

Cours5

Algorithmes de tris – partie2

PLAN

- Tri rapide :
 - Exemple
 - Algorithme
 - Efficacité de l'algorithme
- Tri par comptage de fréquences
 - Tri par comptage
 - Efficacité du tri par comptage
 - Tri par comptages de fréquences
 - Exemple
 - Algorithme
- Tri à bulles

Tri rapide - Rappel

Exemple pas à pas

44	12	65	46	42	23	18	55
0	1	2	3	4	5	6	7

Problème global : trier ce tableau

Pivot : 44 en case 0

Première séparation (à préciser...) :

18	12	23	42	44	46	65	55
0	1	2	3	4	5	6	7

Pivot = 44 est correctement placé

2 sous-tableaux :

- à gauche de 44 : de 0 à 3 (à trier)
- à droite de 44 : de 5 à 7 (à trier)

Exemple pas à pas

18	12	23	42
0	1	2	3

Sous-problème : trier ce tableau

Pivot : 18 en case 0

Séparation :

12	18	23	42
0	1	2	3

Pivot = 18 est correctement placé

2 sous-tableaux :

- à gauche de 18 : de 0 à 0 (tableau trié)
- à droite de 18 : de 2 à 3 (à trier)

Exemple pas à pas

23	42
2	3

Sous-problème : trier ce tableau

Pivot : 23 en case 2

Séparation :

23	42
2	3

Pivot = 23 est correctement placé

2 sous-tableaux :

- à gauche de 23 : tableau trié
- à droite de 23 : de 3 à 3 (tableau trié)

Exemple pas à pas

Ce n'est pas fini...

18	12	23	42	44	46	65	55
0	1	2	3	4	5	6	7

Partie à droite de 44

46	65	55
5	6	7

Exemple pas à pas

Partie à droite de 44

46	65	55
-----------	----	----

5 6 7

Sous-problème : trier ce tableau

Pivot : 46 en case 5

Séparation :

46	65	55
-----------	----	----

5 6 7

Pivot = 46 est correctement placé

2 sous-tableaux :

- à gauche de 46 : tableau trié
- à droite de 46 : de 6 à 7

Exemple pas à pas

65	55
6	7

Sous-problème : trier ce tableau

Pivot : 65 en case 6

Séparation :

55	65
6	7

Pivot = 65 est correctement placé

2 sous-tableaux :

- à gauche de 65 : de 6 à 6 (tableau trié)
- à droite de 65 : tableau trié

Exemple pas à pas

Au final, on recolle tous les tableaux à 1 case triés.

12	18	23	42	44	46	55	65
0	1	2	3	4	5	6	7

Le tableau est trié, le problème global est résolu.

Séparation

44	12	65	46	42	23	18	55
-----------	----	----	----	----	----	----	----

0 1 2 3 4 5 6 7

18	12	65	46	42	23	44	55
-----------	----	----	----	----	----	-----------	----

0 1 2 3 4 5 **6** 7

18	12	44	46	42	23	65	55
----	----	-----------	----	----	----	-----------	----

0 1 **2** 3 4 5 **6** 7

18	12	23	46	42	44	65	55
----	----	-----------	----	----	-----------	----	----

0 1 **2** 3 4 **5** 6 7

18	12	23	44	42	46	65	55
----	----	----	-----------	----	-----------	----	----

0 1 2 **3** 4 **5** 6 7

18	12	23	42	44	46	65	55
----	----	----	-----------	-----------	----	----	----

0 1 2 **3** **4** 5 6 7

Algorithme de séparation

```
/* *  
 * Cette méthode doit placer correctement le pivot  
 * dans le tableau délimité par les indices indL et indR.  
 * Au final, la place du pivot dans le tableau est tel que  
 * tous les éléments à sa gauche sont nécessairement  
 * + petits ou égaux et tous les éléments à sa droite  
 * sont nécessairement + grands ou égaux.  
 * @param tab le tableau dans lequel placer le pivot  
 * @param indL l'indice de début du tableau  
 * @param indR l'indice de fin du tableau  
 * @return l'indice du tableau où se trouve le pivot  
 */
```

```
int separer ( int [ ] tab, int indL, int indR ) {
```

```
    int ret = indL;
```

```
    int pivot = tab[indL]; // le pivot est par convention le  
                          // premier élément du tableau
```

```
    while ( ... ) {
```

```
        // ToDo : à vous d'écrire l'algorithme de placement  
        // du pivot
```

```
    }
```

```
    return ret;
```

```
}
```

Algorithme du Tri rapide

```
void triRapideRec ( int [ ] tab, int indL, int indR ) {  
    // variable locale  
    int indS;      // indS = indice de séparation  
  
    indS = separer ( tab, indL, indR );  
  
    // tri sur le tableau à gauche du pivot  
    triRapideRec ( tab, indL, (indS-1) );  
  
    // tri sur le tableau à droite du pivot  
    triRapideRec ( tab, (indS+1), indR );  
}
```

Cet algorithme est RECURSIF !

La récursivité sera vue au dernier cours.

Tel quel cet algorithme est incomplet :

- il ne s'arrête pas !
- il ne démarre pas !

Algorithme du Tri rapide

Démarrage

Appeler une toute première fois triRapideRec avec comme paramètres :

- $\text{indL} = 0$ (première case du tableau initial)
- $\text{indR} = n - 1$ (dernière case du tableau initial)

```
void triRapide ( int [ ] tab, int nbElem ) {  
    triRapideRec ( tab, 0, (nbElem-1) );  
}
```

Algorithme du Tri rapide

Arrêt

La récursivité doit s'arrêter :

- si le tableau à trier ne contient qu'une seule case, dans ce cas $\text{indL} = \text{indR}$
- arrêt si $\text{indR} < \text{indL}$
- arrêt si $\text{indL} > \text{indR}$

```
void triRapideRec ( int [ ] tab, int indL, int indR ) {  
    // variable locale  
    int indS;      // indS = indice de séparation  
  
    indS = separer ( tab, indL, indR );  
  
    if ( (indS-1) > indL ) {  
        triRapideRec ( tab, indL, (indS-1) );  
    }  
  
    if ( (indS+1) < indR ) {  
        triRapideRec ( tab, (indS+1), indR );  
    }  
}
```

Efficacité du tri rapide

Au départ, un tableau de taille « n ». Une opération élémentaire = une comparaison.

pivot
-------	-----	-----	-----	-----	-----	-----	-----

1 2 3 n

Etape $i = 1$

Pour placer le pivot, il y a « n » comparaisons.

$$n = n / 2^{(i-1)} \text{ (car } i = 1 \text{)}$$

1^{ère} séparation ($i = 1$) meilleur des cas :
séparation en 2 sous-tableaux de même
taille.



...
-----	-----	-----	-----

...
-----	-----	-----	-----

1 $n/2$ $n/2+1$ n

Efficacité du tri rapide

Etape $i = 2$

pivot
-------	-----	-----	-----

1 $n/2$

Pour placer le pivot,
il y a « $n/2$ »
comparaisons.

$$n/2 = n / 2^{(i-1)}$$

pivot
-------	-----	-----	-----

$n/2+1$ n

Pour placer le pivot,
il y a « $n/2$ »
comparaisons.

$$n/2 = n / 2^{(i-1)}$$

Au total il y a $n/2 + n/2 = n$ comparaisons

2^{ème} séparation ($i = 2$) meilleur des cas :
séparation en 2 sous-tableaux de même
taille.

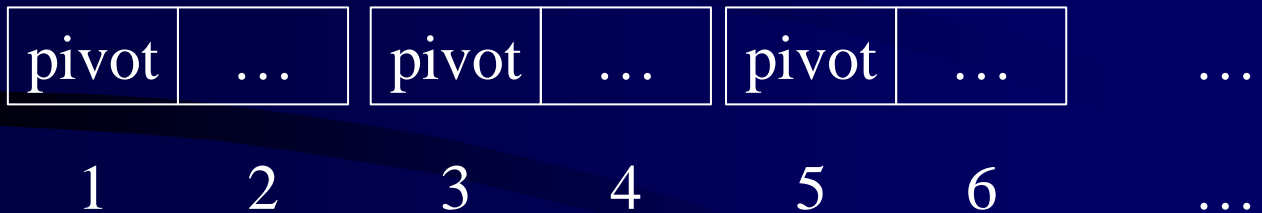
...
-----	-----	-----	-----	-----	-----	-----	-----

1 $n/2$ $n/2+1$ n

Efficacité du tri rapide

Etape $i = k$ (dernière étape)

Que des tableaux de taille 2 (car à l'étape suivante il n'y a que des tableaux de taille un et la comparaison n'est plus nécessaire).



A la dernière étape il y a à nouveau $1 + 1 + \dots + 1 = n$ comparaisons.

Nombre total de comparaisons $f(n)$:

$$f(n) \approx n \times k$$

Efficacité du tri rapide

Nombre total de comparaisons $f(n)$:

$$f(n) \approx n \times k$$

Que vaut k ?

k est tel que : $2 = n / 2^k$

$$\Leftrightarrow$$

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

$$\Leftrightarrow$$

$$\log_2 n = \log_2 2^{(k+1)} = k + 1$$

$$f(n) \approx n \times \log_2 n$$

L'algorithme du tri rapide est en **$\Theta(n \log_2 n)$** pour « n » élevé et dans le meilleur des cas.

Tri par comptage de fréquences

Tri par comptage

On suppose :

- un tableau de valeurs quelconques
- toutes les valeurs sont **distinctes** (aucun doublon de valeurs)

Alors le tri est facile...

Tri par comptage

44	25	0	46	42	23	18	55
0	1	2	3	4	5	6	7

Dans l'exemple :

- 44 doit être placé en 5 car il y a 5 valeurs + petites que 44
- 25 doit être placé en 3 car il y a 3 valeurs + petites que 25
- 0 doit être placé en 0 car il y a 0 valeur + petite que 0
- etc.

Généralisation pour placer $\text{tab}[i]$:

- Compter le nombre de valeurs $< \text{tab}[i]$ = la nouvelle place de $\text{tab}[i]$ dans le tableau trié par ordre croissant

Efficacité du tri par comptage

44	25	0	46	42	23	18	55
0	1	2	3	4	5	6	7

Dans l'exemple :

- 44 doit être comparé à tous les autres
- 25 doit être comparé à tous les autres
- 0 doit être comparé à tous les autres
- etc.

Deux boucles imbriquées :

Dans ce cas-ci, efficacité en $\Theta(n^2)$ pour n ↗

Tri par comptage de fréquences

Adaptation du tri par comptage au cas particuliers des tableaux qui respectent les conditions suivantes :

- que des entiers
- les entiers sont positifs (ou égaux à zéro)
- l'entier max n'est pas trop grand (≤ 500 par exemple)
- les doublons sont possibles...

Alors le tri est rapide...

Idée : « éclater » le tableau initial dans un tableau beaucoup plus grand qui sera une image du tableau final trié.

Exemple

Soit le tableau à trier initial « leTab ».

44	25	0	44	44	25	18	55
0	1	2	3	4	5	6	7

On crée d'abord un tableau « tabFreq » dont la taille = $\max(\text{leTab}) + 1 = 56$

« tabFreq » (beaucoup + grand) sera :

0	...0	...	0	...	0	...	0
0	...18	...	25	...	44	...	55

Les indices de ce tableau sont les entiers du tableau initial « leTab ».

Exemple

Le tableau à trier initial « leTab ».

44	25	0	44	44	25	18	55
0	1	2	3	4	5	6	7

Le tableau « tabFreq » indicé de zéro à $\max(\text{leTab})$ contiendra dans chaque case la fréquence d'apparition de chaque entier du tableau initial.

1	...1	0	2	0	3	0	1
0	...18	...	25	...	44	...	55

Exemple

1	...1	0	2	0	3	0	1
0	...18	...	25	...	44	...	55

Ce tableau « tabFreq » indicé de zéro à $\max(\text{leTab})$ est une image du futur tableau trié car :

- ces indices sont précisément les valeurs que l'on retrouvera dans le tableau trié
- ces indices sont forcément organisés par ordre croissant des valeurs

« Il n'y a plus qu'à » construire le tableau trié à partir de « tabFreq ».

Exemple

Le tableau « tabFreq »

1	...1	0	2	0	3	0	1
0	...18	...	25	...	44	...	55

Le tableau trié « tabTrie »

?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7

Il faut y mettre :

- 1 X 0
- 1 X 18
- 2 X 25
- 3 X 44
- 1 X 55

Algorithme

Le tableau « tabFreq »

1	...1	0	2	0	3	0	1
0	...18	...	25	...	44	...	55

Le tableau trié « tabTrie »

?	?	?	?	?	?	?	?
0	1	2	3	4	5	6	7

- Parcourir le tableau « tabFreq »
- Initialiser un compteur de case à zéro (cpt=0)
- Si $\text{tabFreq}[i] = k \ (\neq 0)$ alors remplir le tableau tabTrie de k valeur « i » en commençant par la case « $\text{tabFreq}[i] - 1 + \text{cpt}$ » et en terminant en « $\text{tabFreq}[i] - 1 + \text{cpt} + k - 1$ »
- Incrémenter cpt : $\text{cpt} = \text{cpt} + k$

Particularités d'un tel tri

- Efficacité de l'algorithme : plutôt bon, meilleur que le tri rapide ?
- Ne fonctionne pas « tel quel » pour des entiers négatifs ni pour des réels.
- Si $\max(\text{leTab})$ est très grand alors « tabFreq » est très grand, il y a donc une limite.
- Si il y a beaucoup de doublons, le tableau « tabFreq » est très creux.

Tri à bulles