Information, Calcul et Communication: Cours 5

Résumé des quatre premières leçons:

- Algorithmes: principes de base et complexité temporelle
- Récursivité et programmation dynamique

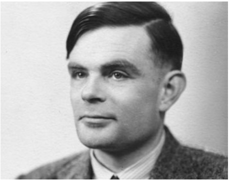
Plan de cette leçon: Introduction à la théorie du calcul

- Tout problème est-il soluble par un algorithme?
- Classification des problèmes par ordre de complexité

Introduction à la théorie du calcul

Première question: Tout problème est-il soluble par un algorithme?

Réponse: Non! (Alan Turing, 1936)



Pour bien comprendre cette réponse, on doit d'abord définir ce qu'on entend par “problème”.

Un exemple de problème qui n'en est pas vraiment un!

Quel est le nombre d'étudiants inscrits dans chaque section en première année à l'EPFL pour l'année académique 2019-2020?

Algorithme de résolution: compter d'une manière ou d'une autre les étudiants inscrits dans chaque section $\longrightarrow$ GC: 120, MX: 70, etc.

Mais le nombre de sections à l'EPFL est un nombre fini! On peut donc établir une fois pour toutes une table de correspondance:

section:	GC	MX	...
inscrits:	120	70	...

Après ça, plus besoin d'algorithme pour résoudre ce problème!

Un autre exemple de problème

Etant donné un nombre entier positif N, celui-ci est-il un nombre premier? (i.e., un nombre admettant exactement deux diviseurs distincts: 1 et le nombre lui-même)

Ce problème a un nombre infini d'instances. On ne peut donc pas établir "une fois pour toutes" une table de correspondance.

Pour autant, existe-t-il un algorithme qui permette de le résoudre? Oui: il suffit de tester qu'aucun nombre entre 2 et $N - 1$ n'est un diviseur de N (pas forcément la méthode la plus efficace...).

= problème de décision, qui ne demande qu'une réponse "oui" ou "non"

Turing (1936):
Certains problèmes sont indécidables, notamment le problème de l'arrêt.

Le problème de l'arrêt

Tout d'abord, voici une question "simple" :

"Etant donné un algorithme P et des données d'entrée X, l'algorithme P(X) s'exécute-t-il en un temps fini ?"

Nous avons déjà vu un exemple d'algorithme (celui générant la suite de Syracuse) pour lequel nous ne sommes pas sûrs de la réponse à donner à la question ci-dessus pour toute donnée d'entrée X.

Mais ceci est peut-être simplement dû à notre incapacité à analyser précisément un tel algorithme...

Ce qu'Alan Turing démontre en 1936, c'est qu'il est impossible d'écrire un algorithme A prenant en entrée un autre algorithme P (concrètement, une suite de 0 et de 1), ainsi que des données d'entrée X (i.e., une autre suite de 0 et de 1) et dont la sortie soit la réponse à la question "simple" ci-dessus : "Est-ce que P(X) s'arrête ?"
Le problème de l'arrêt est donc un problème indécidable.

Note : Evidemment que pour beaucoup de P et de X, il est possible d'obtenir une réponse à la question ci-dessus.
Ce que Turing montre, c'est que c'est impossible pour tous P et X.

Démonstration (par l'absurde)

Supposons qu'un tel algorithme A existe, i.e.

$$\begin{cases} A(P, x) \text{ sort "oui" si } P(x) \text{ s'arrête} \\ A(P, x) \text{ sort "non" si } P(x) \text{ continue indéfiniment} \end{cases}$$

A partir de cet algorithme A, on construit un autre algorithme B :

entrée : algorithme P

sortie : aucune

si $A(P, P) = $ oui, effectuer une boucle infinie

si $A(P, P) = $ non, s'arrêter

Que se passe-t-il si on exécute l'algorithme B avec lui-même en entrée ? i.e., que fait B(B) ?

- si A(B,B) = oui (i.e., si B(B) s'arrête), alors B(B) exécute une boucle infinie ???
- si A(B,B) = non (i.e., si B(B) continue indéfiniment), alors B(B) s'arrête ???

Dans les deux cas, on a clairement une contradiction.

Conclusion: L'hypothèse effectuée (l'algorithme A existe) est fausse. #

Classes de complexité des problèmes

Deuxième question: Les problèmes solubles le sont-ils tous en un temps raisonnable?

Réponse: Encore non!

Pour essayer de clarifier quels problèmes sont faciles à résoudre et quels problèmes le sont moins, on introduit des classes de complexité.

Deux classes de complexité importantes

Définition

La classe P est l'ensemble des problèmes qui peuvent être résolus en temps polynomial, i.e., pour lesquels il existe un algorithme de résolution dont la complexité temporelle est $\mathcal{O}(n^p)$ pour des données d'entrée de taille n (avec $p \geq 1$ un nombre fixé).

Définition

La classe NP est l'ensemble des problèmes pour lesquels, si "on" nous propose une solution du problème, il est alors possible de vérifier en temps polynomial si celle-ci en est une ou pas.

Remarques:

- NP ne veut pas dire "non-polynomial".
- P $\subset$ NP (il est plus facile de vérifier une solution que d'un proposer une!)

Exemples de problèmes appartenant à la classe P

Beaucoup de problèmes que nous avons déjà rencontrés dans ce cours appartiennent à classe P (et sont donc des problèmes "faciles" à résoudre) :

- Le problème de la recherche d'un élément dans une liste de taille n.
- Le problème du tri d'une liste de taille n.
- Le calcul de la somme/moyenne/produit de n nombres.
- Identifier si tous les éléments d'une liste sont différents (de taille n)

Un nouveau problème :

"Etant donné une liste L de n nombres entiers, existe-t-il $i \neq j \in \{1, \dots, n\}$ tels que $L(i) + L(j) = 0$?"

<u>Exemple</u> : $L = (+14, -6, +1, +3, -4, +8, -20) \Longrightarrow$ réponse : <u>non</u>

$O(n^2)$ opérations sont suffisantes pour répondre à cette question : le problème est donc dans P [<u>Exercice</u> : en fait, $O(n \log_2 n)$ opérations suffisent aussi ; voyez-vous lesquelles ?].

Une variation intéressante :

"Etant donné une liste L de n nombres entiers, existe-t-il un sous-ensemble $S \subset \{1, \dots, n\}$ tel que $\sum_{i \in S} L(i) = 0$?"

Ce problème s'appelle le <u>problème des sommes de sous-ensembles</u>.

Pour pouvoir répondre à cette question, il faut a priori essayer <u>tous</u> les sous-ensembles $S \subset \{1..n\}$. Dans l'exemple ci-dessus, il se trouve qu'en choisissant dans $L = (\underline{+14}, \underline{-6}, \underline{+1}, \underline{+3}, -4, \underline{+8}, \underline{-20})$, on obtient $+14-6+1+3+8-20 = +26-26 = 0$, donc la réponse est <u>oui</u>. Mais essayer tous les sous-ensembles $S \subset \{1,..,n\}$ est chronophage ; l'ensemble $\{1,..,n\}$ possède en effet 2^n sous-ensembles différents. Cet algorithme de résolution n'a donc pas une complexité temporelle polynomiale en n.

- Pour autant, on ne sait pas à l'heure actuelle s'il existerait ou non un algorithme avec complexité polynomiale en n capable de résoudre ce problème. On ne sait donc pas si ce problème appartient à la classe P.
- Par contre, on sait que ce problème appartient à la classe NP ; en effet, si on nous propose une solution du problème (comme le sous-ensemble S proposé en rouge ci-dessus), il est possible de vérifier en temps polynomial que c'en est une en additionnant simplement les nombres proposés.

Deux autres exemples de problèmes

1. Primalité

"Etant donné un nombre entier positif N à n chiffres, ce nombre est-il premier ?"

Il n'a été démontré que récemment (en 2002) que ce problème appartient à la classe P (et donc aussi à NP), car admettant un algorithme de résolution de complexité temporelle $O(n^{12})$. (!)

Notes:
- L'algorithme évoqué au début de cette leçon a une complexité temporelle exponentielle en n.
- En pratique, on utilise encore un autre algorithme pour tester si un nombre est premier.

2. Factorisation

Soient P, Q deux nombres premiers à n chiffres, et soit $N = P \cdot Q$.

" Etant donné N, on aimerait retrouver P et Q. "

Exemple : si $N = 98'201$, que valent P et Q ?

Ce problème n'est a priori pas facile à résoudre...

Par contre, si on nous propose une solution (ici, $P = 347$ et $Q = 283$), il est facile de vérifier que $N = P \cdot Q$ en effectuant la multiplication : le problème appartient donc à la classe NP.

(mais on ne sait pas s'il appartient à la classe P)

Problèmes NP-complets et une question à un million de $

Parmi tous les problèmes connus appartenant à la classe NP, plusieurs ont été identifiés comme étant les plus difficiles à résoudre dans cette classe: on les appelle les problèmes NP-complets: c'est le cas notamment du problème des sommes de sous-ensembles (mais ça n'est étonnamment pas le cas de celui de la factorisation des nombres entiers).

"Définition"

Si P_1 est un problème NP-complet, alors tout autre problème P_2 appartenant à la classe NP peut être réduit au problème P_1, i.e., tout algorithme résolvant P_1 peut être utilisé pour résoudre P_2.

A l'heure actuelle, on ne sait pas s'il existe un algorithme de complexité polynomiale capable de résoudre un problème NP-complet. Si on trouvait un tel algorithme, on démontrerait du même coup que les classes P et NP sont identiques. Mais on ne sait pas non plus démontrer qu'un tel algorithme n'existe pas! Le *Clay Mathematics Institute* a proposé en 2000 une récompense d'un million de dollars à qui apportera un réponse à cette question. L'argent dort toujours au frais dans un compte en banque...

Classes de complexité des problèmes : Résumé