Bornes inférieures

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

1 Présentation

On rappelle la notation Ω : si $u_n \in \Omega(v_n)$ cela veut dire que pour une certaine constante $C > 0$ et pour n assez grand on a

$$u_n \geq C \cdot v_n.$$

C'est donc dans l'autre sens que le $\mathcal{O}$.

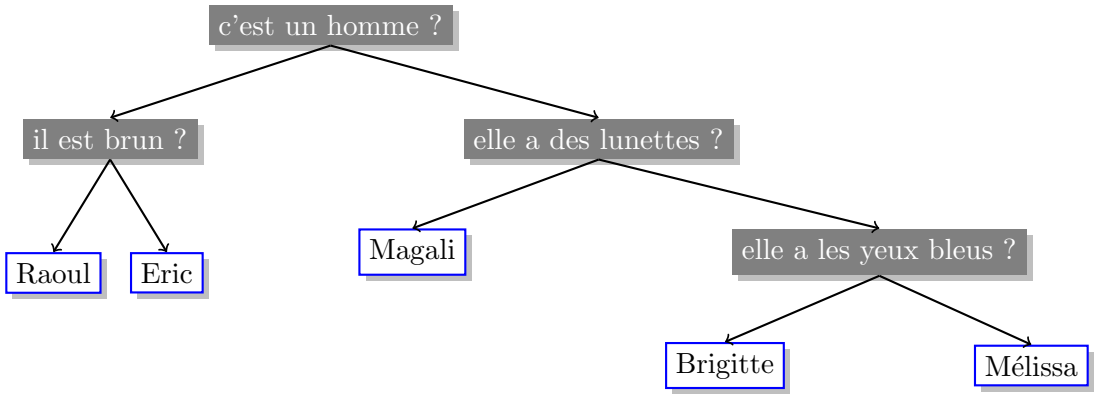
L'objectif est d'établir des *bornes inférieures* à certains problème. Dire qu'un problème P a une borne inférieure en $\Omega(n^2)$ signifie que *tout* algorithme qui permet de résoudre P a une complexité en $\Omega(n^2)$. Cela signifie aussi que si on trouve un algorithme en $\mathcal{O}(n^2)$ pour résoudre P, on a un algorithme *optimal*.

On ne peut pas parler de borne inférieure sans préciser le *modèle* utilisé pour l'ordinateur. Celui-ci peut varier, et *différents modèles* conduisent parfois à *différentes bornes inférieures* : le choix du modèle est vraiment crucial.

2 Modèle de l'arbre de décision

Un modèle souvent utilisé est le *modèle de l'arbre de décision*. Dans ce modèle on accède aux données de l'algorithme uniquement par des questions, disons où on répond par oui ou par non. On le représente par un arbre : l'algorithme pose des questions dans les noeuds et prend la branche gauche si la réponse est "oui" et la branche droite si la réponse est "non". L'arbre modélise *toutes* les exécutions de l'algorithme : une exécution étant un chemin de la racine à une feuille. Quand on arrive à une feuille, l'algorithme doit avoir recolté suffisamment d'information pour produire le résultat voulu : les résultats sont sur les feuilles.

L'arbre ressemble au jeu pour enfants "Qui est-ce ?"[2] où il faut identifier un personnage caché en posant des questions oui/non.



On peut généraliser à plus de réponses possibles que oui/non, mais pour que ce soit un modèle d'arbre de décision, il faut que le nombre de réponses soit *borné*.

La complexité dans le pire des cas d'un algorithme donné par son arbre de décision est la *hauteur de l'arbre* : c'est bien l'exécution qui demande le plus de questions. Sur l'exemple, la hauteur de l'arbre est 3 et Brigitte et Mélissa sont les pires cas de l'algorithme.

[1] Ces notes sont (fortement) inspirées des notes de cours de Jeff Erickson : `http://web.engr.illinois.edu/ jeffe/teaching/algorithms/notes/28-lowerbounds.pdf`

[2] http://fr.wikipedia.org/wiki/Qui_est-ce_%3F

3 Hauteur minimum d'un arbre

On vient d'exprimer la complexité d'un algorithme dans le modèle d'arbre de décision comme la hauteur de son arbre. On a donc besoin sur la plus petite hauteur d'un arbre binaire (ou k-aire si les questions ont k réponses).

Plus précisément on s'intéresse à la question suivante : **quel est la plus petite hauteur possible pour un arbre binaire complet avec f feuilles ?** On note cette quantité $H(f)$. On remarque d'abord que la fonction $f \mapsto H(f)$ est croissante. Ensuite, si on a un arbre binaire a f feuilles qui minimise la hauteur, et qu'il est de la forme $\overset{\circ}{\underset{\mathcal{A}\ \mathcal{B}}{\wedge}}$ alors on peut choisir $\mathcal{A}$ et $\mathcal{B}$ qui minimisent leur hauteur par rapport à leur nombre de feuilles. Si $f_{\mathcal{A}}$ et $f_{\mathcal{B}}$ sont respectivement le nombre de feuilles de $\mathcal{A}$ et $\mathcal{B}$, on a $f_{\mathcal{A}} + f_{\mathcal{B}} = f$ et par symétrie on peut supposer que $f_{\mathcal{A}} \geq f_{\mathcal{B}}$ dans ce cas, comme H est croissante, la hauteur de l'arbre est $1 + H(f_{\mathcal{A}})$. Et la configuration qui minimise cette quantité est quand $f_{\mathcal{A}}$ est la plus petite possible, ce qui, avec la contrainte $f_{\mathcal{A}} \geq f_{\mathcal{B}}$, implique que $f_{\mathcal{A}} = \lceil f/2 \rceil$. On a donc

$$H(f) = 1 + H(\lceil f/2 \rceil),$$

donc en utilisant par exemple le théorème maître, on a que $H(f) = \Theta(\log f)$. Le raisonnement fonctionne pour $k \geq 2$ au lieu de 2 comme facteur de branchement.

4 Borne inférieure pour le modèle d'arbre de décision

Le résultat est le suivant :

Théorème 1 *Soit P un problème qui admet f_n résultats différents pour les entrées de taille n. Dans le modèle d'arbre de décision, le nombre de questions posées par n'importe quel algorithme qui résoud P est en $\Omega(\log f_n)$, pour le pire cas.*

Démonstration : L'arbre de n'importe quel algorithme qui résoud P possède au moins f_n feuilles car il doit pouvoir isoler chaque solution (il peut en revanche y avoir plusieurs feuilles étiquetées par la même solution).

On a vu dans la partie précédente qu'un arbre d'arité au plus k avec f_n feuilles est de hauteur en $\Omega(\log f_n)$.

On en conclut que pour le pire cas, n'importe quel algorithme utilise $\Omega(\log f_n)$ questions. □

5 Application aux algorithmes de tris

Théorème 2 *Dans le modèle d'arbre de décision, trier un tableau se fait en temps $\Omega(n \log n)$.*

Comme on ne peut pas vraiment accéder aux données dans ce modèle (on peut, mais en posant des questions du genre "Est-ce que $T[4] = \pi$?" qui ne sont pas réalistes) on reformule la solution d'un problème de tri : trier T revient à trouver une permutation σ de $\{0, \ldots, n-1\}$ telle que

$$T[\sigma(0)] \leq T[\sigma(1)] \leq \ldots \leq T[\sigma(n-1)]$$

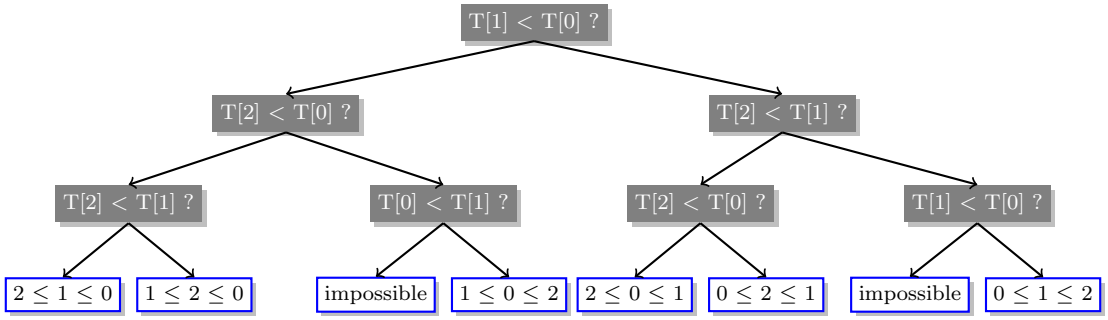
Autrement dit, à donner l'ordre à prendre sur les indices et non sur les éléments.

Prenons l'exemple du tri bulle sur un ensemble à 3 éléments. L'algorithme est :

```
def triBulle(T):
  for i in range(len(T)):
    for j in range(len(T)-i-1):
      if T[j+1] < T[j]:
        T[j], T[j+1] = T[j+1], T[j]
  return T
```

On l'adapte donc de la façon suivante (avec $s = \sigma$) :

```
def triBulle(T):
  s = list(range(len(T))) # s = identite
  for i in range(len(T)):
    for j in range(len(T)-i-1):
      if T[s[j+1]] < T[s[j]]:
        s[j], s[j+1] = s[j+1], s[j]
  return s
```



On voit que l'algorithme repose des questions dont il a déjà eu la réponse. Sa complexité dans le pire des cas ici est 3.

Démonstration : Si tous les éléments du tableau sont distincts, il y a donc $n!$ résultats possibles. Comme on regarde dans le pire des cas, et que cela peut arriver que tous les éléments soient distincts, on a donc une borne inférieure pour trier en $\Omega(\log n!)$, d'après le théorème 1.

Il reste[3] à estimer $\log n!$:

$$n! = \prod_{i=1}^{n} i \geq \prod_{i=\lfloor n/2 \rfloor}^{n} i \geq \left(\frac{n}{2}\right)^{n/2}$$

Et donc

$$\log n! \geq \frac{n}{2} \log \frac{n}{2} = \Omega(n \log n).$$

□

Habituellement le résultat est formulé comme ça :

Théorème 3 *En n'utilisant que des comparaisons d'éléments du tableau, trier un tableau se fait en temps* $\Omega(n \log n)$.

Mais notre théorème est plus puissant, on autorise des question du genre "Est-ce que les $\frac{n}{3}$ premiers éléments du tableau sont triés ?", ou n'importe quelle autre question dont la réponse est oui ou non.

En conséquence du théorème 2, les algorithmes de tri fusion et tri par tas sont optimaux.

[3] Si on connaît la formule de Stirling, c'est direct.

6 Conséquence : enveloppe convexe

Théorème 4 *Dans le modèle de l'arbre de décision, calculer l'enveloppe convexe en ordre circulaire d'un ensemble de n points se fait en temps $\Omega(n \log n)$.*

Démonstration : On a vu que `SORT` $\lll_{\mathcal{O}(n)}$ `CONVEXE`. Donc si on pouvait résoudre `CONVEXE` plus rapidement, on contredirait le théorème 2. □

L'algorithme de Graham pour le calcul de l'enveloppe convexe circulaire est donc *optimal.*

7 Recherche dans un tableau

Le problème classique est le suivant : étant donné un tableau T de n nombres et un nombre x, on veut savoir si $x \in T$ et si oui, quel est le i tel que $T[i] = x$.

Comme il y a $n+1$ réponses possibles, on a une borne inférieure en $\Omega(n \log n)$ dans le modèle de l'arbre de décision. Ceci vaut qu'**on suppose le tableau déjà trié ou non**.

Si on suppose le tableau déjà trié, on connaît un algorithme optimal, la *recherche dichotomique*. S'il n'y a pas d'hypothèse sur le tableau, on voit mal comment faire autrement que de chercher x dans toutes les cases de T : la borne inférieure en $\Omega(n \log n)$ ne semble pas très précise...

8 Remarque : hachage

Mais au fait, on peut faire une recheche dans un ensemble en temps moyen constant si on utilise une table de hachage. Donc la borne inférieure est fausse si on représente l'ensemble ainsi, bien qu'il y ait toujours $n + 1$ (on se rappelle de l'indice dans le tableau).

Il n'y a pas d'incohérence : c'est juste que la recherche par hachage n'entre pas dans le modèle de l'arbre de décision. Les questions sont du type "quelle est la valeur de hachage de x ?", qui n'admet pas un nombre borné de réponses. Plus précisément, si on borne la taille de la table de hachage par une constante k qui ne dépend pas de n, on retrouve la borne inférieure en $\Omega(n \log n)$: il faut faire augmenter la taille de la table avec n pour avoir de meilleures complexités.