COURS

Le paradigme diviser pour régner

Tle

Programme de terminale générale, enseignement de spécialité — BO spécial n°8 du 25 juillet 2019

Couper un problème en deux ne le rend pas automatiquement plus facile. Si l'une des moitiés contient presque tout, ou si recoller les morceaux coûte aussi cher que résoudre le problème entier, on a fait beaucoup de travail pour rien. Le paradigme n'est efficace que sous conditions, et c'est de ces conditions qu'il s'agit.

1 Le principe

RÈGLE

Diviser pour régner décompose un problème en **sous-problèmes** de même nature mais de taille moindre, les résout — le plus souvent par **récurrence** —, puis **combine** les solutions partielles. Trois étapes, donc : diviser, régner, combiner. Le gain ne vient pas de la division en elle-même, mais du fait que la somme des coûts des sous-problèmes **plus** le coût de combinaison soit inférieure au coût direct.

$$diviser \cdot régner \cdot combiner$$

RÈGLE

Le paradigme ne fait gagner que sous **deux conditions**. Les sous-problèmes doivent être de **tailles comparables** : couper en deux moitiés égales fait chuter la profondeur à log n, couper en un élément et le reste ramène au parcours linéaire. Et l'étape de **combinaison** doit rester **peu coûteuse** : si recoller coûte autant que résoudre, le découpage n'apporte rien.

$$taillescomparables + combinaisonbonmarché$$

LE PIÈGE QUI COÛTE LE PLUS DE POINTS

Considérer qu'un algorithme relevant de diviser pour régner est nécessairement plus efficace qu'une approche directe. — Le paradigme est une **méthode de conception**, pas une garantie de performance. Un découpage **déséquilibré** annule le gain : le **tri rapide** mal pivoté dégénère en O(n²), exactement comme un tri élémentaire. Et une **combinaison** coûteuse peut absorber tout le bénéfice. Annoncer une complexité parce qu'on reconnaît le paradigme, c'est confondre la forme de l'algorithme et son coût.

Le réflexe : Vérifier les **deux conditions** avant d'annoncer la moindre complexité.

2 Trois exemples classiques

PROPRIÉTÉ

La recherche **dichotomique** divise l'espace de recherche en deux et n'en garde **qu'une** moitié : il n'y a donc rien à combiner, d'où une complexité en O(log n). Le **tri fusion** divise en deux moitiés égales, trie chacune, puis **fusionne** en temps linéaire : la garantie de tailles égales et la combinaison bon marché donnent O(n log n) **dans tous les cas**. Le **tri rapide** partitionne autour d'un **pivot**, et les deux parties n'ont aucune raison d'être égales.

Le **tri rapide** illustre exactement ce que les deux conditions signifient. Sa combinaison est **gratuite** — après partition, les deux parties sont déjà à leur place relative —, ce qui est un avantage sur le **tri fusion**. Mais l'équilibre des tailles dépend du **pivot** : bien choisi, O(n log n) en moyenne ; systématiquement mal choisi, O(n²). D'où les stratégies de choix de pivot, qui sont tout l'enjeu de son implémentation.

3 Calculer la complexité

PROPRIÉTÉ

La complexité se calcule par une relation de **récurrence** : le coût pour une entrée de taille n s'exprime en fonction du coût des sous-problèmes et du coût de combinaison. L'**équation de maître** donne le résultat selon lequel des deux l'emporte. Trois cas : si la combinaison domine, elle fixe la complexité ; si les sous-problèmes dominent, ce sont eux ; et s'ils s'équilibrent, un facteur logarithmique apparaît — c'est le cas du **tri fusion**.

$$coût(n) = a \times coût(n/b) + combinaison(n)$$

CONCEVOIR UN ALGORITHME DIVISER POUR RÉGNER

1. Identifier un découpage produisant des sous-problèmes **de même nature** que l'original.
2. Vérifier que les sous-problèmes sont de **tailles comparables** — sinon le gain disparaît.
3. Écrire le **cas de base** : la plus petite entrée, résolue sans découpage.
4. Écrire l'étape de **combinaison** et en estimer le coût.
   $1000000 \div 20000 = 50$
   Sur un million d'éléments, n log n vaut environ 20 millions contre 10^{12} pour n^2 : le rapport dépasse 50 000.
5. Poser la relation de **récurrence** et conclure, plutôt que d'annoncer une complexité par reconnaissance de forme.

À VÉRIFIER

- Mes sous-problèmes sont-ils de **même nature** que le problème initial ?
- Sont-ils de **tailles comparables** ?
- Le coût de la **combinaison** est-il inférieur à celui d'une résolution directe ?
- Ai-je écrit un **cas de base** atteint à coup sûr ?
- Ai-je posé la **récurrence** au lieu d'annoncer une complexité de mémoire ?
- Ai-je vérifié le cas **défavorable**, et pas seulement le cas moyen ?

À RETENIR

- **Diviser pour régner** : **diviser**, **régner**, **combiner**.
- Deux conditions : sous-problèmes de **tailles comparables**, **combinaison** bon marché.
- Le paradigme est une **méthode de conception**, pas une garantie de performance.
- **Dichotomie** : une seule moitié gardée, donc rien à combiner — O(log n).
- **Tri fusion** : moitiés égales **par construction**, O(n log n) dans **tous** les cas.
- Il coûte de la **mémoire** supplémentaire : il n'opère pas sur place.
- **Tri rapide** : combinaison **gratuite**, mais équilibre dépendant du **pivot**.
- Mal pivoté systématiquement, il dégénère en **O(n²)**.
- La complexité se calcule par une **récurrence**, pas par reconnaissance de forme.
- L'**équation de maître** tranche selon ce qui domine : sous-problèmes ou combinaison.
- Aucun tri ne l'emporte dans l'absolu : garantie contre performance moyenne.