COURS

Algorithmique — Complexité et algorithmes fondamentaux

1re

Programme de première générale, enseignement de spécialité — BO spécial n°1 du 22 janvier 2019

Sur mille éléments, un algorithme quadratique fait un million d'opérations là où une recherche dichotomique en fait dix. L'écart n'est pas une question de machine rapide ou lente : c'est une loi de croissance, et aucun processeur ne la rattrape. Savoir la calculer, c'est savoir quel programme tiendra quand les données grossiront.

1 Ce que mesure la complexité

RÈGLE

La **complexité** mesure la **croissance du coût** quand la taille de l'entrée augmente : c'est une **loi d'évolution**, pas une durée en secondes. On la note **O(n)** et on ignore les constantes multiplicatives et les termes de moindre ordre, parce qu'ils cessent de compter quand l'entrée grandit. Les classes usuelles, par coût croissant : **O(1)**, O(log n), **O(n)**, O(n log n), O(n²).

$$O(1) < O(logn) < O(n) < O(nlogn) < O(n^2)$$

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Comparer deux algorithmes par leur temps mesuré sur un petit jeu de données. — Sur une petite entrée, les **constantes** dominent : un algorithme quadratique bien optimisé peut battre un algorithme en O(n log n) mal implémenté. Mais l'ordre s'inverse **nécessairement** à partir d'une certaine taille, et l'écart devient ensuite irrattrapable. Un chronomètre sur mille éléments ne dit rien du comportement sur un million.

Le réflexe : Comparer des **lois de croissance**, pas des mesures ponctuelles.

EXEMPLE

Comparer le nombre d'opérations d'un algorithme en O(n²) et d'une recherche par dichotomie, pour n = 1 000.

1. Pour l'algorithme quadratique, l'ordre de grandeur est n × n :
   $1000 \times 1000 = 1000000$
2. Soit environ **un million** d'opérations.
3. La **dichotomie** divise l'espace de recherche par deux à chaque étape : il faut environ dix étapes, puisque 2 puissance 10 dépasse 1 000.
   $2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 = 1024$
4. Soit **dix** opérations contre un million : le rapport est de cent mille.

Environ 1 000 000 contre 10

2 La recherche dichotomique

RÈGLE

La **dichotomie** cherche une valeur en comparant à l'élément **central** : si la valeur cherchée est plus petite, on élimine toute la moitié droite, sinon la moitié gauche. Chaque comparaison **divise par deux** l'espace restant, d'où une complexité en O(log n). Elle exige une condition **impérative** : le tableau doit être **trié**.

$$chaqueétapedivisel'espacederecherchepar2$$

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Appliquer une recherche dichotomie à un tableau qui n'a pas été trié au préalable. — L'algorithme **élimine une moitié** sur la seule foi de la comparaison centrale. Sans tri, rien ne garantit que la valeur cherchée ne s'y trouvait pas : l'algorithme ne **plante pas**, il répond simplement « absent » pour un élément présent. C'est une faute silencieuse, qui ne se manifeste que par des résultats faux sur certaines entrées.

Le réflexe : Avant toute **dichotomie**, vérifier — ou établir — que le tableau est **trié**.

3 Les tris

PROPRIÉTÉ

Le **tri par insertion** construit la partie triée élément par élément, en insérant chaque nouvel élément à sa place. Sa complexité est en $O(n^2)$ dans le cas défavorable, mais il est **efficace sur de petits tableaux** et sur des données presque triées. Le **tri fusion** applique le principe « diviser pour régner » : couper le tableau en deux, trier chaque moitié, puis **fusionner** les deux moitiés triées. Sa complexité est en $O(n \log n)$ dans **tous** les cas.

Le **tri fusion** illustre ce qu'apporte la récursivité : la fusion de deux listes déjà triées est facile, et c'est cette facilité qu'on obtient en découpant. Il a un coût que le **tri par insertion** n'a pas : il utilise de la **mémoire supplémentaire** pour la fusion. Choisir un tri, c'est donc arbitrer entre temps et mémoire, et non désigner un vainqueur absolu.

4 Les graphes

PROPRIÉTÉ

L'algorithme de **Dijkstra** calcule le plus court chemin depuis un sommet de départ dans un graphe dont les arêtes portent des **poids positifs**. Son principe : maintenir pour chaque sommet la meilleure distance connue, traiter à chaque étape le sommet **non traité le plus proche**, et mettre à jour ses voisins. La condition de positivité n'est pas un détail : avec des poids négatifs, l'algorithme peut figer une distance qu'un chemin ultérieur aurait améliorée.

ÉVALUER LA COMPLEXITÉ D'UN ALGORITHME

1. Identifier la **taille de l'entrée**, notée n : nombre d'éléments, de sommets, de caractères.
2. Repérer les **boucles** et leur imbrication : deux boucles imbriquées sur n donnent $O(n^2)$.
3. Repérer les découpages **par deux** : ils introduisent un logarithme.
   $1000 \times 1000 = 1000000$
   Deux boucles imbriquées sur 1 000 éléments donnent un million d'opérations.
4. Ignorer les **constantes** et les termes dominés : $O(3n + 5)$ s'écrit **O(n)**.
5. Conclure en précisant s'il s'agit du cas **favorable**, **moyen** ou **défavorable**.

À VÉRIFIER

- Ai-je identifié la **taille de l'entrée** avant de compter ?
- Ai-je compté l'**imbrication** des boucles, et non leur nombre ?
- Ai-je **ignoré les constantes** dans ma notation ?
- Ai-je précisé le **cas** — favorable, moyen, défavorable ?
- Mon tableau est-il **trié** avant toute **dichotomie** ?
- Ai-je vérifié que les poids sont **positifs** avant d'appliquer **Dijkstra** ?

À RETENIR

- La **complexité** est une **loi de croissance**, pas une durée.
- **O(1)** < O(log n) < **O(n)** < O(n log n) < $O(n^2)$.
- Ignorer les **constantes** : $O(3n + 5)$ s'écrit **O(n)**.
- Boucles **imbriquées** sur n : $O(n^2)$. Découpage **par deux** : logarithme.
- La **dichotomie** exige un tableau **trié** — sinon faute **silencieuse**.
- Chaque étape de dichotomie **divise par deux** l'espace de recherche.
- **Tri par insertion** : $O(n^2)$ au pire, efficace sur petit ou presque trié.
- **Tri fusion** : $O(n \log n)$ dans tous les cas, au prix de **mémoire** supplémentaire.
- **Dijkstra** : plus court chemin, à condition de **poids positifs**.
- Comparer des **lois**, jamais des chronomètres sur petites entrées.
- Aucun algorithme n'est meilleur **dans l'absolu** : tout dépend du contexte.