IFT2015 hiver 2010 — Notes de cours

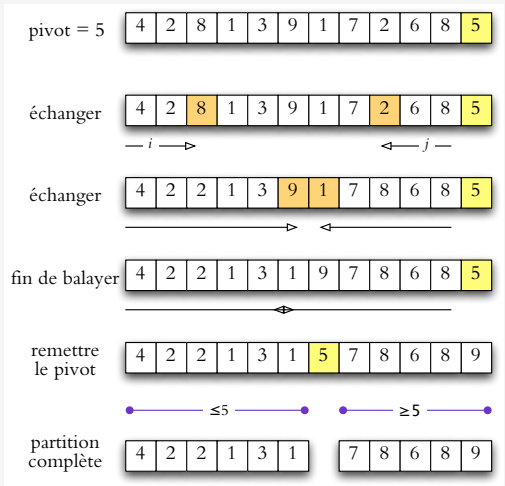
Miklós Csűrös

24 février 2010

10 Tri rapide

10.1 Algorithme

La récursion du **tri rapide** (*quicksort*) représente une logique complémentaire à celle du tri par fusion. De même façon, le tableau à l'entrée est divisé en deux et les deux moitiés sont triés par des appels récursifs. En tri par fusion, la division ne dépend pas du contenu du tableau — les deux moitiés sont combinés après les appels récursifs. En tri rapide, la division se fait en groupant les éléments dans les deux moitiés à l'aide d'un élément appellé le **pivot** x. On place les éléments inférieurs à x à la gauche, et ceux supérieurs à x à la droite du tableau. Avec x au milieu, on devra pas combiner les deux sous-tableaux après les avoir triés. Donc on fait la majeure partie du travail avant les appels récursifs, en une structure algorithmique de genre «combiner pour régner».



L'idée principale est la **partition** autour d'un pivot.

Algo QUICKSORT$(A[0..n-1], g, d)$ *// tri de $A[g..d]$*
Q1 **si** $g \geq d$ **alors retourner** *// cas de base*
Q2 $i \leftarrow$ PARTITION(A, g, d)
Q3 QUICKSORT$(A, g, i-1)$
Q4 QUICKSORT$(A, i+1, d)$

Algo PARTITION(A, g, d)
P1 chosir le pivot $p \leftarrow A[d]$
P2 $i \leftarrow g-1$; $j \leftarrow d$
P3 **boucler**
P4 **faire** $i \leftarrow i+1$ **tandis que** $A[i] < p$
P5 **faire** $j \leftarrow j-1$ **tandis que** $j \geq l$ et $A[j] > p$
P6 **si** $i \geq j$ **alors** sortir de la boucle
P7 échanger $A[i] \leftrightarrow A[j]$
P8 échanger $A[i] \leftrightarrow A[d]$
P9 **retourner** i

Pour trier un tableau $A[0..n-1]$ en ordre croissant, on exécute QUICKSORT$(A, 0, n-1)$. C'est un tri en place.

10.2 Performances

La partition (Lignes P3–P7) se fait en un temps $\Theta(n)$. Le temps de calcul est donc

$$T(n) = \Theta(n) + T(i) + T(n-1-i).$$

La récurrence dépend de l'indice i du pivot.

	pivot i	**récurrence** $T(n)$	**solution** $T(n)$
Meilleur cas	$(n-1)/2$	$2 \cdot T\big((n-1)/2\big) + \Theta(n)$	$\Theta(n \log n)$
Pire cas	$0, n-1$	$T(n-1) + \Theta(n)$	$\Theta(n^2)$
Moyen cas	aléatoire	$\mathbb{E}T(n) = 2\mathbb{E}T(i) + \Theta(n)$	$\Theta(n \log n)$

Le pire cas arrive (entre autres) quand on a un tableau trié au début !

Exercice 10.1. *On a aussi un temps de calcul quand la partition sépare au moins, disons un tiers des éléments. En général, supposons qu'il existe un $m \geq 2$ et un N tels que $\min\{i, n-1-i\} \geq n/m$ pour la partition de tous les sous-tableaux avec une longueur $n \geq N$. On a donc $T(n) \leq \Theta(n) + T\big((n-1)/m\big) + T\big((n-1)(m-1)/m\big)$ pour $n \geq N$. Démontrez que $T(n) = O(n \log n)$.*

10.3 Améliorations

Petits sous-tableaux. Il vaut mieux d'utiliser un tri par insertion quand $d - g$ est petit ($\leq 5..20$). En fait, on peut ignorer les petits sous-tableaux entièrement (tester $d < g + 10$ en Ligne Q1). Il faut juste faire un tri par insertion une fois à la fin.

Choix du pivot. Deux choix performent très bien en pratique : médiane ou aléatoire.

Médiane de trois

```
P1.1  si d ≥ g + 2 alors
P1.2      si A[g] > A[d − 1] alors échanger A[g] ↔ A[d − 1]
P1.3      si A[d] > A[d − 1] alors échanger A[d] ↔ A[d − 1]
P1.4      si A[g] > A[d] alors échanger A[g] ↔ A[d]
P1.5  p ← A[d]                          // A[g] ≤ A[d] ≤ A[d − 1]
```

et on se sert des **sentinelles** $A[g]$, $A[d-1]$:

```
P2'  i ← g; j ← d − 1
P5'      faire j ← j − 1 tandis que A[j] > p
```

Aléatoire

```
P1.1  k ← RANDOM(g, d)
P1.2  p ← A[k]
P1.3  si k ≠ d alors
P1.4      A[k] ← A[d]
P1.5      A[d] ← p
```

10.4 Moyen cas

Théorème 10.1. *Soit $D(n)$ le nombre moyen de comparaisons avec un pivot aléatoire, où n est le nombre d'éléments dans un tableau $A[0..n-1]$. Alors,*

$$\frac{D(n)}{n} = O(\log n).$$

Lemme 10.2. *On a $D(0) = D(1) = 0$, et*

$$D(n) = n - 1 + \frac{1}{n}\sum_{i=0}^{n-1}\Big(D(i) + D(n-1-i)\Big) = n - 1 + \frac{2}{n}\sum_{i=0}^{n-1} D(i).$$

Démonstration. Supposons que le pivot est le i-ème plus grand élément de A. Le pivot est comparé à $(n-1)$ autre éléments pour la partition. Les deux partitions sont de tailles i et $(n-1-i)$. Or, i prend les valeurs $0, 1, \ldots, n-1$ avec la même probabilité. ■

Preuve de Théorème 10.1. Par Lemme 10.2,

$$\begin{aligned} nD(n) - (n-1)D(n-1) &= \Big(n(n-1) + 2\sum_{i=0}^{n-1} D(i)\Big) - \Big((n-1)(n-2) + 2\sum_{i=0}^{n-2} D(i)\Big) \\ &= 2(n-1) + 2D(n-1). \end{aligned}$$

D'où on a

$$\frac{D(n)}{n+1} = \frac{D(n-1)}{n} + \frac{2n-2}{n(n+1)} = \frac{D(n-1)}{n} + \frac{4}{n+1} - \frac{2}{n}.$$

Avec $E(n) = \frac{D(n)-2}{n+1}$, on peut écrire

$$E(n) = E(n-1) + \frac{2}{n+1}.$$

Donc,

$$\begin{aligned} E(n) &= E(0) + \frac{2}{2} + \frac{2}{3} + \cdots + \frac{2}{n+1} \\ &= \frac{D(0)-2}{1} + 2(H_{n+1} - 1) = 2H_{n+1} - 4 \end{aligned}$$

où $H_n = \sum_{i=1}^{n} 1/i$ est le n-ième nombre harmonique.

En retournant à $D(n) = 2 + (n+1)E(n)$, on a alors

$$D(n) = 2(n+1)H_{n+1} - 4n - 2 < 2nH_{n+1}$$

Donc le nombre de comparaisons en moyenne est tel que

$$\frac{D(n)}{n} < 2H_{n+1} = O(\log n).$$

■

En fait la preuve montre que $D(n)/n = \big(2 + o(1)\big)H_n \sim 2\ln n \approx 1.39 \lg n$. C'est seulement 39% pire que le meilleur cas !

10.5 Sélection

Supposons qu'on veut trouver le k-ème plus petit élément dans un tableau $A[0..n-1]$. Il existe des algorithmes qui le font en temps $\Theta(n)$ au pire cas. En pratique, on peut se servir de la partition aussi pour arriver à $\Theta(n)$ en moyen cas. Idée de clé : après avoir appellé $i \leftarrow \text{PARTITION}(A, 0, n-1)$, on trouve le k-ème élément en $A[0..i-1]$ si $k < i$ ou en $A[i+1..n-1]$ si $k > i$. En même temps, on réorganise le tableau pour que $A[k]$ soit le k-ème plus petit élément.

```
   Algo SELECTION(A[0..n−1], g, d, k)
S1 si d ≤ g + 1 alors                                  // cas de base : 1 ou 2 éléments
S2     si d = g + 1 et A[d] < A[g] alors échanger A[g] ↔ A[d]   // 2 éléments
S3     retourner A[k]
S4 i ← PARTITION(A, g, d)
S5 si k = i alors retourner A[k]                       // on l'a trouvé
S6 si k < i alors retourner SELECTION(A, g, i − 1, k)  // continuer à la gauche
S7 si k > i alors retourner SELECTION(A, i + 1, d, k)  // continuer à la droite
```

Comme c'est une **récursion terminale**, on peut transformer le code en forme itérative très facilement.

Théorème 10.3. *Avec un pivot aléatoire, algorithme* SELECTION *fait* $\big(2 + o(1)\big)n$ *comparaisons en moyenne.*

Exercice 10.2. *Démontrer Théorème 10.3.*