Bornes inférieures 2 : adversaire

Notes de cours[1] de complexité des problèmes, M1

1 Présentation

L'argument de comptage des solutions vu dans le cours précédent est un argument puissant ... quand l'algorithme a suffisamment de réponses possibles. Ce n'est pas toujours le cas. Notamment, beaucoup d'algorithmes sont des *algorithmes de décision*, qui répondent par "oui" ou par "non" : est-ce que le graphe est connexe ? est-ce que le nombre x est dans le tableau T ? ... Avec 2 solutions différentes, la borne de comptage est $\Omega(1)$, ce qui n'apporte aucune information.

Il faut donc trouver d'autres méthodes pour établir des bornes inférieures, l'*argument de l'adversaire*, qui fait l'objet de ce cours, en est une.

Considérons le jeu suivant : le professeur choisit mentalement une couleur parmi {rouge, vert, bleu, jaune} et l'étudiant doit la deviner en un nombre minimal de questions du type "est-ce que c'est la couleur x ?". L'étudiant a trouvé quand le professeur répond "oui". Seul le professeur connaît la couleur choisie, *il peut donc facilement tricher* et plus important *sans qu'on puisse le prouver* :

- Il peut s'arranger pour qu'un étudiant trouve du premier coup, en répondant "oui" à la première question.
- Il peut s'arranger pour qu'un étudiant n'ait toujours pas trouvé au troisième coup, en répondant "non" aux trois premières questions.

En revanche il ne peut pas, sans se faire repérer, s'arranger pour répondre non quatre fois de suite : si l'étudiant a bien proposé les quatres couleurs différentes, il aura la preuve que le professeur triche.

Mais dans tous les cas, pour l'algorithme (l'étudiant), il y a toujours une entrée (un choix du professeur) qui demande au moins 3 "non" avant de trouver la bonne réponse : il faut au moins 4 tentatives. Ce résultat est établi en considérant que le professeur cherche à géner au maximum l'étudiant, s'où le nom d' "adversaire".

Pour formaliser un peu l'argument au-dessus, on procède de la façon suivante. Au début, le professeur ne choisit pas une couleur, il prétend juste l'avoir fait. Il garde en mémoire un ensemble C de couleurs. Au début, C ={rouge, vert, bleu, jaune}. Chaque fois qu'il donne une réponse négative à la question "est-ce que c'est la couleur x ?", il enlève x de C (si x est dans C). Donc à tout moment, *tous les éléments de C* sont des couleurs cohérentes avec les réponses qu'il a déjà effectuées. On vient de décrire un algorithme pour l'adversaire.

Maintenant on remarque qu'à chaque réponse négative, on enlève au plus un élément de C. Donc après deux réponses négatives, il reste au moins deux éléments dans C et l'étudiant ne peut pas savoir laquelle c'est.

2 Recherche d'un élément

On peut directement généraliser ce qu'on vient de voir et obtenir :

[1] Ces notes sont (fortement) inspirées des notes de cours de Jeff Erickson : `http://web.engr.illinois.edu/~jeffe/teaching/algorithms/notes/29-adversary.pdf`

Théorème 1 *Soit T un tableau de n nombres (trié ou non). Tout algorithme qui cherche à savoir à quel indice se trouve x dans T (s'il y est) en ne posant que des requêtes de la forme "est-ce que $T[i] = x$?", nécessite au moins n questions dans le pire des cas.*

Le modèle de calcul du théorème 2 n'est pas très puissant, on ne peut par exemple pas faire de dichotomie dans ce modèle si l'entrée est un tableau trié (ce qui explique la borne inférieure en $\Omega(n)$).

Enrichissons le modèle. On autorise maintenant les questions du type "comment se comparent x et $T[i]$?", avec trois réponses possibles $<$, $=$ ou $>$. Le nombre de réponses à chaque noeud reste borné (trois en l'occurrence). On veut établir le même genre de résultat que pour le théorème 2, pour des tableaux sans hypothèse particulière (si on suppose que T est trié ce n'est pas possible d'avoir une borne inférieure en $\Omega(n)$ pour ce modèle). On va montrer qu'il faut encore $n-1$ choix dans le pire des cas grâce au même argument d'adversaire : l'adversaire commence avec un tableau rempli de n fois le symbole ?, qui signifie "pas encore fixé". A chaque fois que l'algorithme pose une question qui porte sur la case i, il met $x-1$ dans la case $T[i]$ et répond "x est plus grand que $T[i]$". Au bout de $n-1$ questions, on a enlevé au plus $n-1$ symboles ?, il reste donc au moins une case avec un ?. Comme l'adversaire peut mettre n'importe quelle valeur dans cette case sans se contredire, l'algorithme n'a aucun moyen de répondre correctement sans comparer x et $T[i]$:

Théorème 2 *Soit T un tableau de n nombres. Tout algorithme qui cherche à savoir à quel indice se trouve x dans T (s'il y est) en ne posant que des requêtes de la forme "comment se comparent $T[i]$ et x ?", nécessite au moins n questions dans le pire des cas.*

Est-ce que la borne inférieure est toujours valide dans le modèle plus général de l'arbre de décision où on peut poser n'importe quelle question oui/non ? la réponse est **non**. Dans un tel modèle on peut simuler une dichotomie en posant des questions du type "est-ce que x est présent dans au moins une des cases d'indice $< \frac{n}{2}$?" et obtenir un algorithme en $\mathcal{O}(\log n)$. Bien sur, ces questions sont trop compliquées pour qu'on puisse raisonnablement les considérer comme traitables en temps constant : le modèle de calcul est trop puissant ce qui, dans ce cas, détériore les bornes inférieures.

3 Formalisation

Voilà comment formaliser la méthode de l'adversaire. On note $\mathcal{E}_n$ l'ensemble des entrées de taille n et par $\mathcal{S}_n$ l'ensemble de ses sorties sur les éléments de $\mathcal{E}_n$. On note α la fonction de $\mathcal{E}_n$ dans $\mathcal{S}_n$ réalisée par l'algorithme : à une entrée $e \in \mathcal{E}_n$ elle associe le résultat $\alpha(e) \in \mathcal{S}_n$ du calcul effectué par l'algorithme.

L'algorithme est vu dans un modèle de décision comme une suite de questions Q_1, Q_2, ... Bien entendu, le choix de la question Q_i dépend des réponses aux questions précédentes. Si

$$s = (Q_1, R_1) \cdots (Q_k, R_k)$$

est une suite de questions/réponses, on note $\mathcal{E}_n(s)$ l'ensemble des entrées de $\mathcal{E}_n$ pour lesquelles toutes les réponses sont correctes. On dit que $\mathcal{E}_n(s)$ est *inachevé* s'il contient deux éléments e et f tels que $\alpha(e) \neq \alpha(f)$: dans ce cas il y a des entrées avec des résultats différents pour lesquelles les réponses sont correctes, donc l'algorithme ne peut pas encore donne le résultat.

On dit que le **problème** admet une *stratégie d'adversaire de longueur k* (c'est en fait une fonction de n, $k := k(n)$) quand quelque soit la suite de question $Q_1 \cdots Q_k$, il existe une suite de réponses $R_1 \cdots R_k$ telles que $E_n(s)$ soit inachevé. Le "quelque soit" signifie donc ici "pour tous les algorithmes". On en déduit directement que :

Théorème 3 *Si dans un modèle de décision fixé, un problème admet une stratégie d'adversaire de longueur k, alors on a une borne inférieure en $\Omega(k)$.*

Le plus souvent, la stratégie est elle-même donnée par un algorithme, qui explique quelle réponse R_i donner à chaque étape.

4 Trouver le min et le max

On s'intéresse maintenant qu problème suivant : étant donné un tableau de nombres de taille n, quels sont les indices du minimum et du maximum du tableau ? On se place dans le modèle de décision par comparaisons (on peut comparer $T[i]$ et $T[j]$).

Par un argument d'adversaire, on montre facilement une borne inférieure (et un algorithme) en $n-1$ comparaisons pour trouver le minimum. C'est donc aussi une borne inférieure pour trouver le min et le max. Si on cherche d'abord le min, puis le max parmi les indices restants, on obtient un algorithme qui utilise $2n-3$ comparaisons.

Pour simplifier, on suppose n pair et on va montrer une borne inférieure en $\frac{3}{2}n-2$, et un algorithme optimal qui effectue $\frac{3}{2}n-2$ comparaisons dans le pire des cas.

L'algorithme est facile à obtenir : on prend compare les éléments deux à deux, $T[2i]$ avec $T[2i+1]$, et à chaque fois on met le plus petit dans un tableau P et le plus grand dans un tableau G. On cherche ensuite le min de P et le max de G, cela coûte au total un nombre de comparaisons égal à

$$\underbrace{\frac{n}{2}}_{\text{calcul de P et G}} + \underbrace{\frac{n}{2}-1}_{\min(P)} + \underbrace{\frac{n}{2}-1}_{\max(G)} = \frac{3n}{2}-2.$$

L'argument d'adversaire est plus subtil. On marque chaque case du tableau avec un + et avec un -. Le + signifie "ce sera peut-être le max" et le - signifie "ce sera peut-être le min". Au début, comme toujours pour ce genre d'arguments, on se laisse toutes les possibilités. Quand l'algorithme fait une comparaison $T[i] \leq T[j]$, l'adversaire procède de la façon suivante (on note $i = \pm$ pour dire que i a ses deux marques, $i = +$ ou $i = -$ s'il n'en a plus qu'une et $i = \emptyset$ s'il n'a plus de marque). On note dans le tableau les modifications de marquage à effectuer, ainsi que la réponse à la question :

	$j = \pm$	$j = +$	$j = -$	$j = \emptyset$
$i = \pm$	$i = -, j = +, vrai$	$i = -, vrai$	$i = +, faux$	$i = -, vrai$
$i = +$	$j = -, faux$	$i = \emptyset, vrai$	$faux$	$faux$
$i = -$	$j = +, vrai$	$vrai$	$j = \emptyset, vrai$	$vrai$
$i = \emptyset$	$j = +, vrai$	$vrai$	$faux$	$T[i] < T[j]$

Pour que cela fonctionne il faut que l'adversaire fixe une valeur dans le tableau quand il enlève la dernière marque d'une case. Pour cela on a un compteur c, initialisé à 0. Si la dernière marque enlevée est un +, on met c dans la case correspondante, si c'est un -, on met $-c$. Dans les deux cas on incrémente le compteur après. Du coup on peut répondre pour la dernière case du tableau. Cette façon de faire garantit que toutes les réponses sont cohérentes entres elle et que tous les indices marqués d'un + peuvent accueillir le maximum et tous ces marqués d'un - peuvent accueillir le minimum[2].

Au début il y a $2n$ marques. On remarque maintenant que chaque comparaison enlève deux marques si on compare deux indices de type $\pm$ et au plus une sinon. On ne peut enlever

[2]La preuve se fait par récurrence, en mettant dans l'hypothèse qu'on peut à tout moment mettre dans une case marquée + (resp. -) une valeur plus grande (resp. plus petite) que toutes celles utilisées jusqu'ici sans contredire les réponses précédentes.

qu'au plus $\frac{n}{2}$ fois deux marques et il en reste ensuite n. Les autres questions enlèvent au plus une marque. Tant qu'il reste trois marques ou plus l'algorithme ne peut pas conclure (il y a forcément deux positions possibles pour le maximum, marquées d'un +, ou pour le minimum, marquée d'un moins). Donc le mieux que peut faire l'algorithme c'est d'enlever $\frac{n}{2}$ fois 2 marques puis $n-2$ fois une marque, pour un nombre de comparaisons égal à $\frac{3n}{2}-2$.

Cela vaut bien un théorème :

Théorème 4 *Il faut $c_n = \lceil \frac{3n}{2} \rceil - 2$ comparaisons pour trouver à la fois le minimum et le maximum d'un tableau de taille n :*

- *il existe un algorithme qui utilise c_n comparaisons dans le pire des cas pour résoudre le problème;*
- *dans le modèle d'arbre de décisions par comparaisons, aucun algorithme ne peut résoudre le problème en moins de c_n comparaisons.*

5 Connexité d'un graphe non-orienté

On considère maintenant comme entrée des graphes non-orientés avec n sommets $\{1, \ldots, n\}$. Il y a $\binom{n}{2}$ arêtes possibles pour un tel graphe : une pour chaque paire $\{i, j\}$ avec $i \neq j$.

Dans le modèle de décision que l'on considère, la seule question que l'on peut poser est "est-ce qu'il y a une arête entre i et j ?". On s'intéresse au problème de savoir si un graphe est connexe, c'est-à-dire si tout sommet est relié à tout autre sommet.

On peut tester la connexité avec un parcours en profondeur depuis le sommet 1, ce qui, dans le pire des cas, demande à tester toutes les arêtes. Est-ce qu'on peut faire mieux ?

La réponse est **non** :

Théorème 5 *Si la seule question que l'on peut poser est "est-ce qu'il y a une arête entre i et j ?", il faut $\binom{n}{2}$ questions dans le pire des cas pour décider si le graphe est connexe ou non.*

L'adversaire procède de la façon suivante. Il garde en mémoire deux graphes A et P d'ensemble de sommets n. Dans A il met toutes les arêtes qui sont forcément dans le graphe (à cause de ses réponses précédentes). Dans P il met toutes les arêtes qui sont peut-être dans le graphe (celles pour lesquelles il n'a pas répondu "non" pour le moment). Notons que P contient, entre autre, toutes les arêtes de A. Au début, il n'y a encore eu aucun oui et aucun non donc A ne contient aucune arête et P contient toutes les arêtes.

L'algorithme de l'adversaire est le suivant pour répondre à une question sur l'arête $i-j$ qui n'a pas déjà été demandée : si enlever $i-j$ de P ne le déconnecte pas, alors on l'enlève et on retourne faux ($i-j$ n'est pas dans le graphe); sinon on l'ajoute dans A et on retourne vrai.

A tout moment A est un sous-graphe de P et P est connecté. Plus intéressant, si P possède un cycle, alors aucune arête de ce cycle n'est dans A : par l'absurde, s'il y a des arêtes du cycle dans A, la première qu'on a mise ne déconnectait pas P, puisqu'on peut utiliser le reste du cycle pour en relier les extrémités. On en déduit que A est acyclique. Par conséquent, si $A \neq P$, alors A n'est pas connecté : s'il l'était, ce serait un arbre; comme P contient au moins une arête de plus e, cette arête forme un cycle dans $A \cup \{e\}$ et donc dans P, mais ce n'est pas possible car les autres arêtes du cycle sont dans A.

A aucun moment l'algorithme ne peut savoir si le graphe d'entrée est A ou P, les deux sont consistants avec les réponses que l'adversaire a donné. Si on a posé strictement moins de $\binom{n}{2}$ questions, alors $A \neq P$ (il y a au moins une arête dont le statut n'est pas décidé et qui est donc dans P mais pas dans A). Mais dans ce cas A n'est pas connexe alors que P l'est et l'algorithme ne peut pas répondre.