15/02/2023

Les algorithmes de tri

Études des algorithmes et leurs complexités

LAZAAR El Mahdi
UNIVERSITE LE HAVRE NORMANDIE

LAZAAR El Mahdi
Université Le Havre Normandie
Année : 2022/2023

Les algorithmes de tri

A) Etudes des algorithmes du tri (tri à bulles, tri rapide, tri par dénombrement)

1.1 Algorithme de tri à bulles

La méthode d'algorithme de tri à bulles consiste à comparer les éléments d'un tableau deux à deux (le premier élément avec le deuxième, le deuxième avec le troisième ainsi de suite). Si les deux éléments comparés ne sont pas en ordre, on les inverse. On se décale d'une case pour recommencer la comparaison et l'élément maximum se place à la position (n-1) (en considérant l'hypothèse que le tableau est de taille n et le premier indice du tableau et 0).

Le deuxième passage sur le tableau se fait en le privant de son dernier élément, puis l'avant dernier ainsi de suite jusqu'à n'avoir qu'un seul élément qui sera le minimum du tableau. On aura au final un tableau trié.

Exemple d'algorithme

En entrée : tab (tableau a trié), N (taille du tableau).

En sortie : Le tableau tab trié.

Variables : echange (booléen), i,j (compteurs), dernierEchange (indice du dernier échange), tmp (variable d'échange)

Début

```
    Echange <- vrai
    i <- 0
    Tant que (i < N ET echange == vrai)
        echange = faux
        Pour j de 0 à (N – i – 1)
            Si (tab [j] > tab [j + 1])
                // on fait l'echange
                Alors tmp = tab [j]
                tab[j] = tab [j + 1]
                tab [j + 1] = tmp
                echange = vrai
                dernierEchange = j
            Fin si
        Fin pour
```

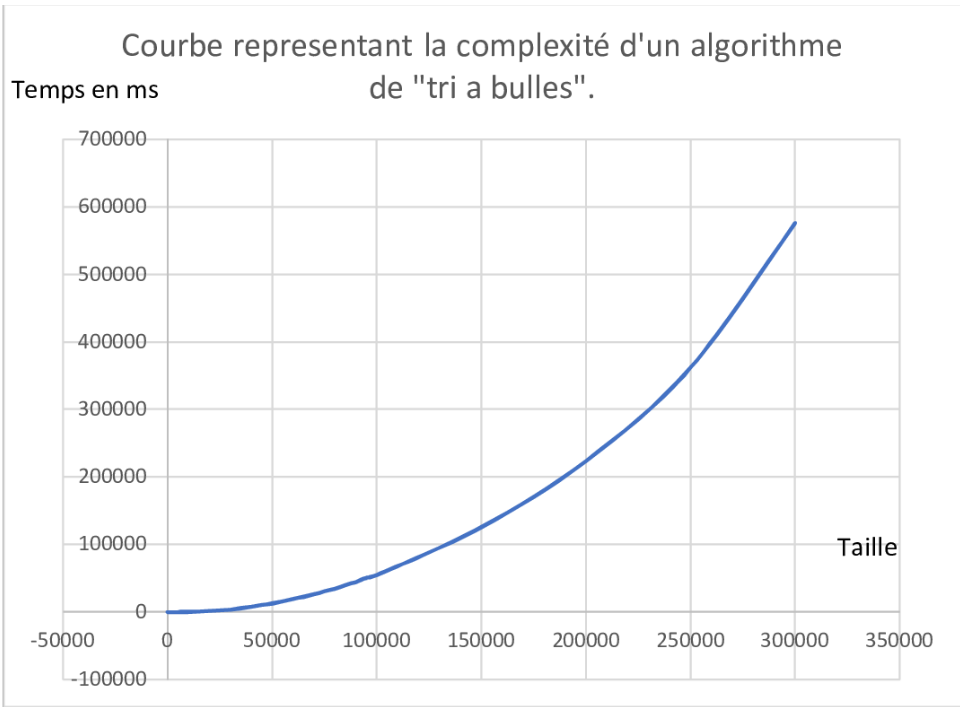
i = i + 1

Fin Tant que

Fin

1.2 Etude de complexité de l'algorithme tri à bulles

La méthode de tri à bulles est un algorithme qui compare les éléments d'un tableau deux à deux et s'ils ne sont pas en ordre il les inverses. On suppose qu'on a un tableau de n éléments, au pire des cas ce tri effectue (n) comparaisons pour placer le premier plus grand élément à sa position finale puis (n – 1) comparaisons pour le deuxième plus grand élément, puis (n-2) comparaisons ainsi de suite jusqu'à que le tableau soit entièrement trié. De ce fait la somme de [n + (n-1) + (n-2) + (n-3) +…+1] égale à n(n+1) /2, alors on déduit que la complexité au pire du tri à bulles est de $O(n^2)$, ce qui est démontré par la courbe suivante qui est obtenue en récupérant le temps d'exécution en ms pour chaque taille de 10 à 300 000.



Taille	Temps en ms
10	0
100	1
1000	9
5000	62
10,000	285
15,000	704
20,000	1400
25,000	2158
30,000	3626
35,000	5447
40,000	7540
45,000	10299
50,000	12638
55,000	15686
60,000	19186
65,000	22318
70,000	26015
75,000	30366
80,000	34567
85,000	39619
90,000	44212
95,000	50587
100,000	54741
150,000	125612
200,000	223457
250,000	361628
300,000	575473

On voit clairement que la courbe de l'algorithme de tri à bulles prend la forme d'une parabole ce qui confirme ce que nous avons conclus auparavant, nous remarquons aussi que le temps d'exécution de l'algorithme de 15 000 à 20 000 est à peu près multiplié par 2.

2.1 Algorithme du tri rapide

L'algorithme du tri rapide utilise la méthode : diviser pour régner (**Divide and Conquer rule**), c'est un tri qui se fait d'une manière récursive, il utilise un élément du tableau comme pivot qui sera utile pour mettre les éléments du tableau qui lui sont inférieurs à gauche et le reste à droite et pour le deuxième appel il choisit un autre pivot qui sera la référence (le nouveau pivot) pour trier les deux tableaux qu'on a obtenus après le premier passage (c-à-d le tableau de gauche qui commence de début jusqu'à (pivot – 1) et celui de droite de (pivot + 1) jusqu'à la fin du tableau).

Après le choix du pivot on utilisera deux variables (des entiers) qui serviront pour parcourir les deux extrémités du tableau et voir si chaque élément est bien placé par rapport au pivot si non on fait un échange pour que tous se met en ordre. Par exemple si on trouve un élément dans la partie gauche supérieur au pivot et un autre dans la partie droite inferieur à ce dernier on fait l'échange entre ces deux et nos indices continue leurs parcours du tableau jusqu'à ce qu'ils soient égaux ou ils se croisent. À la fin on retourne l'indice du pivot.

Dans notre algorithme le pivot sera toujours le premier élément du tableau.

Exemple d'algorithme

En entrée : tab (le tableau à trié), debut et fin (le premier et le dernier indice du tableau)

En sortie : le tableau tab trié.

Variables : g,d (compteurs), pivot et indicePivot (des entiers).

//La fonction partition qui gère les tableaux

```
Debut
    g <-debut + 1
    d <- fin
    Pivot <- tab[debut]
    IndicePivot <- 0
    Tant que g < d
        Tant que g < d ET tab[g] < pivot
            g++
        Fin Tant que
        Tant que d > g ET tab[d] > pivot
            d--
        Fin Tant que
        // on appelle la méthode echanger
        echanger(tab, g, d)
        Si g < d
            g++
            d--
        Fin si
    Fin Tant que
    Si g == d ET tab[g] <= pivot
        IndicePivot <- g
    Sinon
        IndicePivot <- g – 1
    Fin si
    echanger(tab, debut, IndicePivot)
```

Retourner IndicePivot

Fin

La fonction tri Rapide qui appel la méthode partition :

En entrée : tab (le tableau à trié), debut et fin (le debut et la fin du tableau)

En sortie : tab (tableau trié)

Variables : pivot (référence pour trier le tableau) (entier)

Debut :

Si debut < fin

pivot <- partition (tab, debut, fin)

triRapide (tab, debut, pivot – 1)

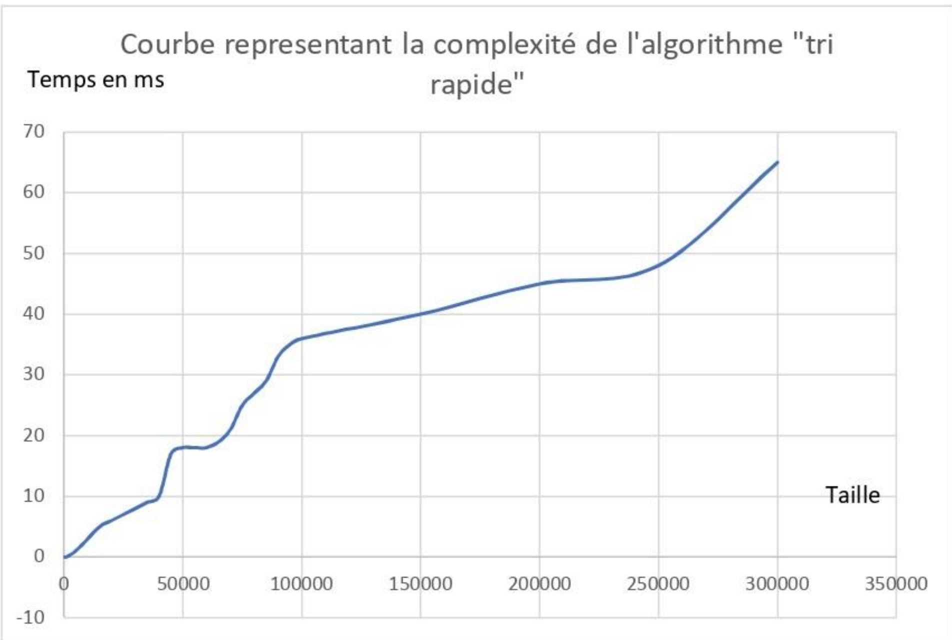
triRapide (tab, pivot + 1, fin)

Fin si

Fin

2.2 Etude de complexité de l'algorithme tri rapide

L'algorithme de tri rapide est un algorithme récursif qui prend en entrée le tableau à trié de taille n, l'indice de début et l'indice de la fin du tableau, cette méthode de tri consiste à diviser le tableau en deux à chaque appel (d'une manière simple il divise un problème en petits sous problèmes pour mieux le résoudre) pour réduire sa taille et le trie devient de plus en plus facile (il utilise le principe de diviser pour conquérir). Ce qui implique une complexité de $O(n\log(n))$. Ce qui est démontré par la courbe suivant :



Taille	Temps en ms
100	0
1000	0
5000	1
10,000	3
15,000	5
20,000	6
25,000	7
30,000	8
35,000	9
40,000	10
45,000	17
50,000	18
55,000	18
60,000	18
65,000	19
70,000	21
75,000	25
80,000	27
85,000	29
90,000	33
95,000	35
100,000	36
150,000	40
200,000	45
250,000	48
300,000	65

En interprétant la courbe, on regarde quelle prend approximativement la forme d'une courbe de $n \log(n)$ ce qui confirme notre hypothèse.

3.1 Algorithme du tri par dénombrement

Le tri par dénombrement est un algorithme qui ne se base pas sur les comparaisons entre les éléments mais sur leurs nombres d'occurrences dans le tableau.

Dans un premier temps la seule comparaison qu'aura lieu c'est la recherche du maximum pour qu'il servira de taille + 1 pour le tableau intermédiaire, après nous allons incrémenter chaque case du tableau intermédiaire de 1 en utilisant comme indice l'élément du tableau à trié.

La troisième étape sera la détermination du rang finale de chaque élément cela se fait en calculant le nombre d'occurrences de toutes les valeurs plus petites que la valeur en question.

La quatrième étape est de créer un tableau de résultat où nous allons utiliser le rang final que nous avons calculer pour placer chaque élément dans sa position, ensuite on décrémente la case en cours, par exemple si on a 4 dans la case d'indice 10 c-à-d qu'on a 4 fois le nombre 10 et lorsque nous allons faire le premier passage on doit décrémenter cette case par 1 ce qui fait qu'il restera que trois 10 à placés.

La dernière étape consiste à parcourir le tableau de résultat et affecter chaque élément au tableau initial, ce qui donnera un tableau trié.

Exemple d'algorithme

En entrée : tab (le tableau à trié), N (la taille du tableau)

En sortie : tab (tableau trié)

Variables : tabInter (le tableau intermédiaire), tabRes (tableau de résultat), max (la valeur maximale du tableau tab), tailleC (la taille du tableau intermédiaire)

```
Debut
    max <- tab [0]
    Pour i de 1 à N
        Si tab[i] > max
            max <- tab[i]
        Fin si
    Fin pour
    tailleC <- max + 1
    Pour i de 0 à tailleC
        tabInter[i] <- 0
    Fin pour
    Pour i de 0 à N
        tabInter[tab[i]] ++
    Fin pour
    Pour i de 1 à tailleC
        tabInter[i] <- tabInter[i] + tabInter[i - 1]
    Fin pour
    Pour i de taille – 1 à 0
        tabRes[--tabInter[tab[i]]] <- tab[i]
    Fin pour
    Pour i de 0 à taille
        tab[i] <- tabRes[i]
    Fin pour
Fin
```

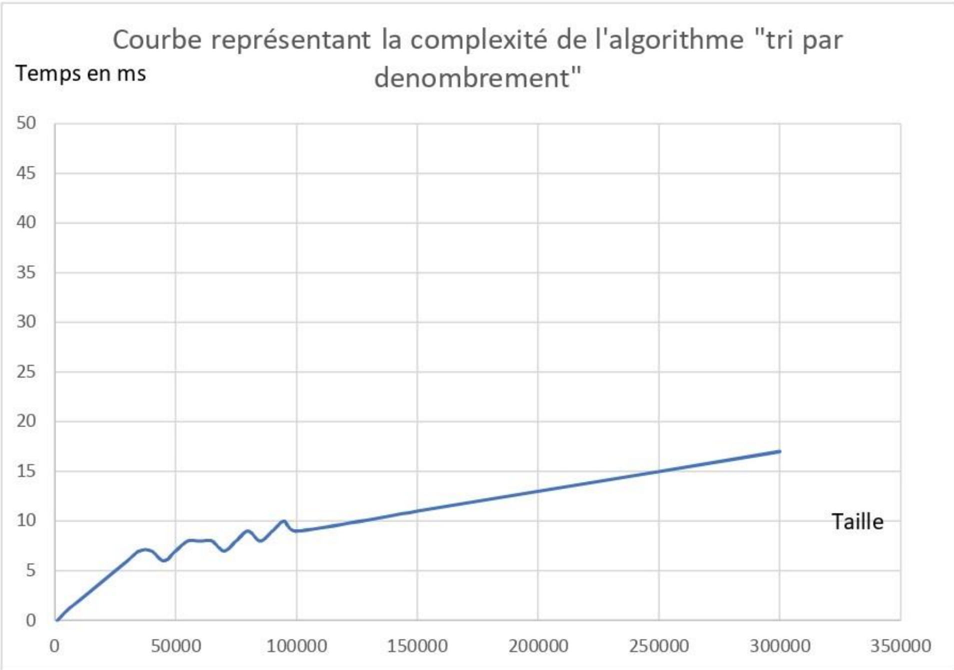
<u>3.2 Etude de complexité de l’algorithme tri par dénombrement</u>

Le tri par dénombrement est l’un des tris les plus efficace pour les tableaux de grandes tailles, il prend en entrée le tableau à trier, et sa taille “n”. Cette méthode consiste à trier le tableau sans comparer les éléments entre eux en utilisant deux tableaux intermédiaires.

Nous avons d’après l’exemple de l’algorithme chaque boucle “pour” (“une boucle for”) à une complexité linéaire c-à-d $O(n)$ car le nombre d’itérations de la boucle est proportionnel à “n” (la taille du tableau).

Ce qui fait que la complexité du tri par dénombrement est $O(n)$ (“complexité linéaire”).

On peut aussi la déduire à partir de la courbe suivante qui représente la complexité du tri par dénombrement.



Taille	Temps en ms
1000	0
5000	1
10,000	2
15,000	3
20,000	4
25,000	5
30,000	6
35,000	7
40,000	7
45,000	6
50,000	7
55,000	8
60,000	8
65,000	8
70,000	7
75,000	8
80,000	9
85,000	8
90,000	9
95,000	10
100,000	9
150,000	11
200,000	13
250,000	15
300,000	17

A partir de la courbe ci-dessus on remarque qu'elle est majoritairement linéaire malgré la présence de quelques variations sinusoïdales ce qui est normal car le tableau initial peut contenir des valeurs redondantes puisque les valeurs du tableau sont générées aléatoirement.

B) Comparaison entre les trois courbes

Le tri à bulles est un algorithme simple à mettre en œuvre de complexité $O(n^2)$ mais très lent pour trier les grands tableaux car au bout d'une certaine taille (par exemple : 100 000) il triera le tableau en 1 minute ce qui n'est pas efficace, par contre un algorithme comme celui du tri rapide qui a une complexité $O\ (n\log(n))$ mettra moins de temps pour le trier (35ms) ce qui est efficace, pourtant ce n'est pas le meilleur car avec des tableaux de 10^6 ou 10^7 il sera lent, il existe un algorithme de complexité $O(n)$ (complexité linéaire) qui triera le même tableau de taille 100 000 en 9ms qui est l'algorithme de tri par dénombrement.

Alors on déduit que les algorithmes de complexité $O(n)$ sont plus efficace que les algorithmes de complexité $O(n\log(n))$, $O(n^2)$ ou $O(n!)$.