Complexité implicite du tri

Benjamin Hellouin

Cormen, "Sorting in linear time"

On veut établir une borne inférieure sur le temps utilisé par un algorithme de tri "par comparaison", c'est-à-dire qui ne prend pas en compte les propriétés spécifiques des éléments à trier. Plus formellement, on suppose qu'on a un ensemble totalement ordonné $(E, \leq)$ et un algorithme de tri sur E qui n'utilise que des comparaisons pour avoir des informations sur l'ordre des éléments $(a_i \leq a_j, a_i < a_j)$. Autrement dit, l'algorithme n'effectue qu'une suite de tests et son comportement ne dépend que du résultat de ces tests.

On note une instance quelconque $(a_1 \dots a_n)$, et on suppose sans perte de généralité que ses éléments sont distincts (une borne inférieure pour ces cas est en particulier une borne inférieure pour le cas général). L'action d'une permutation π sur ce tableau correpond $a_{\pi(1)} \leq \dots \leq a_{\pi(n)}$: le problème se ramène à trouver la permutation qui trie le tableau. La supposition précédente permet de représenter l'algorithme sous forme d'un arbre binaire de décision : on étiquète chaque noeud par un couple (i, j) et chaque feuille par une permutation. On dit qu'un tel arbre rdécrit le comportement de l'algorithme si (définition inductive) :

- la racine est étiquetée (i, j) et, sur toute entrée, l'algorithme commence par tester si $a_j \leq a_j$;
- si le résultat est négatif, le fils droit décrit le comportement futur de l'algorithme ;
- si le résultat est positif, c'est le fils gauche

Enfin, une feuille étiquetée par σ décrit le fonctionnement de l'algorithme si l'algorithme termine et renvoie le tableau correspondant à σ. On peut bien associer un tel arbre à un algorithme qui se comporte comme on l'a décrit car . Le comportement de l'algorithme sur une instance correspond alors à un chemin de la racine à une feuille dans cet arbre.

Lemme 1. *Si un algorithme est correct, alors l'ensemble des permutations de $\mathcal{S}_n$ apparait parmi les feuilles de son arbre de décision.*

C'est tout à fait intuitif : si une permutation σ n'apparait pas dans les feuilles de son arbre de décision, alors il est impossible que cet algorithme trie correctement une liste qui est triée par cette permutation. Une telle liste existe évidemment : si $a_1 \dots a_n$ est triée, alors $a_{\sigma^{-1}(1)} \dots a_{\sigma^{-1}(n)}$ est une telle liste.

Théorème 1. *Tout algorithme de tri par comparaison nécessite $\Omega(n \log n)$ dans le pire cas.*

En effet, le lemme précédent montre que l'arbre de décision associé à cet algorithme comporte au moins $n!$ feuilles. Or un arbre binaire de heuteur h

comporte au plus 2^h feuilles (récurrence immédiate). On en déduit $n! \leq 2^h$, soit $h \geq \log(n!) = \Omega(n \log n)$. De plus, la hauteur de l'arbre correspond au nombre de comparaisons faites par l'algorithme dans le pire cas (l'instance correspondant à ce chemin).

Remarque. Dans le cas où des informations supplémentaires sont disponibles, cette remarque n'est pas valable. Par exemple, il existe un algorithme qui, pour k fixé, trie en temps linéaire tout tableau dont les éléments sont compris entre 0 et k.