Calculabilité et Complexité: CM5

Florian Bridoux

Université de Caen

2021-2022

Table des matières

Table des matières

Source

La partie de ce cours sur la complexité a été écrite à l'aide du livre de Sylvain Perifel intitulé COMPLEXITÉ ALGORITHMIQUE (en français).
Plus précisément, le cours ne portera (quasiment) que sur les chapitres 2,3 et 4 de ce livre.

Table des matières

Temps de calcul

Définition: Temps de calcul

Le temps mis par une machine de Turing M sur une entrée x est le nombre d'étapes que met la machine M pour arriver à un état final quand on la lance sur une entrée x.

On notera $M(x)$ l'exécution d'une machine M sur une entrée x.

Remarque

Il serait tentant de mesurer le temps d'exécution en seconde, mais le temps d'exécution dépendrait alors de la puissance de l'ordinateur faisant tourner l'algorithme.

Temps de calcul

Remarque

Pour la plupart des problèmes, plus la taille de l'entrée est grande, plus il faudra du temps pour trouver la solution.

Par exemple, il est évident qu'on mettra plus de temps à trier une liste d'un million d'éléments plutôt qu'une liste de 3 éléments. Il est alors naturel d'évaluer le temps de calcul d'un algorithme en fonction de la taille de son entrée. D'où la définition suivante.

Définition: classe complexité temps déterministe

Pour une fonction $t : \mathbb{N} \to \mathbb{N}$, la classe $\mathbf{DTIME}(t)$ est l'ensemble des langages reconnus par une machine de Turing M telle qu'il existe une constante α pour laquelle, sur toute entrée x, $M(x)$ calcul en temps $\leq \alpha t(|x|)$.

Si T est un ensemble de fonctions, alors $\mathbf{DTIME}(T) = \bigcup_{t \in T} \mathbf{DTIME}(t)$.

Temps de calcul

Remarque

Dans le domaine de la complexité (contrairement à la calculabilité), toutes les machines que nous considérerons s'arrêteront sur toute entrée. C'est par exemple implicite dans la définition précédente puisque $M(x)$ est censée fonctionner en temps $\leq \alpha t(|x|)$. La question n'est plus de savoir si une machine s'arrête, mais en combien de temps elle le fait.

Temps de calcul

Proposition

- Si pour tout n, $f(n) \leq g(n)$ alors $\mathbf{DTIME}(f) \subseteq \mathbf{DTIME}(g)$.
- Si pour tout n, $t(n) \geq n$, $\mathbf{DTIME}(t)$ est clos par union finie, intersection finie et complémentaire.

- Si $L \in \mathbf{DTIME}(f)$ alors il existe une machine reconnaissant L en temps au plus $\alpha f(n) \leq \alpha g(n)$ et donc $L \in \mathbf{DTIME}(g)$.
- Si $L_1, L_2 \in \mathbf{DTIME}(t)$, soit M_1 et M_2 deux machines reconnaissant L_1 et L_2 en temps $\leq \alpha_1 t(n)$ et $\leq \alpha_2 t(n)$ respectivement. Alors une machine pour $L_1 \cup L_2$ ($L_1 \cap L2$) est la machine $M_\cup(x)$ ($M_\cap(x)$) qui exécute $M_1(x)$ puis $M_2(x)$ et accepte ssi l'une des deux (resp. les deux) acceptent. Puisque $M_\cup$ ($M_\cap$) doit réinitialiser le ruban entre les calculs de M_1 et de M_2, elle fonctionne en temps $\leq \alpha_1 t(n) + \alpha_2 t(n) + O(n)$. Puisque $t(n) \geq n$, on a donc $L1 \cup L2 \in \mathbf{DTIME}(t(n))$ ($L1 \cap L2 \in \mathbf{DTIME}(t(n))$).

Temps de calcul

- Une machine pour $\overline{L_1}$ est la machine $\overline{M}$ qui exécute $M_1(x)$ et accepte ssi $M_1(x)$ rejette : elle fonctionne en temps $\leq \alpha_1 t(n)$ donc $\overline{L_1} \in \mathbf{DTIME}(t)$.

Pourquoi définir la classe **DTIME** à une constante α près ? La raison vient du résultat suivant qui montre que le fonctionnement d'une machine peut être accéléré par n'importe quel facteur constant. En d'autres termes, le temps de calcul d'une machine n'est une mesure pertinente qu'à une constante près.

Accélération linéaire

Théorème: Accélération linéaire

Pour toute constante $\epsilon > 0$, si un langage L est reconnu par M en temps $\leq t(n)$, alors il existe une M' reconnaissant L et fonctionnant en temps $\leq (1+\epsilon)n + \epsilon t(n)$.

- L'astuce consiste à agrandir l'alphabet pour pouvoir réaliser en une seule fois plusieurs étapes de calcul de M.
- Pour réaliser c étapes d'un coup, il faut connaître le voisinage de rayon c autour de la cellule en cours : chaque nouvelle cellule contiendra des $(2c+1)$-uples d'anciennes cellules (ce qui revient à passer à l'alphabet de travail Γ^{2c+1}).
- En réalité nous n'allons pas simuler c anciennes étapes en une seule nouvelle, mais en 6 nouvelles : les informations nécessaires sont en effet contenues dans les voisines de gauche et de droite qu'il nous faut donc visiter.

Accélération linéaire

Remarque

Il n'y a pas de magie dans ce résultat : pour accélérer une machine, il suffit d'agrandir l'alphabet pour faire plus de calculs en une étape. C'est grosso-modo ce qu'on fait en remplaçant les micro-processeurs 32 bits par des micro-processeurs 64 bits.

Remarque

La complexité d'un problème est une mesure asymptotique: un temps d'exécution lorsque la taille de l'entrée tend vers l'infini. Ainsi, pour montrer qu'un langage est dans $\mathbf{DTIME}(n^2)$, il suffit de donner un algorithme en temps cn^2 pour n suffisamment grand: il existe alors une constante c' tel qu'il fonctionne en temps $\leq c'n^2$ pour tout n. Et comme la valeur de la constante nous importe peu, nous dirons que l'algorithme fonctionne en temps $O(n^2)$.

Hiérarchie en temps déterministe

Définition

Une fonction $t : \mathbb{N} \to \mathbb{N}$ est constructible en temps s'il existe une constante α et une machine de Turing M qui, sur l'entrée 1^n (l'entier n en unaire) renvoie $1^{t(n)}$ (l'entier $t(n)$ en unaire) en temps $\leq \alpha t(n)$.

En quelque sorte, les fonctions constructibles en temps sont des fonctions $t : \mathbb{N} \to \mathbb{N}$ "raisonnables" qui ont de bonnes propriétés. En réalité, la plupart des fonctions que nous utilisons sont constructibles en temps. En voici quelques exemples.

- $t(n) = c$ pour une constante $c \in \mathbb{N}$;
- $t(n) = n$;
- $t(n) = 2^{n^c}$ pour une constante $c \in \mathbb{N}$;
- de plus, si t_1 et t_2 sont constructibles en temps, alors il en est de même de $t1 + t2$ et $t_1 t_2$: ainsi, tout polynôme $t \in \mathbb{N}[n]$ est constructible en temps

Hiérarchie en temps déterministe

Théorème: Hiérarchie en temps déterministe

Soit $f : \mathbb{N} \to \mathbb{N}$ et $g : \mathbb{N} \to \mathbb{N}$ des fonctions telles que $f(n) \neq 0$ (pour tout $n \in \mathbb{N}$), g est **constructible en temps** et $f \log(f) = o(g)$. Alors $\mathbf{DTIME}(f) \subset \mathbf{DTIME}(g)$.

Soit M_v la machine suivante sur l'entrée $(\langle M \rangle, x)$:

- Simule M sur l'entrée $(\langle M \rangle, x)$ pendant $g(n)$ étapes avec $n = |(\langle M \rangle, x)|$.
- Si M n'a pas terminé son calcul, alors rejeter.
- Sinon, accepter ssi $M(x)$ rejette.

On remarquera que M_v calcule d'abord $g(n)$ pour savoir quand arrêter l'exécution de $M(\langle M \rangle, x)$: puisque g est constructible en temps, cela prend $O(g(n))$ étapes et l'exécution de $M(\langle M \rangle, x)$ prend à nouveau $g(n)$ étapes, donc en tout M_v fonctionne en temps $O(g(n))$: ainsi, $L \in \mathbf{DTIME}(g)$.

Hiérarchie en temps déterministe

Montrons maintenant que $L \notin \mathbf{DTIME}(f)$. Soit M une machine fonctionnant en temps $\alpha f(n)$. Ainsi M_v simule M en temps $\alpha_M \alpha f(n) log(f(n))$ (où α_M est une constante dépendant seulement de M). Pour n assez grand, par hypothèse $g(n) \geq \alpha_M \alpha f(n) log(f(n))$, donc pour x assez grand, la simulation de M par M_v se termine avant $g(n)$ étapes, donc $M_v(\langle M \rangle, x)$ accepte ssi $M(\langle M \rangle, x)$ rejette. Ainsi, M " se trompe " sur l'entrée $(\langle M \rangle, x)$ et ne reconnaît donc pas L.

Table des matières

P et EXP

Définition

La classe P est l'ensemble des langages reconnus en temps polynomial, c'est-à-dire

$$P = \mathbf{DTIME}(n^{O(1)}) = \bigcup_{k \in \mathbb{N}} \mathbf{DTIME}(n^k).$$

La classe EXP (ou $EXPTIME$) est l'ensemble des langages reconnus en temps exponentiel, c'est-à-dire

$$EXP = \mathbf{DTIME}(2^{n^{O(1)}}) = \bigcup_{k \in \mathbb{N}} \mathbf{DTIME}(2^{n^k}).$$

P et EXP

Théorème

$P \subset EXP$.

Preuve:

- On peut se convaincre que $P \subseteq \mathbf{DTIME}(2^n)$: en effet, pour tout langage $L \in P$ $L \in \mathbf{DTIME}(n^c) \subseteq \mathbf{DTIME}(2^n) \subseteq EXP$.
- De plus, par le théorème de hiérarchie en temps déterministe $\mathbf{DTIME}(n^c) \subset \mathbf{DTIME}(n^c c \log(n)) \subset \mathbf{DTIME}(2^n)$.
- Donc $P \subset EXP$.

Exemples de problèmes dans P ou EXP

MULTIPLICATION D'ENTIERS
Entrée: Deux entiers a et b donnée en binaire, un entier k
Question: Le k-ème bit du produit ab vaut 1?

Exemples de problèmes dans P ou EXP

MULTIPLICATION D'ENTIERS
Entrée: Deux entiers a et b donnée en binaire, un entier k
Question: Le k-ème bit du produit ab vaut 1?

Ce problème est dans P : il suffit d'effectuer la multiplication ab grâce à l'algorithme de l'école primaire et de regarder le k-ème bit du résultat. Cela prend un temps polynomial.

Exemples de problèmes dans P ou EXP

ACCESSIBILITÉ
Entrée: Un graphe orienté G et deux sommets s et t
Question: Existe-t-il un chemin dans G de s à t?

Exemples de problèmes dans P ou EXP

ACCESSIBILITÉ
Entrée: Un graphe orienté G et deux sommets s et t
Question: Existe-t-il un chemin dans G de s à t?

Ce problème est dans P : il suffit de faire un parcours du graphe en partant de s et de voir si l'on atteint t avec un parcours en largeur par exemple.

Exemples de problèmes dans P ou EXP

PRIMALITÉ:
Entrée: Un entier a donné en binaire.
Question: a est-il premier?

Exemples de problèmes dans P ou EXP

PRIMALITÉ:
Entrée: Un entier a donné en binaire.
Question: a est-il premier?

Ce problème est dans P mais la démonstration n'est pas évidente. L'algorithme naïf consistant à essayer tous les diviseurs potentiels jusqu'à $\sqrt{a}$. Mais comme la taille de l'entrée est $n = log(a)$, c'est un algorithme exponentiel!.
Il existe bien un algorithme polynomial appelé AKS, mais il n'a été découvert qu'en 2002 par Manindra Agrawal, Neeraj Kayal y Nitin Saxena.
Voir https://fr.wikipedia.org/wiki/Test_de_primalité_AKS si ça vous interesse.

Exemples de problèmes dans P ou EXP

ARRÊT EXP
Entrée: le code d'une machine de Turing M et un entier k en binaire
Question: est-ce que $M(\epsilon)$ s'arrête en $\leq k$ étapes?

Exemples de problèmes dans P ou EXP

ARRÊT EXP
Entrée: le code d'une machine de Turing M et un entier k en binaire
Question: est-ce que $M(\epsilon)$ s'arrête en $\leq k$ étapes?

Ce problème est dans EXP, car on peut simuler $M(\epsilon)$ pendant k étapes : puisque la taille de l'entrée k est $\log(k)$, cela prend un temps exponentiel. Nous n'avons pas encore vu cette notion, mais mentionnons au passage que ce problème est EXP-complet.

P et EXP

Remarque

La classe P est généralement présentée comme celle des problèmes résolubles "efficacement". Bien qu'en pratique on sache résoudre rapidement certains problèmes hors de P, et par ailleurs qu'il soit difficile d'admettre qu'un algorithme fonctionnant en temps $1000000n^{1000}$ soit efficace, il n'en reste pas moins que cette présentation est très souvent pertinente : la plupart des problèmes "naturels" dans P ont un algorithme en $O(n^k)$ pour un petit exposant k et pour la plupart des problèmes "naturels" hors de P, on ne connaît pas d'algorithme efficace.
Finalement, la classe P est " robuste " (bonnes propriétés, notamment de clôture) et donne un bon cadre d'étude en complexité.