $I^+ \subset I$ est l'ensemble des instances positives du problème

On peut aussi définir un problème sous forme d'une question binaire sur une instance.

Exemples de problèmes de décision :

SAT

- Instance : une formule logique φ .
- Question : φ est-elle satisfiable ?

$I = \{\varphi \mid \varphi \text{ formule logique}\}, I^+ = \{\varphi \mid \varphi \text{ satisfiable}\}$

APPARTIENT

- Instance : un mot w et un automate A .
- Question : $w \in L(A)$?

$I = \{(w, A) \mid w \text{ mot et } A \text{ automate}\}, I^+ = \{(w, A) \mid w \in L(A)\}$

Définition

Un problème de décision (I, I^+) est dit décidable s'il existe un algorithme qui :

- prend une instance $i \in I$ du problème en entrée
- renvoie **true** en temps fini si i est une instance positive ($i \in I^+$)
- renvoie **false** en temps fini si i est une instance négative ($i \notin I^+$)

Sinon, le problème est dit indécidable.

Remarques :

- Cet algorithme peut être écrit en pseudo-code ou dans un langage de programmation (OCaml, C...).
- L'algorithme doit terminer en temps fini sur toutes les instances.
- Pour montrer qu'un problème est décidable, il suffit d'exhiber un algorithme qui le décide.
Montrer qu'il est indécidable est a priori plus difficile : il faut montrer qu'aucun algorithme ne peut le résoudre.
- Seule l'existence d'un algorithme importe pour montrer la décidabilité : sa complexité n'a aucune importance. Il n'est donc pas non plus nécessaire de préciser les structures de données utilisées, tant que celles-ci sont calculables.
- Si I^+ est fini, le problème est trivialement décidable : il suffit d'énumérer I^+ et tester si l'une d'entre elles est égale à l'entrée.

Exercice

Montrer que le problème SAT est décidable.

Définition

Une fonction $f : E \rightarrow F$ est calculable s'il existe un algorithme A qui, pour tout élément $x \in E$, termine en temps fini et renvoie $f(x)$.

ARRET

- Instance : le code source d'un programme f et un argument x .
- Question : f termine-t-il sur l'entrée x ?

ARRET

- Instance : le code source d'un programme f et un argument x .
- Question : f termine-t-il sur l'entrée x ?

En OCaml, cela revient à écrire une fonction

`arret : string -> string -> bool` qui termine sur toute entrée et telle que si f est une fonction et x une chaîne de caractères, `arret f x` renvoie `true` si f termine sur x , `false` sinon.

ARRET

- Instance : le code source d'un programme f et un argument x .
- Question : f termine-t-il sur l'entrée x ?

En OCaml, cela revient à écrire une fonction

`arret : string -> string -> bool` qui termine sur toute entrée et telle que si f est une fonction et x une chaîne de caractères, `arret f x` renvoie `true` si f termine sur x , `false` sinon.

Théorème

Le problème de l'arrêt est indécidable.

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq \Pi_2$, s'il existe une fonction calculable $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \iff f(i) \in I_2^+$$

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq \Pi_2$, s'il existe une fonction calculable $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \iff f(i) \in I_2^+$$

Intuitivement : $\Pi_1 \leq_p \Pi_2$ signifie qu'on peut transformer une instance de Π_1 en une instance de Π_2 en complexité polynomiale, en préservant la réponse (oui/non). Un algorithme polynomial pour Π_2 permet alors de résoudre Π_1 en complexité polynomiale.

Exercice

Montrer que $\text{ACCESSIBLE} \leq \text{CHEMIN}$.

ACCESSIBLE

- Instance : un graphe $G = (S, A)$ et deux sommets $s, t \in S$.
- Question : existe-t-il un chemin de s à t dans G ?

CHEMIN

- Instance : un graphe $G = (S, A)$, $s, t \in S$, $k \in \mathbb{N}$.
- Question : existe-t-il un chemin de s à t dans G de longueur au plus k ?

Théorème

Soient Π_1 et Π_2 deux problèmes de décision tels que $\Pi_1 \leq \Pi_2$. Alors :

- Si Π_1 est indécidable, alors Π_2 est indécidable.
- Si Π_2 est décidable, alors Π_1 est décidable.

Théorème

Soient Π_1 et Π_2 deux problèmes de décision tels que $\Pi_1 \leq \Pi_2$. Alors :

- Si Π_1 est indécidable, alors Π_2 est indécidable.
- Si Π_2 est décidable, alors Π_1 est décidable.

Méthode pour montrer qu'un problème Π_1 est indécidable :

- 1 Supposer par l'absurde que Π_1 est décidable par une fonction f_1 .
- 2 En déduire une fonction f_2 qui décide un problème Π_2 connu comme indécidable (par exemple ARRET).
- 3 Conclure que Π_2 est indécidable.

Exercice

Montrer que le problème ARRET-VIDE est indécidable.

ARRET-VIDE

- Instance : une fonction f .
- Question : f termine-t-il sur l'entrée vide ?

Définition

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Définition

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Remarques :

- Un entier n est codé en base 2, donc sa taille est $\log_2(n)$. On pourrait aussi le coder en unaire ce qui donnerait une taille n , mais ce n'est pas « raisonnable ».
- On s'intéresse seulement à l'ordre de grandeur de la taille ($O(\dots)$).

Définition

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Remarques :

- Un entier n est codé en base 2, donc sa taille est $\log_2(n)$. On pourrait aussi le coder en unaire ce qui donnerait une taille n , mais ce n'est pas « raisonnable ».
- On s'intéresse seulement à l'ordre de grandeur de la taille ($O(\dots)$).

Exemples :

- La taille d'un entier n est

Définition

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Remarques :

- Un entier n est codé en base 2, donc sa taille est $\log_2(n)$. On pourrait aussi le coder en unaire ce qui donnerait une taille n , mais ce n'est pas « raisonnable ».
- On s'intéresse seulement à l'ordre de grandeur de la taille ($O(\dots)$).

Exemples :

- La taille d'un entier n est $\log_2(n)$.
- Un ensemble de p entiers dans $\llbracket 1, n \rrbracket$ a une taille de

Définition

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Remarques :

- Un entier n est codé en base 2, donc sa taille est $\log_2(n)$. On pourrait aussi le coder en unaire ce qui donnerait une taille n , mais ce n'est pas « raisonnable ».
- On s'intéresse seulement à l'ordre de grandeur de la taille ($O(\dots)$).

Exemples :

- La taille d'un entier n est $\log_2(n)$.
- Un ensemble de p entiers dans $\llbracket 1, n \rrbracket$ a une taille de $p \log_2(n)$.
- Un graphe à n sommets représenté par une matrice d'adjacence a une taille de

Définition

La taille $|x|$ d'une instance x d'un problème est le nombre de bits nécessaires pour la coder.

Remarques :

- Un entier n est codé en base 2, donc sa taille est $\log_2(n)$. On pourrait aussi le coder en unaire ce qui donnerait une taille n , mais ce n'est pas « raisonnable ».
- On s'intéresse seulement à l'ordre de grandeur de la taille ($O(\dots)$).

Exemples :

- La taille d'un entier n est $\log_2(n)$.
- Un ensemble de p entiers dans $\llbracket 1, n \rrbracket$ a une taille de $p \log_2(n)$.
- Un graphe à n sommets représenté par une matrice d'adjacence a une taille de n^2 .

Classes de complexité : P

Définition

La classe P est l'ensemble des problèmes de décision qui admettent un algorithme de complexité polynomiale en la taille de l'entrée (c'est-à-dire $O(n^k)$ pour une constante k , où n est la taille de l'entrée).

Classes de complexité : P

Définition

La classe P est l'ensemble des problèmes de décision qui admettent un algorithme de complexité polynomiale en la taille de l'entrée (c'est-à-dire $O(n^k)$ pour une constante k , où n est la taille de l'entrée).

Définition (HP)

La classe EXP est l'ensemble des problèmes de décision qui admettent un algorithme de complexité exponentielle en la taille de l'instance (c'est-à-dire $O(2^{n^k})$ pour une constante k , où n est la taille de l'entrée).

Remarque : $P \subset \text{EXP} \subset \text{Décidables}$.

Exemple :

PGCD

- Instance : entiers a , b , d .
- Question : d est-il le PGCD de a et b ?

On peut calculer le PGCD de a et b en utilisant l'algorithme d'Euclide en complexité $O(\log_2(a) + \log_2(b))$, polynomiale en la taille de l'entrée.

Classes de complexité : P

Exemple :

PREMIER

- Instance : un entier n .
- Question : n est-il premier ?

On peut énumérer les entiers de 2 à \sqrt{n} pour tester si n est divisible par l'un d'entre eux, en complexité $O(\sqrt{n})$.

Ceci est polynomial en n mais exponentielle en la taille $\log_2(n)$ de n (car $\sqrt{n} = 2^{\frac{\log_2(n)}{2}}$), ce qui montre seulement $\text{PREMIER} \in \text{EXP}$.

Exercice

- 1 Soit k un entier fixé. Montrer que k -CLIQUE \in P.
- 2 Montrer que CLIQUE \in EXP.

k -CLIQUE

- Instance : un graphe G .
- Question : G contient-il une clique de taille k , c'est-à-dire un sous-graphe complet de G de taille k ?

CLIQUE

- Instance : un graphe G et un entier k .
- Question : G contient-il une clique de taille k ?

Définition

Un problème de décision $\Pi = (I, I^+)$ appartient à la classe NP s'il existe :

- un algorithme de décision A prenant en entrée un couple (x, c) où $x \in I$
- un polynôme Q

tels que :

- A s'exécute en temps polynomial en $|x| + |c|$
- $\forall x \in I, x \in I^+ \Leftrightarrow \exists c, |c| \leq Q(|x|), A(x, c) = \text{true}$

Autrement dit : Π appartient à NP si, pour chaque instance positive x de Π , il existe un certificat c de taille polynomiale en $|x|$ qui permet de vérifier en temps polynomial en $|x| + |c|$ que x est une instance positive de Π .

En pratique : 99% du temps, si le problème est de la forme « Existe-t-il S tel que ... ? », S peut être choisi comme certificat. Il faut alors justifier que S est de taille polynomiale et qu'on peut vérifier que c'est une solution en complexité polynomiale.

Classes de complexité : NP

Exercice

Montrer que les problèmes suivants appartiennent à NP :

CLIQUE

- Instance : un graphe G et un entier k .
- Question : G contient-il une clique de taille k ?

SAT

- Instance : une formule logique φ en forme normale conjonctive.
- Question : φ est-elle satisfiable ?

co-FACTORISATION

- Instance : deux entiers n et p .
- Question : n ne possède-t-il aucun diviseur d tel que $1 < d < p$?

Théorème

$P \subset NP$.

Remarque : La question « $P = NP$? » est un des problèmes ouverts les plus célèbres en informatique.

Théorème

$P \subset NP$.

Remarque : La question « $P = NP$? » est un des problèmes ouverts les plus célèbres en informatique.

Exercice

Montrer que $NP \subset EXP$.

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit polynomialement à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq_p \Pi_2$, s'il existe une fonction calculable en complexité polynomiale $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \iff f(i) \in I_2^+$$

Définition

On dit qu'un problème de décision $\Pi_1 = (I_1, I_1^+)$ se réduit polynomialement à un problème de décision $\Pi_2 = (I_2, I_2^+)$, noté $\Pi_1 \leq_p \Pi_2$, s'il existe une fonction calculable en complexité polynomiale $f : I_1 \longrightarrow I_2$ telle que :

$$\forall i \in I_1 : \quad i \in I_1^+ \iff f(i) \in I_2^+$$

Intuitivement : $\Pi_1 \leq_p \Pi_2$ signifie qu'on peut transformer une instance de Π_1 en une instance de Π_2 en complexité polynomiale, en préservant la réponse (oui/non). Un algorithme polynomial pour Π_2 permet alors de résoudre Π_1 en complexité polynomiale.

Exercice

Montrer que $\text{STABLE} \leq_p \text{CLIQUE}$.

STABLE

- Instance : un graphe G et un entier k .
- Question : G contient-il un ensemble stable de taille k , c'est-à-dire un ensemble de k sommets deux à deux non adjacents ?

CLIQUE

- Instance : un graphe G et un entier k .
- Question : G contient-il une clique de taille k ?

Exercice

Montrer que \leq_p est réflexive et transitive.

Théorème

Soient Π_1 et Π_2 deux problèmes de décision tels que $\Pi_1 \leq_p \Pi_2$.

- Si $\Pi_2 \in P$ alors $\Pi_1 \in P$.
- Si $\Pi_2 \in NP$ alors $\Pi_1 \in NP$.

Définition

Soit Π_1 un problème de décision.

- Π_1 est dit NP-difficile si $\forall \Pi_2 \in \text{NP}, \Pi_2 \leq_p \Pi_1$.
- Π_1 est dit NP-complet si Π_1 est NP-difficile et $\Pi_1 \in \text{NP}$.

Intuitivement : un problème est NP-difficile s'il est au moins aussi difficile à résoudre que tous les problèmes de NP.

Théorème

Supposons $\Pi_1 \leq_p \Pi_2$.

Si Π_1 est NP-difficile alors Π_2 est NP-difficile.

Classes de complexité : NP-complétude

Exercice

Soit Π un problème NP-difficile.

Montrer que si $\Pi \in P$ alors $P = NP$.

Exercice

Soit Π un problème NP-difficile.

Montrer que si $\Pi \in P$ alors $P = NP$.

Remarque : Il est conjecturé que $P \neq NP$, donc qu'il est impossible de résoudre un problème NP-difficile en temps polynomial.

Classes de complexité : NP-complétude

Théorème de Cook-Levin (admis)

SAT est NP-complet.

Méthode pour montrer qu'un problème Π est NP-complet :

- 1 Montrer que $\Pi \in \text{NP}$.
- 2 Montrer que $\Pi' \leq_p \Pi$, où Π' est un problème NP-complet connu (par exemple SAT).

Théorème (HP)

3-SAT est NP-complet.

k -SAT

- Instance : une formule logique φ en forme normale conjonctive (FNC) avec au plus k littéraux par clause.
- Question : φ est-elle satisfiable ?

Théorème (HP)

3-SAT est NP-complet.

k -SAT

- Instance : une formule logique φ en forme normale conjonctive (FNC) avec au plus k littéraux par clause.
- Question : φ est-elle satisfiable ?

Remarques :

- On en déduit que k -SAT est NP-complet pour $k \geq 3$, car $3\text{-SAT} \leq_p k\text{-SAT}$.
- Par contre 1-SAT et 2-SAT sont dans P (voir X-ENS 2016)

Exercice

Transformer $\varphi = (x \wedge y) \vee \neg z$ en formule 3-SAT comme dans la preuve précédente (transformation de Tseytin).

Classes de complexité : NP-complétude

Exercice

- ➊ Montrer que $\text{STABLE} \in \text{NP}$.
- ➋ Pour $\varphi = \bigwedge_{k=1}^p C_k$ une instance de 3-SAT, on définit $G_\varphi = (S, A)$ où
 - S contient un sommet par littéral, autant de fois qu'il apparaît dans φ .
 - A contient une arête entre deux sommets s'ils sont dans la même clause ou s'ils sont la négation l'un de l'autre.Dessiner G_φ si $\varphi = (\bar{x} \vee y \vee \bar{z}) \wedge (x \vee \bar{y} \vee z) \wedge (x \vee y \vee z) \wedge (\bar{x} \vee \bar{y})$.
- ➌ Montrer que si G_φ contient une clique de taille p alors φ est satisfiable.
- ➍ Montrer que si φ est satisfiable alors G_φ contient une clique de taille p . Conclure.
- ➎ Montrer que CLIQUE est NP-complet.

Définition (HP)

- Si $\Pi = (I, I^+)$ est un problème de décision, on définit son complémentaire $\bar{\Pi} = (I, I^-)$ où $I^- = I \setminus I^+$.
- On définit co-P et co-NP comme les classes des problèmes dont le complémentaire est dans P et NP respectivement.
- On dit qu'un problème Π est co-NP complet si $\Pi \in \text{co-NP}$ et $\forall \Pi' \in \text{co-NP}, \Pi' \leq_p \Pi$.

Exercice

- 1 Montrer que Π est co-NP complet si et seulement si $\bar{\Pi}$ est NP-complet.
- 2 Montrer que le problème suivant est co-NP complet :

TAUTOLOGIE

- Instance : une formule logique φ .
- Question : φ est-elle une tautologie ?