Complexité d'un algorithme

En cours de rédaction

Source : Données et algorithmes (Mooc Fun)

Contexte : recherche d'un mots dans un tableau.

Cas 1 : tableau non trié (recherche exhaustive ou par force brute)

Enoncé

1. Pour tester la présence d'un mot, on le compare au premier élément du tableau, puis le cas échéant au second, au troisième etc. L'algorithme se termine de deux façons possibles :
2. On a trouvé le mot à l'indice i dans le tableau.
3. On a atteint la fin du tableau sans trouver le mot.

Algorithme

A faire

Complexité

Le point important dans l'évaluation du temps d'exécution est le nombre de comparaisons que l'on va effectuer : on peut voir l'opération de comparaison entre deux mots comme une sorte d''« unité de mesure » du temps nécessaire. Essayons alors d'évaluer ce nombre, en considérant que le tableau contient **N mots**.

Deux cas possibles :

- Si le mot est correct, alors le stockage aléatoire des mots dans le tableau fait que la probabilité de trouver le mot à n'importe quelle place dans le tableau est la même : en moyenne, on fera donc **N/2 comparaisons** (i.e. le mot est en moyenne au milieu du tableau).
- Si le mot n'est pas correct, alors on dois tester tous les éléments du tableau soit **N comparaisons**.

Même si le traitement est plus rapide (statistiquement deux fois plus) dans le cas où le mot est correct, le temps nécessaire reste dans les deux cas **proportionnel à la taille N** du tableau : si on double cette taille, alors on double le temps nécessaire pour répondre.

Cas 2: tableau trié (recherche par dichotomie)

Enoncé

On compare le mot cherché à l'élément central du tableau. * Si le mot cherché est avant l'élément central, on sait qu'il suffit de regarder dans la première moitié du tableau. * Symétriquement, si le mot cherché est après l'élément central, on sait qu'il suffit de regarder dans la deuxième moitié du tableau.

Algorithme

A faire

Complexité

Le point important ici est que, dans les deux cas où on échoue, la moitié des éléments du tableau a été éliminé. Si l'on répète l'opération ci-dessus sur la moitié restante, il nous restera au pire un quart du tableau initial ; si on la répète à nouveau (pour la troisième fois donc), il restera un huitième du tableau initial, etc : la taille du tableau restant diminue d'un facteur 2 à chaque comparaison !

On voit donc que pour **i comparaisons**, la taille du tableau restant (dans lequel le mot est susceptible de se trouver) est de **N/2i**. Nous pourrons nous arrêter lorsque cette taille sera 1, puisque là il sera facile de dire si le mot est correct ou non (c'est ou ce n'est pas le seul élément du tableau restant). Pour que N/2i soit égal à 1, il faut que i soit égal à **$\log_2(N)$** (le logarithme à base 2 ; c'est une fonction qui croît très lentement, sa valeur augmentant de 1 lorsque son paramètre est doublé, avec $\log_2(1) = 0$).

Performance comparée des deux méthodes

On effectue les tests sur un tableau de 100000 mots. On suppose que le temps nécessaire à la comparaison de deux mots est de 1ms.
On se place dans la situation la plus favorable.

- **Cas 1** : t = N/2 x tc = $(10^5 / 2) \times 10^{-3}$ = **50s!**
- **Cas 2** : t = $\log_2(N) \times tc = \log_2(10^5) \times tc = 5 \log_2(10) \times 10^{-3}$ ~= **17 ms**.