Exemples d'algorithmes utilisant la méthode «diviser pour régner». Exemples et Applications.

Motivation Premières notions d'algorithmiques, présenter le « diviser pour regner » permet d'introduire de façon très fluide la prorgammation dynamique, on peut profiter de cette leçon pour sensibiliser aux enjeux de complexités lorsque l'on compare des algos, et même sur la complexité en espace.

I. Introduction

Définition 1 **« Diviser pour régner »** est une méthode algorithmique permettant de résoudre un problème en le séparant en sous-problèmes de taille moindre. Ces sous-problèmes sont résolus, puis leurs solutions sont combinées afin de résoudre le problème initial.

Méthode 2 **Schémas de la résolution DPR.**

- Diviser : décomposer le problème en un ou plusieurs sous-problèmes.
- Régner : résoudre les sous-problèmes individuellement
- Fusionner : combiner les solutions des sous-problèmes pour obtenir celle du problème initial

Remarque 3 Pour résoudre un sous-problème on considère deux cas:

- Cas de base : instances triviales du problème, pour les tailles les plus petites, pour lesquelles on peut exprimer directement les solutions.
- Cas récursif : autres cas, où le problème sera divisé et les sous-problèmes résolus récursivement, jusqu'à atteindre un cas de base.

Problème 4 **Recherche dans un tableau trié**

- Entrée: Un tableau trié l, une valeur x.
- Sortie: Oui si x appartient au tableau, Non sinon.

Algorithme 5 **Recherche dichotomique**

Remarque 6

- Il est souvent question de résoudre les sous-problèmes récursivement dans un algorithme suivant l'approche DPR.
- Lors de l'implémentation d'un programme DPR récursif, il convient de se méfier d'un potentiel dépassement de pile lors de l'éxécution.
- Il est toujours possible d'écrire un algo DPR dans un style impératif, il en résulte un code moins naturel mais qui n'est pas soumis à la contrainte du dépassement de pile.

Méthode 7 **Complexité.** Si l'on découpe le problème initial en k problèmes de tailles $(n_i)_{1\leq i\leq k}$ et la division et fusion des problèmes demande un coût $f(n)$, on obtient la relation de récurrence suivante pour déterminer la complexité $C(n)$ d'un algorithme DPR: $C(n) = f(n) + \sum_{i=1}^{k} C(n_i)$.

Remarque 8 Souvent la relation de récurrence est très simple. Par exemple dans le cas où l'on traite les deux moitiés d'un problème : $C(n) = f(n) + 2 * C(\frac{n}{2})$.

Application 9 **Recherche dichotomique** $C(n) = C(\frac{n}{2}) + \alpha$. Donc $C(2^p) = C(2^{p-1}) + \alpha = ... = C(2^0) + p.\alpha$ et $C(n) = \Theta(\log n)$.

Théorème 10 **Théorème Maitre(Admis)** [COR3 4.5 4.6] . De manière générale, pour calculer le coût d'un algorithme qui découpe un problème de taille N en a sous-problèmes de taille $\frac{N}{b}$ et tel que $f(N) = \Theta(N^c)$, posons $c_{\text{crit}} = \log_{b(a)}$:

- si $c < c_{\text{crit}}, C(N) = \Theta(N^{c_{\text{crit}}})$
- si $c = c_{\text{crit}}, C(N) = \Theta(N^c \log N)$
- si $c > c_{\text{crit}}, C(N) = \Theta(N^c)$

Remarque 11 Ce théorème est hors programme mais peut être utile pur avoir rapidement la complexité d'un algorithme.

Remarque 12 Les algorithmes diviser pour régner ont plusieurs interêts :

- Complexité généralement faible par rapoort aux algorithmes naïfs

- Parallélisme facile à mettre en place grâce à la division en sous-problèmes

Implémentation 13 Code Multithreadé [VERT TODO] Voici un exemple d'un code C multithreadé. TODO

II. Etudes d'algos DPR

Problème 14

- Entrée: Un tableau a de valeurs.
- Sortie: Un tableau a' de avec les élments de a dans l'ordre.

Idée 15 Trier le début et la fin du tableau séparément puis combiner les deux tableaux ainsi triés.

Algorithme 16 Tri Fusion

```
FUSION(a1, a2, l, m, r):
  # tableaus a1 et a2, indices l, m et r
  # l <= m <= r et a1[l, m[ et a2[m, r[ sont triés
  i <- j, j <- m
  for k = l .. r -1 do
    if i < ù et (j = r)
  ...
  ...
# trouver un bouquin avec le code dedans
  ...
```

Remarque 17 Le coût de séparation est en 0(1), celcui de la combinaison en $\mathcal{O}(n)$: $C(n) = 2 * C(\frac{n}{2}) + \mathcal{O}(n)$

Propriété 18 Complexité. $C(n) = \mathcal{O}(n \log n)$.

Idée 19 Trier les plus grands éléments et les plus petits éléments séparément puis combiner les deux parties ainsi triés.

DEV

Algorithme 20 Tri rapide

```
TriRapide():
 # todo trouver bouquin avec le code dessus
```

Remarque 21 Le coût de la division est en $\mathcal{O}(n)$, celui de la fusion est en $\mathcal{O}(1)$. Il n'y aura cependant pas toujours le même nombre d'éléments chez les grands et les petits, on parle de complexité en moyenne.

Propriété 22 Complexité. La complexité au pire cas du tri rapide est en $\mathcal{O}(n^2)$, comme les tris naïfs (insertion, selection). Cependant sa complexité en moyenne est en $\mathcal{O}(n \log n)$.

Remarque 23 Dans le pire cas, le pivot est toujours le plus grand élément. Cela n'arrive sur la liste triée dans l'ordre inverse. On fait alors pareil que le tri par insertion dans son pire cas.

Remarque 24 Cependant, si l'implémentation est récursive, ce pire cas peut facilement mener à un débordement de pile. Pour éviter cela, on peut traiter le sous-tableau le plus petit en premier si on bénéficie de l'optimisation pour les appels terminaux.

Problème 25 Multiplication de deux nombres à n chiffres.

Algorithme 26 Karatsuba. On remaque que $\big(a \times 10^k\big)\big(c \times 10^k + d\big) = ac \times 10^k + (ac + bd - (a-b)(c-d)) \times 10^k + bd$ ce qui ne demande que trois multiplications. L'algorithme est alors:

- Diviser: si x et y sont à $2n$ chiffres, les écrire $x = a \times 10^k + b$
- Regner: Lancer l'algorithme récursivement sur les entrées $(a, c), (b, d)$ et $(a - b, c - d)$
- Fusionner: Calculer $ac \times 10^{2n} + [ac + bd - (a-b)(c-d)] \times 10^k + bd$.

Propriété 27 Complexité. On a donc besoin de $\mathcal{O}\big(n^{\log_3 n}\big)$ multiplications (les autres opérations sont en temps négligeable).

DEV

Problème 28 Deux points les plus proches.

- Entrée: une liste L de points du plan $p_i = (x, y)$.
- Sortie: $(x, y), (x', y')$ minimisant la distance.

Propriété 29 Complexité. Même équation que pour le tri fusion : $\mathcal{O}(n \log n)$.

III. Limites de la méthode DPR

A. Gain asymptomatique

Remarque 30 Les algos DPR présentés ont une meilleure complexité asymptotique que leur homologue naïf, cependant il arrive que sur de petites instances la version naïve s'éxécute plus rapi-

dement : cela peut par exemple être dû aux constantes cachées dans la notation de $\mathcal{O}$.

Exemple 31 Algorithmes Galactiques. Ce sont des algorithmes optimaux mais que pour des entrées beaucoup trop grandes (par exemple, plus grandes que le nombre d'atomes de l'univers).

Idée 32 La division en sous-problèmes peut s'arrêter lorsque la taille des problèmes est suffisamment petites pour être avantageuse pour l'algo naïf.

B. Recoupement des sous-problèmes

DEV

Application 33 La Pyramide d'entiers (Seam Carving) [TOR]

- Entrée: Une pyramide d'entiers de hauteur n.
- Sortie: Un chemin descendant de somme maximale.

Propriété 34 (Complexité) On a un algo naïf en $\mathcal{O}(n * 2^n)$ et un algo en DPR en $\mathcal{O}(2^n)$.

Définition 35 (Mémoïsation) On appelle mémoïsation le fait de stocker les résultats déjà calculés pour pouvoir y accéder rapidement à l'avenir.

Propriété 36 Complexité. En mémoïsant l'algo DPR on obtient une complexité en $\mathcal{O}(n^2)$

Remarque 37 Mémoïser un algo DPR est une des façons de faire ce que l'on appelle de la programmation dynamique.

Remarque 38 La programmation dynamique est une stratégie algorithmique plus adaptée dans le cas où les sous-problèmes se chevauchent.

Exemples d'algorithmes utilisant la méthode «diviser pour régner». Exemples et Applications.

I. Introduction

1 Def « Diviser pour régner »

2 Métho Schémas de la résolution DPR.

3 Rem

4 Prob Recherche dans un tableau trié

5 Algo Recherche dichotomique

6 Rem

7 Métho Complexité.

8 Rem

9 App Recherche dichotomique

10 Thm Théorème Maitre(Admis) [COR3 4.5 4.6].

11 Rem

12 Rem

13 Implem Code Multithreadé [VERT TODO]

II. Etudes d'algos DPR

14 Prob

15 Idée

16 Algo Tri Fusion

17 Rem

18 Prop Complexité.

19 Idée

20 Algo Tri rapide

21 Rem

22 Prop Complexité.

23 Rem

24 Rem

25 Prob

26 Algo Karatsuba.

27 Prop Complexité.

28 Prob Deux points les plus proches.

29 Prop Complexité.

III. Limites de la méthode DPR

A. Gain asymptomatique

30 Rem

31 Ex Algorithmes Galactiques.

32 Idée

B. Recoupement des sous-problèmes

33 App La Pyramide d'entiers (Seam Carving) [TOR]

34 Prop (Complexité)

35 Def (Mémoïsation)

36 Prop Complexité.

37 Rem

38 Rem

Remarque

Différents livres parlent de la méthode « diviser pour regner » [COR3 4] , [TOR 9.4.1] .

Bibliographie

[COR3] T. H. Cormen, *Introduction à l'algorithmique (3rd édition)*.

[VERT] F. Becker & O. Bournez & J. Carré & M. Liedloff & J. Reichert & G. Rozsavolgyi, *Informatique MP2I-MPI*.

[TOR] T. Balabonski & S. Conchon & J. Filliâtre & K. Nguyen & L. Sartre, *MP2I MPI, Informatique Cours et exercices corrigés*.