

Analyse comparée du Tri-Fusion et du Tri-Rapide

Ces deux tris utilisent "diviser pour régner" mais de deux façons différentes. Il est donc intéressant de comparer leur complexité.

Complexité du tri fusion

- TRI-FUSION(T, p, n):
si $p < n$ alors
 $q = \lfloor \frac{p+n}{2} \rfloor$
 $\left[\begin{array}{l} \text{TRI-FUSION}(T, p, q) \\ \text{TRI-FUSION}(T, q+1, n) \\ \text{FUSION}(T, p, q, n) \end{array} \right.$

Sur un tableau de taille n FUSION (cf. plus) fait toujours exactement n comparaisons. Donc on a :

$$T(n) = T(\lfloor \frac{n}{2} \rfloor) + T(\lceil \frac{n}{2} \rceil) + n, \quad T(0) = T(1) = 0$$

- Pour simplifier la résolution de cette récurrence on admet que $T(n)$ est croissante et on calcule quand n est une puissance de 2.

$$T(2^{k+1}) = 2T(2^k) + 2^{k+1}$$

$$\text{d'où } \frac{T(2^{k+1})}{2^{k+1}} = \frac{T(2^k)}{2^k} + 1$$

$$\text{donc par réc : } \frac{T(2^k)}{2^k} = k \quad \text{d'où } T(n) = n \log n \quad (n=2^k)$$

Donc dans tous les cas $T(n) = \Theta(n \log n)$ (par croissance).

Rq: Il existe un théorème général qui donne la solution à ce genre de récurrences.

Complexité du Tri Rapide

TRI-RAPIDE(T, p, n):

si $p < n$ alors

$\left\{ \begin{array}{l} q \leftarrow \text{PARTITION}(T, p, n) \\ \text{TRI-RAPIDE}(T, p, q-1) \\ \text{TRI-RAPIDE}(T, q+1, n) \end{array} \right.$

PARTITION fait exactement $n-1$ comparaisons (cf. plan).

Cas le plus défavorable

↳ Quand à chaque exécution de PARTITION un des deux tableaux obtenus est vide (ie. pivot = min ou max).

$$\text{Alors } C_{\text{pire}}(n+1) = n-1 + C_{\text{pire}}(n)$$

$$\text{d'où } \underline{C_{\text{pire}}(n) = \mathcal{O}(n^2)}.$$

↳ Mais les cas pathologiques sont peu nombreux... d'où le cas moyen en $\mathcal{O}(n \log n)$!

Cas moyen

Quand le pivot est en j ème position les deux sous-problèmes à traiter sont de taille $j-1$ et $n-j$.

Par hypothèse (cas moyen) toutes les permutations de l'entrée sont équiprobables donc le pivot a pour probabilité $1/n$ d'être en j ème position après l'exécution de partition.

On a donc:

$$C_{\text{moy}}(n) = n-1 + \frac{1}{n} \sum_{j=1}^n (C(j-1) + C(n-j))$$

$$= n-1 + \frac{2}{n} \sum_{j=1}^{n-1} C(j)$$

$$\text{car } \underline{C(0) = C(1) = 0}$$

On fait un peu de calcul:

$$n(C(n) - n + 1) = 2 \sum_{j=1}^{n-1} C(j)$$

$$(n-1)(C(n-1) - n + 2) = 2 \sum_{j=1}^{n-2} C(j)$$

On soustrait et on obtient:

$$nC(n) - (n+1)C(n-1) = 2(n-1)$$

$$\text{d'où } \frac{C(n)}{n+1} - \frac{C(n-1)}{n} = \frac{2(n-1)}{n(n+1)}$$

$$\text{d'où } \frac{C(n)}{n+1} \leq \frac{C(n-1)}{n} + \frac{2}{n+1}$$

Donc par réc : $C(n) \leq (n+1) \sum_{k=1}^n \frac{2}{k+1}$

$$= 2(n+1)(H_{n+1} - 1)$$

$$\sim_{n \rightarrow \infty} 2n \ln n$$

$$\sim 1,38 n \log n$$

d'où $C_{\text{moy}}(n) = \Theta(n \log n)$.

Bilan

- Le tri fusion fait moins de comparaisons que le tri rapide ($n \log n$ vs $1,38 n \log n$ en moy). Mais il est cependant plus lent, notamment car il n'est pas en place : c'est long de recopier tous les tableaux (ce développement aussi d'ailleurs). C'est donc pour cela que l'on utilise le tri rapide en pratique !

Tri par tas mouc

Référence Fenderson, Gaudel, Loria p 343

• Définition :

Un tas mouc est un arbre binaire A vérifiant

- A est parfait
- Tout noeud de A a une étiquette supérieure à celles de ses fils

On va utiliser deux opérations sur ces tas, l'ajout d'un élément à un tas et la suppression du maximum

• Lemme préliminaire : la hauteur h d'un arbre parfait de taille m vérifie : $\log_2(m+1) - 1 \leq h < \log_2(m+1)$

Preuve : Un arbre parfait de hauteur h est un arbre complet de hauteur $h-1$ auquel on a ajouté des feuilles de profondeur h , placées de gauche à droite. En particulier :

$$2^{h-1} - 1 \leq m \leq 2^h - 1$$

nombre de noeuds d'un arbre complet de hauteur $h-1$

On en déduit $h < \log_2(m+1) \leq h+1$
 d'où $\log_2(m+1) - 1 \leq h < \log_2(m+1)$

• Représentation et accès du tas

On représente notre liste d'éléments à trier par un tableau de taille $m+1$ $[1..m]$.

Le tableau sera ensuite en cours d'algorithme à moment donné notre structure de base, avec les conventions suivantes :

- $A[1]$ est la racine
- $\forall i, \text{ pour } i > 1, A[i]$ est un élément du tas, ainsi $A[\lfloor \frac{i}{2} \rfloor]$ est son père
- $A[2i]$ et $A[2i+1]$ sont, s'ils existent, les fils de $A[i]$
- $\exists p$ le nombre de nœuds du tas est pair, alors $A[\frac{p}{2}]$ a uniquement fils $A[1]$
- $\forall i > \lfloor \frac{p}{2} \rfloor, A[i]$ est une feuille

Le principe du tri va être de construire un tas à partir des éléments à trier, puis d'extraire le maximum des tas successifs pour obtenir la liste triée.

Algorithmes du tri par tas

• Ajouter d'un élément à un tas

- État initial : $A[1, \dots, m]$ un tableau et $p < m$ tq $A[1 \dots p]$ représente un tas et $A[p+1]$ l'élément à ajouter

Ajouter (A, p)

$i \leftarrow p+1$

tant que $i > 1$ et $A[i] > A[\lfloor \frac{i}{2} \rfloor]$ faire

$A[i] \leftrightarrow A[\lfloor \frac{i}{2} \rfloor]$

$i \leftarrow \lfloor \frac{i}{2} \rfloor$

fin tant

fin

- État final : $A[1 \dots p+1]$ représente un tas

Terminaison: pour $i > 0$ $E(\frac{i}{2}) < i$,

Correction: On ajoute le nouvel élément sur une feuille et on le remonte le long de la branche si besoin est, ce si il est supérieur à son père dans ce cas, il est aussi supérieur à son éventuel frère, et la structure de tas est vérifiée localement après l'échange

Complexité: Dans le pire cas, on compare le nouvel élément à tous les éléments d'une branche, ce qui nous donne $O(\log_2 p) = O(\log_2 n)$ comparaisons, d'après le lemme. De même pour les échanges

Suppression du maximum du tas

Etat initial: $A[1..m]$ un tableau et $p \leq m$ tel que $A[1..p]$ représente un tas

Supprimer $\max(A, p)$

$A[1] \leftrightarrow A[p]$

$i \leftarrow 1$

tant que $i \leq E(\frac{p-1}{2})$ faire

si $2i = p-1$ ou $A[2i] > A[2i+1]$ faire

$j \leftarrow 2i$

sinon

$j \leftarrow 2i+1$

fin si

si $A[i] < A[j]$

$A[i] \leftrightarrow A[j]$

$i \leftarrow j$

sinon

exit

fin tant que

• Etat final: $A[1, \dots, n-1]$ est un tas ne contenant pas la racine maximum qui sera trouvée en $A[n]$.

• Terminaison: $\forall i \geq 1, 2i > i$, et on finit par l'opération $E\left(\frac{n-1}{2}\right)$.

• Correction: On place l'élément $A[n]$ à la racine puis, on l'échange avec le plus grand de ses fils (lequel est obtenu simplement à son père) tant qu'il y a des fils, on conserve la structure de tas.

• Complexité: Dans le pire cas, on compare le élément racine à la racine avec tous les éléments d'une branche, ce qui nous donne $O(\log n) = O(\log m)$ comparaisons.

Algorithme final:

• Etat initial: A un tableau à n éléments

Trier par tas (A):

pour i de 1 à $n-1$ faire

 Ajouter(A, i)

fin pour

pour p de n à 2 faire

 Supprimer_max(A, p)

fin pour

• Etat final: A est trié.

• Terminaison : trivial

• Correction :

· A la fin de la première boucle $k[1..m]$ est un tas

· A la i^{e} itération de la seconde boucle, le maximum de $k[1..m-i]$ est placé en $k[m-i+1]$

Le tableau est donc trié

• Complexité :

Chaque boucle fait intervenir une ^{fonction de} complexité en $O(\log_2 m)$.

La complexité dans le cas le pire est donc en $O(m \log_2 m)$

Conclusion :

Le tri par tas est donc un tri stable de complexité en $O(m \log_2 m)$ dans le cas pire et en moyenne (car le théorème d'optimalité des tris comparatifs impose une meilleure complexité moyenne)