

Complexité

Lors du chapitre concernant les tris, vous vous êtes aperçu que certains d'entre eux étaient plus ou moins rapides en fonction de leurs entrées. Mais n'était-il pas possible de le savoir avant ? L'objectif de ce chapitre est d'étudier un algorithme afin d'en évaluer sa complexité, c'est-à-dire prédire sa rapidité d'exécution ou sa consommation mémoire.

1 Introduction

L'étude de la complexité d'un algorithme a plusieurs objectifs. Le premier est la comparaison d'algorithmes. Il existe souvent plusieurs solutions pour résoudre un même problème. Dès lors, il est intéressant de trouver la meilleure. Ce choix dépend alors souvent du temps que va mettre le programme à trouver une solution. Ce temps étant entièrement dépendant l'architecture matérielle sur laquelle s'exécute le programme, on préfère étudier le nombre d'opérations réalisées au cours du traitement.

TD Exercices 1 et 2.

Le dernier exercice vous a montré une chose intéressante : un algorithme semble posséder moins d'opérations qu'un code. La réalité fait que ce qui est exécuté sur un ordinateur est bien un code et non un algorithme. De plus, suivant le code, le nombre d'opérations peut être différent. Cela signifie que ce qui nous intéresse est le nombre « moyen » d'opérations. Ce nombre d'opération doit donc être indépendant du langage, et doit être propre à l'algorithme.

L'étude de complexité d'un algorithme doit donc nous fournir une information permettant d'évaluer le temps de déroulement d'un algorithme, quelles que soient les entrées. Dans certains cas, il est possible que des entrées entraînent une exécution plus rapide de l'algorithme. Par exemple, un tableau déjà presque ordonné peut être trié bien plus rapidement qu'un tableau totalement désordonné. L'étude de la complexité d'un algorithme ne doit donc pas dépendre non plus des entrées.

Dès lors, on introduit deux types de calculs. Le premier évalue la complexité *dans le pire cas*. Il s'agit d'étudier le nombre d'opérations réalisées lorsque l'organisation des entrées est la moins favorable à la rapidité d'exécution du programme. Ce cas est crucial puisqu'il permet de fiabiliser l'algorithme : il se terminera en un temps maximal précisé. On peut également s'intéresser à la *complexité moyenne*. L'idée est de calculer la distribution moyenne des données afin de savoir combien de temps prendra l'algorithme *en moyenne*. Si l'algorithme est appelé un grand nombre de fois, la prévision du temps d'exécution en est facilitée.

TD Exercice 3.

2 Étude de complexité

En pratique, une complexité se détermine en fonction de la taille des entrées. L'idée est d'évaluer le nombre d'opérations que va réaliser l'algorithme pour traiter une certaine taille d'entrée. Par exemple, un algorithme qui recherche l'élément minimal dans un tableau mettra deux fois plus de temps si la taille du tableau double. Autre exemple, la génération d'un nombre aléatoire entre 0 et N prendra le même temps et le même nombre d'opérations que la génération d'un nombre aléatoire entre 0 et $10N$. On introduit alors la notation suivante.

Définition 1 (grand O). Soient f et g deux fonctions positives sur \mathbb{N} . On dit que f est en grand O de g s'il existe deux entiers positifs c et N_0 tels que pour tout entier $N > N_0$, on a $f(N) < cg(N)$.

Notation 1 (de Landau). Lorsque f est en grand O , on utilise la notation de Landau et on écrit $f = O(g)$.

Autrement dit. une fonction f est en grand O de g si f est bornée à constante près par g . On dit aussi que f est *dominée* par g en $+\infty$. Par exemple, la fonction réelle $f : x \mapsto x^2$ est en $O(g)$ avec $g : x \mapsto e^x$ ou avec $g : x \mapsto 2x^2$.

Cela signifie que toute fonction f qui croît asymptotiquement (pour des valeurs de x très grandes) moins vite qu'une fonction g est en $O(g)$. Pour une étude de complexité, on cherche à déterminer une fonction g « de référence », immédiatement supérieure à f .

Voyons la liste des fonctions références que nous considérerons à l'avenir. Soit N un entier « grand ». On définit les notations de Landau suivantes classées par complexité croissante.

Notation	Appellation	Exemples d'algorithmes
$O(1)$	Constante	Temps d'accès à un élément d'un tableau
$O(\log N)$	Logarithmique	Jeu du juste prix (la vitrine)
$O(N)$	Linéaire	Parcours d'un tableau
$O(N \log N)$	Quasi-linéaire	Tri rapide (à découvrir au S2)
$O(N^2)$	Quadratique	Tris à bulles, insertion et sélection
$O(N^c)$	Polynomiale	$c > 2$, c parcours imbriqués
$O(c^N)$	Exponentielle	$c > 1$ Énumération complète
$O(N!)$	Factorielle	Génération des permutations de N éléments

TD Exercices 4 à 6.

Dès lors, vous devrez évaluer la complexité de tous les algorithmes que vous croiserez sur votre chemin avant de les coder. L'idée est de vérifier avant de coder que le programme s'arrêtera. Pour évaluer la complexité d'un algorithme, la méthodologie est la suivante.

Méthode 1 (complexité d'un algorithme). On suppose que l'on veut étudier la complexité d'un algorithme.

1. Trouver le N , c'est-à-dire déterminer la taille de l'entrée de l'algorithme ayant le plus d'impact sur son temps d'exécution.
2. Compter le nombre d'opérations présentes dans l'algorithme afin d'extraire une fonction f dépendante du N .
3. Évaluer la fonction f quand N tend vers $+\infty$ et ne considérer que le terme de plus haute complexité.

Exemple 1. Évaluons la complexité du tri à bulles non optimisé. On rappelle tout d'abord son algorithme.

Algorithme 1 – Tri à bulles


Entrée. Tableau d'entiers T de n éléments

Sortie. Tableau d'entiers T trié

```
Pour i allant de 0 à n-1 faire
  Pour j allant de 0 à n-2 faire
    Si  $T[j] > T[j+1]$  alors
      Échanger ( $T[j], T[j+1]$ )
    Fin Si
  Fin Pour
Fin Pour
```

Déterminons tout d'abord l'entrée et sa taille. Ici, le cas est simple puisqu'il s'agit d'un tableau de n éléments. Plus ce tableau sera grand, plus l'algorithme mettra de temps à se dérouler. Le type du tableau n'a ici aucune importance.

Déterminons maintenant le nombre d'opérations qui est fonction de n . La première boucle (ligne 1) indique que l'on réalise n fois une série d'opérations. La seconde boucle (ligne 2), indique que l'on réalise $n-1$ fois une série d'opérations. Les opérations (ligne 4) dépendent de la condition de la ligne 3. Dans le pire des cas, on peut considérer que cette condition est toujours vérifiée.

L'échange (k opérations) est réalisé $n \times (n-1)$ fois. On obtient donc approximativement $k \times n \times (n-1) = kn^2 - kn$ opérations. Lorsque n tend vers $+\infty$, kn devient négligeable devant kn^2 . La valeur k étant une constante, elle est négligée. La complexité du tri à bulles est donc en $O(n^2)$, avec n la taille du tableau. 

TD Exercice 7.

3 Exercices d'approfondissement

TD Exercices 8 à 10.