Leçon 903 : Algorithmes de tri : Exemples

07/04 2015

Introduction :

Trier, c'est organiser un ensemble de données suivant un ordre prédéfini.

L'intérêt de leur étude est multiple : améliorer le rendement d'un programme en triant préalablement l'entrée, application au domaine graphique (algorithme du peintre)

Parmi les propriétés les plus recherchées, on peut parler de tri

- en place, s'il modifie directement la structure à trier, sans necessiter de variables supplementaires (ou très peu)
- stable, si les éléments égaux pour ce tri sont dans le même ordre à l'entrée et à la sortie

Parmi les tris que nous étudierons, on remarquera d'abord les tris par comparaison, avant de finir sur d'autres méthodes

I. Tris quadratiques.

1) Tri par insertion

Le tri par insertion est, parmi les algorithmes de tri, un des premiers auquel on pense. On s'en sert notamment pour trier nos jeux de cartes, entre autres.

~~Def~~ Il se présente comme suit

Def 1 : Le tri par insertion

```
TRI-INSERTION(A).
| Pour j entre 2 et longueur(A).
|  | x = A[j]
|  | i = j-1
|  | tant que i > 0 et A[i] > x
|  |  | A[i+1] = A[i]
|  |  | i = i+1
|  | A[i+1] = x.
| Retourner A.
```

Pté 2 : Le tri par insertion est un tri en place, et un tri stable.

Pté 3

Le tri par insertion possède une complexité en moyenne, et une complexité dans le pire cas, en $O(n^2)$. Dans le meilleur cas, la complexité est en $O(n)$.

(La complexité est ici calculée en comptant les comparaisons dans l'algorithme.)

On a donc une complexité en $O(n^2)$, ce qui est moins intéressant que les algorithmes que nous verrons ultérieurement. Toutefois, le tri par insertion se révèle plus efficace, sur de petites entrées, ou des entrées déjà presque triées que les algorithmes basés sur la méthode "diviser pour régner."

C'est pour cela qu'il est parfois utilisé en combinaison avec d'autres algorithme de tri.

② Autres exemples de tris quadratiques.

Le tri par insertion n'est pas le seul tri quadratique, on peut également citer :

- le tri par sélection, dans lequel on recherche à la fois le plus petit élément d'un sous-tableau de l'entrée, pour le mettre au début
- le tri bulle, dans lequel on fait remonter les plus grands éléments d'une liste, en les comparant avec leurs successeurs.

Ces algorithmes ont eux aussi une complexité en $O(n^2)$, contrairement aux tris par comparaisons que nous allons étudier désormais.

II Tris dichotomiques

Ces méthodes de tri sont issues de la méthode "diviser pour régner", et s'ils sont aussi des tris par comparaison, ils présentent de meilleures complexités que les tris quadratiques.

① Le tri fusion

Le tri fusion se base sur 2 algorithmes pour marcher.

Def 4:
```
Tri-fusion (A,p,r)
si p<r
    q = ⌊(p+r)/2⌋
    Tri-fusion (A,p,q)
    Tri-fusion (A,q,r)
    Fusion(A,p,q,r).
```

L'algorithme FUSION, lui, fusionne deux listes, en cherchant, à chaque itération, le plus petit élément dans les 2 listes, et en l'ajoutant à une nouvelle liste.

Prés 5 : Le tri-fusion est un tri stable, mais pas un tri en place.

En effet, dans l'algorithme Fusion, on recrée une liste pour en fusionner 2.

Pté 6 : Le tri fusion possède une complexité en $O(n \log n)$ en moyenne.

Cette complexité est bien meilleure que celle des tris quadratiques, mais peut-on, avec des tris par comparaison, faire mieux que cela ?

Avantages du tri fusion : Plus rapide, stable

Inconvénient : Algorithme qui demande beaucoup d'espace, non en place.

② Autres tris dichotomiques

Le tri par tas est un autre tri de complexité $O(n \log n)$, qui a l'avantage d'être un tri en place.

Il consiste à voir le tableau en entrée comme un arbre binaire avec le premier élément en racine consiste à faire remonter le plus gros élément du tableau à la racine, pour ensuite le retirer pour le remplacer par un autre élément, et ainsi de suite

S'il reste un tri en $O(n\log n)$, il reste malgré tout plus lent qu'un autre tri en $O(n\log n)$.

Le tri rapide, qui consiste à placer un pivot à sa bonne position, puis à permuter tous les éléments autour de lui, suivant leur place relative.

Clé 7 : Le tri rapide a une complexité au pire en $O(n^2)$, et une complexité moyenne en $O(n \log n)$ [DuPpt 1].

$\boxed{\text{Thm 8}}$: La meilleure complexité que l'on puisse avoir, pour un tri par comparaison, est $O(n \log n)$ [DuPpt 2].

Sachant donc cette limite atteinte, existe-t-il d'autres méthodes de tri qui offre une meilleure complexité.

III : D'autres Méthodes de tri

1) Le tri par dénombrement

On suppose ici que les éléments de l'entrée sont des entiers de l'intervalle $[0,k]$, avec $k\in\mathbb{N}$.

Si $k = O(n)$, l'algorithme s'exécute en temps $\Theta(n)$.

Le principe du tri par dénombrement consiste à déterminer, pour chaque élément x de l'entrée, le nombre d'éléments inférieurs à x. On peut alors directement placer x au bon endroit.

Tri. Dénombrement s'exécute en 3 temps., pour i entier :

(a) Remplir un tableau avec, en $i^{\text{ème}}$ position, le nombre d'éléments égaux à i.

(b) Remplacer les valeurs de ce tableau par le nombre d'éléments inférieurs ou égaux à i

(c) Placer l'élément i à sa position exacte dans le tableau final.

On a donc un algorithme ayant une complexité en $\Theta(n)$, ~~et~~ qui demande un espace mémoire en $O(n)$.

2) Autres tris

Parmi les autres tri dont nous aurions pu parler, on pourra citer

- le tri par base
- le tri par paquet
- les réseaux de tri [Cor ; ancienne version] (hors-programme).

Référence : Cormen - Leiserson - Rivest - Stein : Algorithmique, 2e et 3e éditions [Cormen]

Froidevaux - Gaudel - Soria . Types de données et algorithmes : [FGS]

Annexe: Algorithmes du tri rapide

```
TRI-RAPIDE(A,p,r)
  si p<r
    q ← PARTITION(A,p,r)
    TRI-RAPIDE(A,p,q-1)
    TRI-RAPIDE(A,q+1,r)

PARTITION(A,p,r).
  x ← A[r]
  i ← p-1
  pour j entre p et r-1
    si A[j] ≤ x
      i ← i+1
      permuter A[i] et A[j].
  permuter A[i+1] et A[r]
  retourner i+1
```

Complexité du Quicksort.

(I) Complexité dans le pire des cas.

Etudions tout d'abord le cas du partitionnement.

Le cas le plus defavorable serait de produire un sous problème à $(n-1)$ éle

0 éléments.

Si on suppose que ce cas de figure arrive tout le temps, on a alors, sach

partitionnement coûte $\Theta(n)$

$$T(n) = T(n-1) + \Theta(n), \text{ avec } T(n) \text{ le nombre de comparaisons effectuées.}$$

ce qui donne $T(n) = \Theta(n^2)$

Plus precisement,

Rec : $$T(n) = \max_{0 \leq q \leq n-1} \left(T(q) + T(n-1-q) \right) + \Theta(n)$$

or $T(n) \leq cn^2$, pour un certain c constant, d'où

$$T(n) \leq \max\left(cq^2 + §c(n-q-1)^2\right) + \Theta(n)$$

$$T(n) \leq C \max_{0 \leq q \leq n-1} \left(q^2 + (n-q-1)^2\right) + \Theta(n)$$

l'expression $q^2 + (n-q-1)^2$ atteint ses extrema aux extrem

d'où $q^2 + (n-1-q)^2 \leq (n-1)^2 = n^2 - 2n + 1$.

d'où $T(n) \leq cn^2 - c(2n-1) + \Theta(n)$

$T(n) \leq cn^2$

on peut choisir c suffisamment grand pour que

$\Theta(n)$

donc. $T(n) = O(n^2)$

(II) Complexité moyenne

Pour simplifier les notations, on appellera $z_1, \dots, z_n$ les elements du tableau, et z_{ij}

Nous allons donc compter le nombre de comparaisons effectuées dans l'execution

D...

$$X = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}$$

et alors $\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{E}[X_{ij}]$. par linéarité de l'espérance

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mathbb{P}\{z_i \text{ et } z_j \text{ sont comparés dans } Z_{ij}\}.$$

Or z_i et z_j seront comparés, dans Z_{ij}, ~~s~~uniquement si l'un de 2 est le premier pivot choisi, sinon, ils seront séparé dans 2 sous tableaux et ne seront plus comparés par la suite.

D'où $\mathbb{P}\{z_i \text{ et } z_j \text{ sont comparé dans } Z_{ij}\} = \mathbb{P}\{z_i \text{ est le premier pivot ou } z_j \text{ est le } 1^{er} \text{ pivot}\}$

les 2 evenements étant independants, $\mathbb{P}\{z_i \text{ et } z_j \text{ sont comparé}\} = \mathbb{P}\{z_i = 1^{er} \text{ pivot}\} + \mathbb{P}\{z_j = 1^{er} \text{ pivot}\}$.

les pivots étant choisis aleatoirement, avec distribution uniforme, ces probabilités valent $\frac{1}{j+1-i}$

d'où $\mathbb{P}\{z_i \text{ et } z_j \text{ sont comparé}\} = \frac{2}{j-i+1}$

d'où $\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j+1-i}$

Par changement de variable, on a. $(k = j-i)$.

$$\mathbb{E}[X] = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \frac{2}{j+1-i} = \sum_{i=1}^{n-1} \sum_{k=1}^{n-i} \frac{2}{k+1} < \sum_{i=1}^{n-1} \sum_{k=1}^{n} \frac{2}{k}$$

$$\mathbb{E}[X] < \sum_{i=1}^{n-1} O(\log n) = O(n \log n).$$

On arrive bien au temps attendu de $O(n \log n)$.

Reference: [Cormen]

Dans le pire cas? Sous la complexité quadratique

↳ à chaque fois, on choisit le ~~pivot~~ maximum ou le minimum.

$$\mathbb{P}(c = O(n^2)) = \frac{2}{n}\, \mathbb{P}(c = O((n-1)^2)) = \frac{2^{n-1}}{n!}$$

Optimalité du tri par comparaison

Rappel: La meilleure complexité, pour un tri par comparaison, est $O(n \log n)$.

On parle ici de comparaison au pire et en moyenne, et on compte les comp

Pour établir cette borne inférieure, on va utiliser les arbres binaires de décisions

<u>Lemme 1</u>: L'arbre binaire de décision associé à un algo de tri par compara clés a exactement $n!$ feuilles.

<u>Preuve</u>: Chaque feuille indique le résultat des différentes exécutions de toutes les données possibles.

le nombre d'ordres possibles pour les n éléments étant $n!$, on a le annoncé.

Notons $TC_{max}(n)$ le nombre minimal de comparaison, (et $TC_{moy}(n)$ le nombre moyen à effectuer dans le pire cas (resp. en moyenne).

On a alors $TC_{max}(n) = \inf\{h(A), A$ arbre de décision associé à l'algo de tri

$TC_{moy}(n) = \inf\{PE(A), A$ arbre de décision associé à l'algo de tri

avec $PE(A)$ profondeur moyenne externe de A.

<u>Lemme 2</u>: Tout arbre binaire de décision ayant n feuilles a une haut égale à $\lceil \log_2 n \rceil$

On en déduit que $TC_{max}(n) \geq \lceil \log_2 (n!) \rceil$

De plus $n! \; \lceil \ldots \; (n)^n / \ldots$

... $\lceil \log_2(n!) \rceil = \Theta(n \log n)$.

Et on connaît des algos de tri dont la complexité au pire est $\Theta(n \log n)$.

Exemple le tri fusion, pour lequel le nombre de comparaisons pour trier n éléments vérifie la relation (sachant que, dans le pire des cas, il y a $n-1$ comparaisons).

$$F(n) = n-1 + F(\lfloor n/2 \rfloor) + F(\lfloor n/2 \rfloor)$$

et alors $F(n) = n \log_2(n) - 2^{\lceil \log_2 n \rceil} + 1$.

Pour le cas de la complexité moyenne,

on a $TC_{moy}(n) = \inf(\{PE(A)\})$.

Lemme 3 : Pour un arbre binaire A ayant n feuilles, $PE(A) \geq \log_2(n)$.

Ce lemme nous donne alors directement $TC_{moy}(n) \geq \log_2(n!)$, et donc

$$TC_{moy}(n) = \Theta(n \log n)$$

Pour l'optimalité, on citera le tri rapide, dont on sait qu'il atteint cette borne

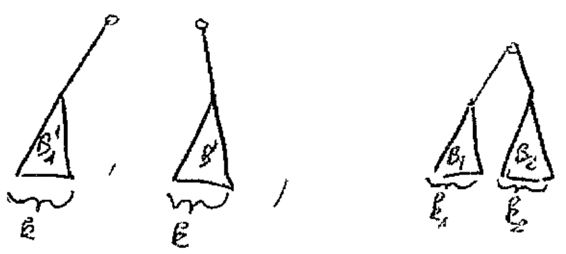
/// Démo du lemme 3 : raisonnement par récurrence :

Initialisation : c'est vrai pour l'arbre à une feuille, qui est de profondeur 0.

Supposons la propriété vraie pour les arbres ayant moins de $k-1$ feuilles

Soit B un arbre à k feuilles

B est d'une de ces 3 formes



Le seul cas à montrer ici est le 3e cas.

Comme $k_1 < k$ et $k_2 < k_1$ B_1 et B_2 vérifient la propriété par récurrence, donc

$$PE(B_1) \geq \log_2 k_1 \quad \text{et} \quad PE(B_2) \geq \log_2 k_2.$$

Or $PE(B) = 1 + \dfrac{k_1}{k_1+k_2} PE(B_1) + \dfrac{k_2}{k_1+k_2} PE(B_2)$.

d'où $PE(B) = 1 + \dfrac{k_1 \log_2 k_1 + k_2 \log_2 k_2}{k_1 + k_2}$.

Comme $k_1 + k_2 = k$, on peut écrire

$$PE(B) = 1 + \frac{k_1 \log_2 k_1 + (k-k_1)\log_2(k-k_1)}{k} = q(k_1).$$

et $q(k_1)$ est minimal quand $k_1 = \frac{k}{2}$), et alors $q(k/2) \geq \log_2(k$

Ce qui achève la récurrence.

Référence: [FGS].