Algorithmique (12X001) - Automne 2023

5. Rappels de complexité algorithmique

Enseignant: Arnaud Casteigts *Assistant: Brian Pulfer*

5.1 Notations asymptotiques

Nous rencontrons souvent les notations O, Ω, Θ en algorithmique (prononcer "grand o", "grand omega" et "grand theta") et parfois aussi les notations o et ω ("petit o", "petit omega"). Mais sait-on vraiment ce qu'elles représentent ?

En fait, ces notations sont très générales et non-exclusivement dédiées à l'informatique. Elles permettent toutes de comparer des quantités asymptotiques en masquant des détails jugés non significatifs, simplifiant au passage d'éventuels calculs.

Prenons l'exemple d'un algorithme dont la complexité en temps serait de $2n^3 + 3n^2 + 4$, où n est la taille de l'entrée à traiter. À mesure que n grandit, la contribution des termes $3n^2$ et 4 deviendra négligeable en comparaison du terme $2n^3$. On peut exprimer cela plus précisément par le fait que $2n^3/(2n^3 + 3n^2 + 4)$ tend vers 1 quand n tend vers ∞, le terme $2n^3$ fournit donc à lui seul une excellente approximation du coût de l'algorithme et il est utile de pouvoir ignorer les deux autres termes, qui sont *dominés* asymptotiquement. C'est la première simplification : ignorer les **termes dominés**.

Supposez maintenant que vous avez le choix entre deux algorithmes, l'un ayant une complexité de $100n^2$ et l'autre de $0.1n^3$. Pour des entrées petites, le second algorithme pourrait être intéressant (jusqu'à $n = 1000$), mais à mesure que n grandit, l'algorithme de complexité $100n^2$ finira forcément par être plus intéressant. En fait, quand $n \to \infty$, le facteur constant (ici 0.1 ou 100) n'a plus aucun impact sur le verdict : c'est la croissance, et seulement elle, qui importe (à moins que les deux aient la même croissance). Il est alors utile de pouvoir ignorer ces facteurs. C'est la seconde simplification : ignorer les **facteurs constants**.

Une fois ces deux simplification effectuées, l'expression $2n^3 + 3n^2 + 4$ devient n^3. Toutes les notations évoquées plus haut ont en commun qu'elles ne s'intéressent qu'à cette forme simplifiée. On a alors les équivalences suivantes :

$a = o(b)$	$a = O(b)$	$a = \Theta(b)$	$a = \Omega(b)$	$a = \omega(b)$
$a < b$	$a \leq b$	$a = b$	$a \geq b$	$a > b$

Notez que le symbole "=" n'a pas la même signification qu'habituellement. En particulier,

il n'est pas symétrique, par exemple $a = O(b)$ n'implique pas que $b = O(a)$! Pour éviter les confusions, on utilise parfois le symbole $\in$ à la place (avec la même signification).

Voici le tableau discuté en classe (sans les réponses) :

Égalité	Vrai	Faux
$4n^2 - 5n + 1 = O(n^2)$		
$4n^2 - 5n + 1 = \Theta(n^2)$		
$n \log(n) = \Omega(n)$		
$n \log(n) = O(n^2)$		
$n \log(n) = \Theta(\_\_\_\_\_)$?	N/A	N/A
$n \log(n^2) = \Theta(\_\_\_\_\_)$?	N/A	N/A
$500 = \Theta(1)$		
$17n^2 + 3 = n^{O(1)}$		
$\sqrt{n} = O(n)$		
$n! = \Omega(2^n)$		

5.2 Quelques adjectifs

Constant	$O(1)$
Logarithmique	$O(\log n)$
Linéaire	$O(n)$
Quasi-linéaire	$O(n \log n)$
Quadratique	$O(n^2)$
Polynomial	$O(n^c) = n^{O(1)}$
Quasi-polynomial	$O(n^{log^{O(1)} n})$
Exponentiel	$O(2^n)$ ou parfois $O(2^{n^{O(1)}})$
Factoriel	$O(n^n)$

Ces adjectifs peuvent s'appliquer aux algorithmes aussi bien qu'aux problèmes qu'ils résolvent. Les mêmes adjectifs sont utilisés vis à vis des notations Θ et Ω. Pour o et ω, on parlera plutôt de super-X et de sous-X (où X est l'adjectif de votre choix), par exemple, $\omega(n)$ est super-linéaire et $o(n^2)$ est sous-quadratique. Notez que les bases des logarithmes disparaissent également (c'est bien pratique), car les logs en bases différentes ne diffèrent que par des facteurs constants, il n'est donc pas utile de préciser s'il s'agit du logarithme naturel ou du logarithme en base deux, par exemple.

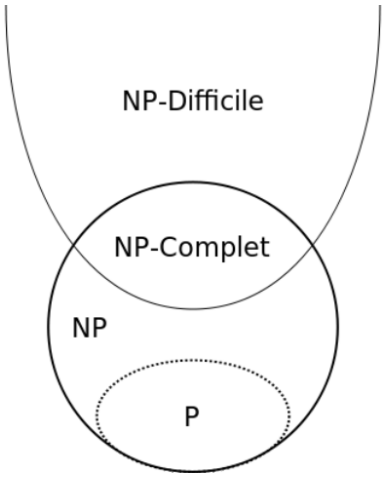
5.3 P, NP, NP-difficile, NP-complet

Ces noms désignent tous des familles de problèmes (aussi appelées classes de problèmes). Le sujet est vaste ! Nous rappelons ici les points clés seulement. Ces classes concernent les problèmes *de décisions* (problèmes dont la réponse est OUI ou NON), à l'exception de NP-difficile, qui peut être n'importe quel type de problème.

- P : Problèmes qui peuvent être résolus en temps polynomial
- NP : Problèmes qui admettent des certificats positifs vérifiables en temps polynomial. Cela signifie que si la réponse est OUI, alors il existe une preuve de cela, qui peut être vérifiée rapidement (par exemple, pour la 3-coloration, la coloration elle-même).
- NP-difficile : Problèmes qui sont au moins aussi difficile que n'importe quel problème dans NP. Un problème A est "au moins aussi difficile" qu'un problème B si toute instance de B peut se réduire à une instance de A (cette réduction devant être elle-même polynomiale, sinon c'est de la triche).
- NP-complet : à la fois dans NP et dans NP-difficile. Il existe en effet des problèmes dans NP qui sont au moins aussi difficiles que tous les autres. En fait, il y en a plein !

La classe NP peut aussi être définie comme les problèmes de décisions (vus ici comme des langages formels où chaque mot du langage représente une instance dont la réponse est OUI) reconnaissables en temps polynomial par une machine de turing non-déterministe (P étant la même chose, mais déterministe). Il se trouve que c'est équivalent à la définition donnée plus haut, qui est plus facile à appréhender. Nous pouvons donc mettre de côté les machines de Turing, temporairement.

L'une des plus grandes questions en informatique fondamentale est de savoir si P $\neq$ NP. La majorité des chercheurs du domaine pensent que c'est le cas, ce qui donne le schéma :



Il semble donc peu probable qu'un problème NP-complet soit résoluble en temps polynomial (mais qui sait ?).