Calculabilité et Complexité: CM 7

Florian Bridoux

Polytech Nice Sophia

2024-2025

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 et 3.

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 dont le temps de calcul au pire des cas de M est en $O(t)$.

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 s'arrêter en temps $O(t(|x|))$ au pire des cas. La question n'est plus de savoir si une machine s'arrête, mais en combien de temps elle le fait.

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' telle 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

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 $\mathbf{DTIME}(f) \subset \mathbf{DTIME}(g)$.

Traduction: augmenter le temps d'exécution par au moins "un logarithme" permet de pouvoir décider de nouveaux langages.

Remarque

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

Hiérarchie en temps déterministe: preuve

Preuve:
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, $L(M_v) \in \mathbf{DTIME}(g)$.

Hiérarchie en temps déterministe: preuve (suite)

- Montrons maintenant que $L(M_v) \notin \mathbf{DTIME}(f)$.
- Soit M une machine fonctionnant en temps $\alpha f(n)$ (et montrons que $L(M) \neq L(M_v)$).
- 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 $L(M_v)$.

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 $P \subset EXP$.

On verra une preuve plus concrète dans le TD 6.

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.

Table des matières

Temps non déterministe

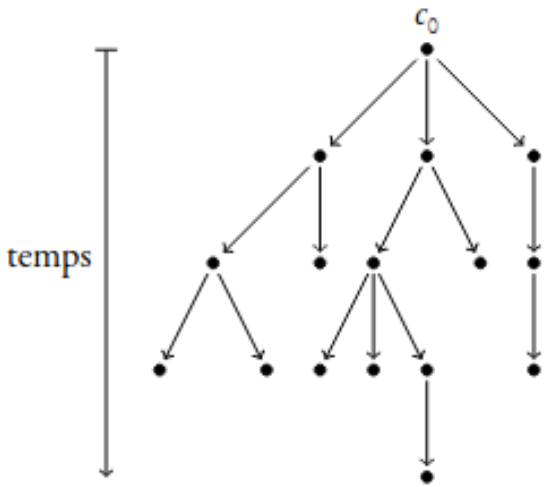
Jusqu'à présent, les machines de Turing que nous avons vues n'avaient aucun choix : la transition qu'elles effectuaient était déterminée uniquement par leur état et les cases lues par la tête. C'est pourquoi on les appelle *machines déterministes*. Si en revanche, à chaque étapes plusieurs transitions sont possibles, la machine a plusieurs exécutions possibles et est appelée *non déterministe*.

Définition: Machine de Turing non déterministe

Une machine de Turing N est dite non déterministe si sa "fonction de transition" δ n'est plus une fonction mais une relation:
$\delta \subseteq (Q \times \Gamma) \times (Q \times \Gamma \times \{\leftarrow, \rightarrow\})$.
À chaque étape, plusieurs transitions sont possibles : tout élément $(q, \Gamma_i), (q', \Gamma_j, d) \in \delta$ signifie que de l'état q, en lisant la lettre Γ_i, la machine N peut aller dans un état q' écrire Γ_j et se déplacer dans la direction d.

Temps non déterministe

Plutôt qu'une suite de configurations, le calcul d'une machine de Turing non déterministe est un arbre de configurations puisqu'une configuration a plusieurs successeurs possibles. Dans cet arbre, certaines configurations peuvent éventuellement apparaître plusieurs fois si elles sont atteintes par différents chemins.



Arbre de calcul d'une machine non déterministe

Temps non déterministe

Définition

Une exécution d'une machine non déterministe N est une suite de configurations compatible avec la relation de transition δ, d'une configuration initiale à une configuration finale.

- Une exécution de N est aussi appelée chemin, car il s'agit d'un chemin dans l'arbre de calcul, de la racine à une feuille.
- Le temps de calcul d'une machine de Turing non déterministe est le temps maximal d'une de ses exécutions, c'est-à-dire la hauteur de son arbre de calcul.
- De même, l'espace utilisé par une machine de Turing non déterministe est l'espace maximal utilisé par l'une de ses exécutions

Temps non déterministe

Définition

Le langage reconnu par une machine non déterministe N sur l'alphabet Σ est l'ensemble des mots $x \in \Sigma^*$ tels qu'il existe un chemin acceptant dans le calcul $N(x)$.

Ainsi, pour accepter un mot x, il faut et il suffit qu'il y ait au moins une exécution qui mène à un état acceptant dans l'arbre de calcul de $N(x)$. En d'autres termes, un mot x est rejeté ssi aucun chemin n'est acceptant dans l'arbre de calcul de $N(x)$.

Remarque

C'est le même principe que pour les AFI.

Temps non déterministe

Définition

Pour une fonction $t : \mathbb{N} \to \mathbb{N}$, la classe **NTIME**$(t)$ est l'ensemble des langages reconnus par une machine de Turing non déterministe N telle qu'il existe une constante α pour laquelle, sur toute entrée x, $N(x)$ fonctionne en temps $O(t(|x|))$ (c'est-à-dire que toute branche de son arbre de calcul sur l'entrée x est de taille majorée par $O(t(|x|))$).

Temps non déterministe

Les classes **NTIME**sont closes par union et intersection.

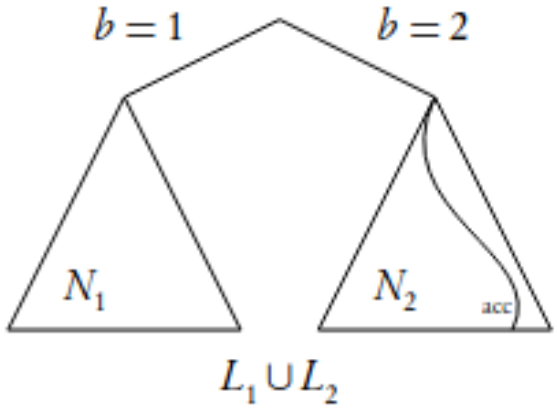
Proposition

Si $L_1, L_2 \in \mathbf{NTIME}(t)$ alors $L_1 \cup L_2 \in \mathbf{NTIME}(t)$ et $L_1 \cap L_2 \in \mathbf{NTIME}(t)$.

Preuve:

- Supposons que N_1, N_2 reconnaissent L_1, L_2 en temps t.

Temps non déterministe

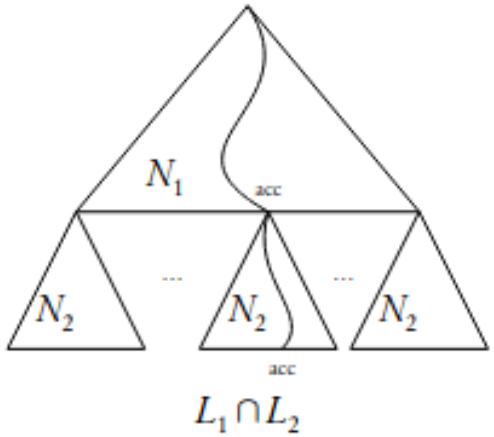


Union pour des machines non déterministes.

Pour reconnaître $L_1 \cup L_2$, on considère la machine $N_\cup$ suivante sur l'entrée x :

- deviner $b \in \{1, 2\}$;
- exécuter $N_b(x)$;
- accepter ssi le chemin simulé de $N_b(x)$ accepte.

Temps non déterministe



Intersection pour des machines non déterministes.

Pour reconnaître $L_1 \cap L_2$, on considère la machine $N_\cap$ suivante sur l'entrée x :

- Exécuter $N_1(x)$;
- Exécuter $N_2(x)$;
- accepter ssi les chemins simulés de $N_1(x)$ et $N_2(x)$ acceptent tous les deux.

Temps non déterministe

Question

Considérons $L \in \mathbf{NTIME}(t)$. Est-ce que son complémentaire $\overline{L}$ est dans $\mathbf{NTIME}(t)$ aussi? Autrement dit, est-ce que $\mathbf{NTIME}(t)$ est clos par complémentation comme $\mathbf{DTIME}(t)$ l'est?

Temps non déterministe

Question

Considérons $L \in \mathbf{NTIME}(t)$. Est-ce que son complémentaire $\overline{L}$ est dans $\mathbf{NTIME}(t)$ aussi? Autrement dit, est-ce que $\mathbf{NTIME}(t)$ est clos par complémentation comme $\mathbf{DTIME}(t)$ l'est?

C'est une question très importante dont la réponse n'a pas encore été prouvée mais la plupart des chercheurs en informatique pensent que non. En tout cas, on se convaincra au tableau qu'il ne suffit pas de prendre l'opposé de la réponse de N qui reconnaît L.

Temps non déterministe

Proposition

Pour une fonction $t : \mathbb{N} \to \mathbb{N}$,

$$\mathbf{DTIME}(t(n)) \subseteq \mathbf{NTIME}(t(n)) \subseteq \mathbf{DTIME}(2^{O(t(n))}).$$

Idée de la démonstration: Pour simuler une machine non déterministe fonctionnant en temps t(n) par une machine déterministe, on parcourt tous les chemins de calcul et on détermine si l'un d'entre eux accepte. Puisqu'il y a un nombre exponentiel de chemins, la machine déterministe prend un temps $2^{O(t(n))}$.