Regarder : "Analysis of Algorithms" Sedgewick - Flajolet

1

926

Analyse des algorithmes : complexité. Exemples.

I) Introduction

[SED]

Déf 1 : Un algorithme est une méthode permettant de résoudre un problème de façon finie, déterministe, effective et adaptée à l'implémentation par un programme informatique.

Objectif : Savoir comparer plusieurs algorithmes résolvant un même problème indépendamment de la machine qui les exécute.

[FRO]

Déf 2 : Soit A un algorithme. Parmi ses opérations, on en choisit certaines, dites opérations fondamentales, telles que le temps d'exécution de l'algorithme est toujours proportionnel à leur nombre. Étant donnée une entrée x, on définit :

- La complexité (temporelle) $C_A(x) :=$ nombre d'opérations fondamentales effectuées lors de l'exécution de A sur l'entrée x
- La complexité spatiale $C^S_A(x) :=$ le nombre de cases mémoire nécessaires à l'exécution de A sur x (sans compter l'entrée)

Exc 3 : * tri (par comparaison) d'un tableau → op. fond : comparaison
* multiplication de matrices → op. fond. : multiplication et addition de scalaires

Déf 4 : Soit A un algorithme. On note D_n l'ensemble des entrées possibles de taille n. Sa complexité dans le pire des cas est $Max_A(n) = \max\{C_A(x) : x \in D_n\}$. Sa complexité en moyenne (par rapport à $(p_k)_{k\in\mathbb{N}}$ où pour tout $k \in \mathbb{N}$, p_k est une probabilité sur les entrées de taille k) est $Moy_A(n) = \sum_{x \in D_n} p_n(x) \times C_A(x)$. Ne pas oublier ! Si toutes les entrées d'une taille donnée sont équiprobables, alors $Moy_A(n) = \frac{1}{|D_n|}\sum_{x\in D_n} C_A(x)$.
On définit de même Max^S_A et Moy^S_A.

[FRO]

Rq 5 : $Moy_A(n) \leq Max_A(n)$

Rq 6 : Il n'existe pas d'algorithme prenant en entrée A et n et calculant $Max_A(n) \in \mathbb{N} \cup \{+\infty\}$. Cependant, on a certaines majorations dont $Max_{\text{if } C \text{ then } I_1 \text{ else } I_2}(n) \leq Max_C(n) + \max(Max_{I_1}(n), Max_{I_2}(n))$.

Exc 7 : Recherche d'un zéro dans un tableau T de chiffres.

```
Recherche0(T)
  Pour i = 0 à |T|-1
    Si T[i] = 0
      Retourner oui
  Retourner non
```

Opération fondamentale : comparaison à 0

$C_R(T) = \min\{i \in [\![0, |T|-1]\!] : T[i] = 0\}$

$Max_R(n) = n$

$Moy_R(n) = \left(\sum_{k=0}^{n-1}\left(\frac{9}{10}\right)^k \frac{k}{10}\right) + \left(\frac{9}{10}\right)^n n$ en supposant les entrées équiprobables.

Not 8 : Soit $f, g \in \mathbb{R}_+^{\mathbb{N}}$. On note :

- $f(n) = O(g(n))$ si $\exists c \in \mathbb{R}_+^*, \exists n_0 \in \mathbb{N}, \forall n > n_0, f(n) \leq c\,g(n)$
- $f(n) = \Omega(g(n))$ si $g(n) = O(f(n))$
- $f(n) = \Theta(g(n))$ si $f(n) = O(g(n))$ et $f(n) = \Omega(g(n))$

Rq 9 : Connaître l'ordre de grandeur asymptotique de la complexité d'un algorithme suffit souvent à le comparer aux autres.

Intro : - Quelles sont les qualités attendues d'un algo ?
$\boxed{1}$ Correction
2 - Efficacité. But : Comment évaluer l'efficacité ?

ordre inverse

2

II) Méthodes de calcul de complexité

1) Calcul direct

[COR]

Exc 10: Tri par tas.
$Moy_{TT}(n) = Max_{TT}(n) = O(n \log n)$ } DVT ① BoF

[FRO], p. 387-395

Exc 11: Tri polyphasé : - mémoire interne rapide de taille M
- mémoire externe lente
- opération fondamentale : lecture / écriture sur la mémoire externe

$$Moy_{TP}(n) = Max_{TP}(n) = O\left(n \log\left(\frac{n}{M}\right)\right)$$

2) Algorithmes Diviser pour régner

[COR], p. 86-87

Thm 12: Soient $a \geq 1$, $b > 1$ et $T : \mathbb{N} \to \mathbb{R}_+^*$ pour tout n, $T(n) = aT(\lfloor n/b \rfloor) + f(n)$. Alors

① Si $f(n) = O(n^{\log_b a - \varepsilon})$ pour un $\varepsilon > 0$, $T(n) = \Theta(n^{\log_b a})$

② Si $f(n) = \Theta(n^{\log_b a})$, $T(n) = \Theta(n^{\log_b a} \log n)$

③ Si $f(n) = \Omega(n^{\log_b a + \varepsilon})$ pour un $\varepsilon > 0$, et si $af(\lfloor n/b \rfloor) \leq c f(n)$ pour un $c < 1$ et tout n suffisamment grand, alors $T(n) = \Theta(f(n))$

[COR]

Exc 13: • Tri fusion : $Moy_{TF}(n) = Max_{TF}(n) = O(n \log n)$
• Tri rapide : $Max_{TR}(n) = O(n \log n)$

dvpt (max, min)

[COR]

• Sélection du k-ième plus grand élément dans un tableau T : $O(n)$
• Multiplication de matrices $n \times n$: $O(n^{2,81})$

[PAP]

Exc 14: (FFT) Calcul de la transformée de Fourier discrète d'un vecteur de $\mathbb{C}^n$: $O(n \log n)$

Appli 15: Multiplication de deux polynomes de $\mathbb{C}_n[X]$: $O(n \log n)$ } DVT ②

"BoF"

3) Séries génératrices

[FRO], p. 519

Déf 16: Soit $(a_n)_{n \in \mathbb{N}} \in \mathbb{C}^{\mathbb{N}}$. On appelle série génératrice ordinaire de coefficient a_n la série $a(z) = \sum_{n=0}^{+\infty} a_n z^n$

Rq 17: On peut définir de nombreuses opérations sur les séries génératrices (sans imposer qu'elles convergent) et leur faire correspondre des opérations sur les suites correspondantes.
Par exemple : $b_n = a_{n-k} \iff b(z) = z^k a(z)$
Lorsque la série converge, l'utilisation d'outils d'Analyse permet de calculer la valeur exacte ou une estimation asymptotique des a_k.

[FRO], p. 544

Exc 18: Tri rapide : $Moy_{TR}(n) = O(n \log n)$

pas au pgme ; pas de dvpt ?
↳ Est-ce que ça a du sens de le mettre ds le plan ?!
ou le rédiger différemment

Dvpts : - Tri polyphasé
- Tri rapide
- ABRO

III) Complexité amortie

Motiv 19: On considère une pile (implémentée par une liste) initialement vide à laquelle on applique successivement m opération empiler ou tout_dépiler. On a $Max_{empiler}(n) = O(1)$ et $Max_{Tout\_depiler}(n) = O(n)$ mais $Max_{c_1,\dots,c_m}(0) = O(m)$ car chaque élément dépilé doit avoir été empilé. Une analyse plus naïve aurait donné $Max_{o_1,\dots,o_m}(0) = O(m^2)$ car la pile est au pire de taille k après k ops. Le pire cas (tout_dépiler avec beaucoup d'éléments dans la pile) ne peut pas se produire tout le temps, d'où l'idée de regarder le coût moyen d'une séquence d'opérations dans le pire cas. On parle de complexité amortie.

1) Méthode de l'agrégat

Méthode 20: On calcule $Max_{o_1,\dots,o_m}(n)$ et la complexité amortie de chaque opération est alors $\frac{Max_{o_1,\dots,o_m}(n)}{m}$.

Ex 21: Compteur binaire (sur k bits) avec une unique opération : incrémenter. Une opération incrémenter coûte au pire cas $O(k)$ mais m opérations incrémenter coûtent $O(m)$ donc la complexité amortie de incrémenter est $O(1)$.

2) Méthode comptable

Méthode 22: On donne un coût amorti $\hat{c}_i$ à chaque opération o_i, potentiellement différent de son coût réel c_i et on prouve que pour toute séquence d'opérations, $\sum_{i=1}^{m} \hat{c}_i \geq \sum_{i=1}^{m} c_i$.

Ex 23: Le type abstrait ensemble disjoint avec les opérations créer_ensemble, union et trouver peut être implémenté avec une complexité amortie de $O(m\,\alpha(n))$ (où α est l'inverse de la fonction d'Ackermann) pour m opérations portant sur n ensembles disjoints.

3) Méthode du potentiel

Méthode 24: On suppose que toutes les opérations se font sur un seul objet auquel on associe un potentiel. On note D_i l'état de la structure de données après $o_1,\dots,o_{i-1}$ et Φ le potentiel. On pose alors le coût amorti $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$. On a $\sum_{i=1}^{m} \hat{c}_i = \sum_{i=1}^{m} c_i + \Phi(D_n) - \Phi(D_0)$. On impose donc $\Phi(D_i) \geq \Phi(D_0)$.

Ex 25: Tableau de taille dynamique : dès que l'on dépasse du tableau, on double la taille. Le potentiel $\Phi(T) = 2\times(\text{nb d'élts dans } T) - (\text{taille } T)$ permet de montrer que le coût amorti de l'insertion est $O(1)$.

dvpt possible

IV) Amélioration de la complexité

1) Changement d'implémentation du type abstrait

Rq 26: Si on considère les opérations d'un type abstrait comme fondamentales, la complexité réelle sera la composition de la complexité calculée et de celle des opérations sur l'implémentation du type abstrait.

Ex 27: Dijkstra : $O((S+A)\log S)$ (tas binaire) $\rightarrow$ $O(S\log S + A)$ (tas de Fibonacci)

Dijkstra en dvpt ? (complexité superficielle)

2) Compromis espace / temps

Rq 28: Il est parfois possible de réduire la complexité temporelle en augmentant la complexité spatiale.

Ex 29: Retenir les résultats d'appels récursifs s'ils risquent d'être faits plusieurs fois.

Ex 30: Arbres binaires de recherche optimaux. On se donne $c_1 < \dots < c_m$ des clés et la probabilité qu'une clé recherchée soit égale à c_i ou dans $]c_i, c_{i+1}[$ et on cherche à minimiser le coût moyen.

À développer.

Marges : définition BP. ; [COR] ; dvpt possible ; [COR]

→ C'est bien de mettre les 3 méthodes _ Et de le faire avec 3 exemples différents

→ Donner un ex. d'algo ds lequel c'est utilisé

Transformée de Fourier rapide et multiplication de polynomes

__But__: Multiplier A et $B \in \mathbb{C}_d[X]$ donnés par leurs coefficients.

__Algo naïf__: $O(d^2)$

__Idée__: Représenter A et B par leurs valeurs en $n = d+1$ points.

$$A = a_0 + a_1 X + \dots + a_d X^d \qquad , x_0, \dots, x_d \text{ distincts } 2 \text{ à } 2$$

$$\begin{tikzcd} \boxed{a_0, \dots, a_d} \arrow[r, bend left, "\text{évaluation}"] & \boxed{A(x_0), \dots, A(x_d)} \arrow[l, bend left, "\text{interpolation}"] \end{tikzcd}$$

$$\begin{tikzcd} \boxed{\begin{array}{c} A, B \\ \dots a_k \dots b_k \dots \end{array}} \arrow[r, bend left, "1", "O(?)"'] & \boxed{\begin{array}{c} A \quad , B \\ \dots A(x_k) \dots , \dots B(x_k) \dots \end{array}} \arrow[d, "O(n)"] \\ \boxed{\begin{array}{c} C = AB \\ \dots c_k \dots \end{array}} & \boxed{\begin{array}{c} C = AB \\ \dots A(x_k) B(x_k) \dots \end{array}} \arrow[l, bend left, "2"', "O(?)"] \end{tikzcd}$$

1. Soit A un polynome de degré $n-1$. On suppose n pair.

On a $A = A_P(X^2) + X A_I(X^2)$ avec $A_P, A_I \in \mathbb{C}_{\frac{n}{2}}[X]$

__Astuce__: Choisir $x_k = \omega^k$ avec ω racine primitive $n^{\text{ème}}$ de l'unité.

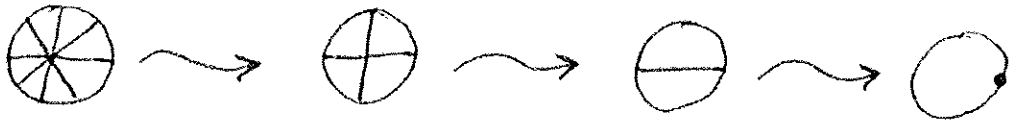
Alors $A(x_k) = A_P(x_k^2) + x_k A_I(x_k^2)$

et $A(x_{k+\frac{n}{2}}) = A(-x_k) = A_P(x_k^2) - x_k A_I(x_k^2)$

Évaluer A en n pts revient à évaluer A_P et A_I en $\frac{n}{2}$ pts $\qquad T(n) = 2T(n/2) + O(n)$

Si $n = 2^m$, en itérant, on obtient un algo. diviser pour régner en $O(n \log n)$.

Sinon on complète par des zéros avant de commencer, ça reste $O(n \log n)$.



(2) On pose $M_n(\omega) = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ \vdots & \omega & \omega^2 & \omega^{n-1} \\ \vdots & \vdots & & \vdots \\ 1 & \omega^{n-1} & \cdots & \omega^{(n-1)(n-1)} \end{pmatrix}$. Alors $\begin{pmatrix} A(\omega^0) \\ \vdots \\ A(\omega^{n-1}) \end{pmatrix} = M_n(\omega) \begin{pmatrix} a_0 \\ \vdots \\ a_{n-1} \end{pmatrix}$ (2)

$v_1 \ \cdots \ v_n$

$$\langle v_p, v_q \rangle = \sum_{k=1}^{n} \omega^{p(k-1)} \overline{\omega^{q(k-1)}}$$

$$= \sum_{k=1}^{n} \left(\omega^{(p-q)}\right)^{(k-1)}$$

$$= \begin{cases} n & \text{si } i = j \\ \dfrac{(\omega^{(i-j)})^n - 1}{\omega^{(i-j)} - 1} = 0 & \text{si } i \neq j \end{cases}$$

Donc $M_n(\omega)(M_n(\omega))^* = n\,Id$

Or $M_n(\omega)^* = \overline{M_n(\omega)} = M_n(\omega^{-1})$

d'où $(M_n(\omega))^{-1} = \dfrac{M_n(\omega^{-1})}{n}$

$$FFT^{-1}((v_0, \dots, v_{n-1}), \omega) = \frac{FFT((v_0, \dots, v_{n-1}), \omega^{-1})}{n}$$

"$v_k = A(x_k)$"

CCL Multiplier deux polynômes de degré d : $O(d \log d)$