Algorithmes de tri interne (2) [tr]
Méthodes par insertions

Karine Zampieri, Stéphane Rivière, Béatrice Amerein-Soltner

Unisciel algoprog Version 21 mai 2018

Table des matières

Algorithmes de tri interne (2)

Méthodes par insertions

Les **méthodes par insertions** travaillent par *insertion* d'un élément dans une partie déjà triée.

Méthodes par insertions

```
Action trParInsertions ( DR A : Element [ NMAX ] ; n : Entier )
Début
  |  Si ( n > 1 )
  |    |  trParInsertions ( A , n - 1 )
  |    |  inserer ( A , n - 1 , A [ n ] )
  |  FinSi
Fin
```

Le « tri par insertion » réalise des insertions successives d'un élément. Des améliorations nous amèneront à l'algorithme connu sous le nom de « tri de Shell ».

1 Tri par insertion

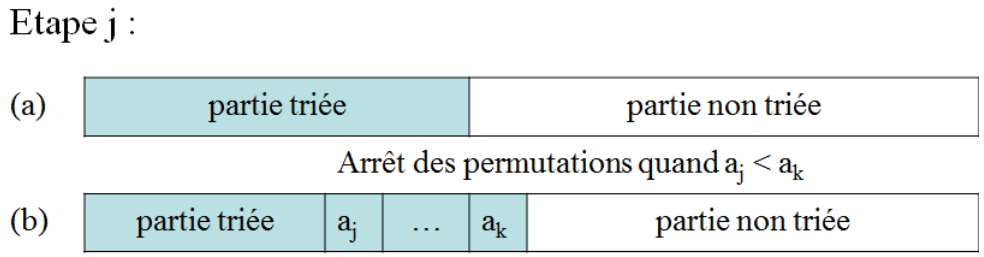
Nom anglais : *insertion sort*

Propriétés : tri interne, sur place, stable

Visualisation : http://www.sorting-algorithms.com/insertion-sort

1.1 Principe du tri par insertion

Le **tri par insertion** insère, au fur et à mesure, l'élément frontière en position `j` dans la partie triée.



Remarque

La stratégie est identique à celle utilisée par les joueurs de cartes.

1.2 Tri par insertion en récursif

La procédure de tri par insertion de façon récursive insère, en décalant les éléments vers la gauche, le premier élément de la partie non triée de n éléments dans une partie triée (par appel récursif) des $n-1$ éléments suivants. Le cas de base est le cas où la partie à trier ne contient qu'un seul élément et est, donc, déjà triée.

Procédure trInsertionRec

(Tri par insertion en récursif)

```
Action trInsertionRec ( DR A : Element [ NMAX ] ; n : Entier )
Début
  |  Si ( n > 1 ) Alors
  |    |  trInsertionRec ( A , n - 1 )
```

```
  |    |  insererTab ( A , n - 1 , t [ n ] )
  |  FinSi
Fin
Action insererTab ( DR A : Element [ NMAX ] ; k : Entier )
Début
  |  j <- k - 1
  |  TantQue ( j > 0 Et A [ j + 1 ] < A [ j ] ) Faire
  |    |  permuterTab ( A , j , j + 1 )
  |    |  j <- j - 1
  |  FinTantQue
Fin
```

1.3 Tri par insertion en itératif

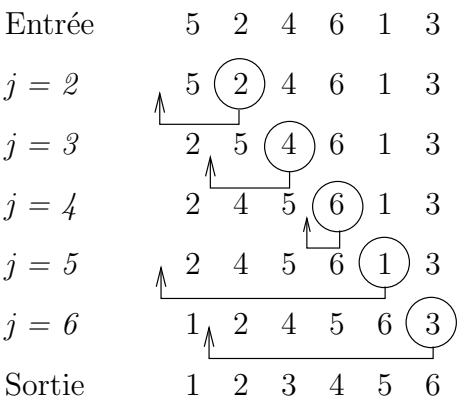
Sous forme non récursive, le tri par insertion s'écrit :

Procédure trInsertion
(Tri par insertion en itératif)

```
Action trInsertion ( DR A : Element [ NMAX ] ; n : Entier )
Début
  |  Pour j <- 2 à n Faire
  |    |  insererTab ( t , j - 1 , t [ j ] )
  |  FinPour
Fin
```

1.4 Illustration du tri par insertion

Les différentes étapes sur le tableau [5,2,4,6,1,3] sont présentées sur la figure suivante où l'élément à insérer est entouré par un cercle.



1.5 Complexité du tri par insertion

Nombre de comparaisons
Dans la procédure récursive, trier un tableau de n éléments revient à trier récursivement un tableau de $n-1$ éléments et y insérer un élément. Si $C(n)$ désigne le nombre de

comparaisons pour trier un tableau de taille n et $I(n)$ le nombre de comparaisons pour insérer un élément à sa place dans un tableau trié de taille n, on a :

- $C(1) = 0$: lorsque le tableau n'a qu'un élément on ne fait aucune comparaison
- pour $n > 1$: $C(n) = C(n-1) + I(n-1)$.

Donc :

$$C(n) = I(n-1) + I(n-2) + ... + I(1)$$

Or $I(n)$ vaut au maximum n et en moyenne $(n+1)/2$. Dans le pire cas (tableau trié dans l'ordre inverse), la récurrence se résout en :

$$C_{max}(n) = \sum_{k=2}^{n} k = \frac{n(n+1)}{2} - 1 \in \Theta(n^2)$$

En moyenne, la récurrence se résout en :

$$C_{moy}(n) = \frac{n}{2} + \frac{n-1}{2} + ... + \frac{2}{2} = \frac{n(n-1)}{4} + 1/2 \in \Theta(n^2)$$

Donc l'algorithme est bien quadratique à la fois dans le pire cas et en moyenne.

Nombre d'échanges

Le nombre d'échanges dans le pire cas (= majorant du nombre d'échanges) est celui où le vecteur est classé dans l'ordre inverse et donc chaque cellule doit être permutée : dans ce cas il y a donc autant d'échanges que de tests :

$$E(n) = C(n) \in \Theta(n^2)$$

Remarque

L'utilisation de la *recherche dichotomique* n'améliore pas son efficacité : elle la dégrade même en général puisqu'elle entraîne environ $\lg n$ tests mais aussi $n/2$ déplacements en moyenne par insertion.

1.6 Propriétés du tri par insertion

On vérifie aisément que cette méthode de tri est **stable**.

On notera également une **propriété importante** du tri par insertion : son efficacité est meilleure si le tableau initial possède un certain ordre. Dans le cas du tableau « idéal », initialement trié, la complexité est $\Omega(n)$. Plus généralement, l'algorithme tirera parti de tout ordre partiel présent dans le tableau. Jointe à la simplicité de l'algorithme, cette propriété le désigne tout naturellement pour « terminer le travail » de méthodes plus ambitieuse qui, comme le tri rapide, séparent rapidement un tableau en parties « presque » triées mais s'essoufflent lors du tri définitif de petits sous-tableaux.

2 Tri de Shell

Nom anglais : *Shell sort*

Propriétés : tri interne, non stable, sur place

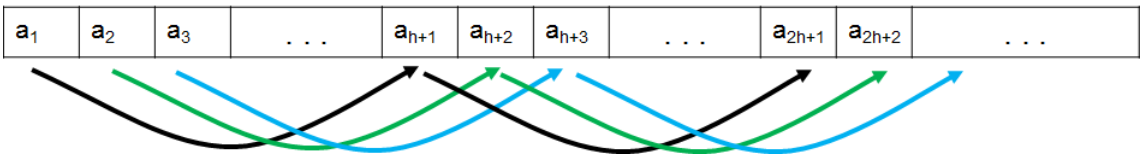
Visualisation : http://www.sorting-algorithms.com/shell-sort

Le **tri de Shell** (appelée aussi tri par *insertion avec incréments dégressifs*) fut proposé en 1959 par Donald Shell en tant qu'amélioration du tri par insertion, puis optimisé par Shell et Metzner et analysé par [Pratt 1979].

2.1 La méthode de Shell

Le principal inconvénient du tri par insertion est que chaque élément n'est déplacé que d'une position à la fois. Ainsi si le m-ème élément du tableau doit être déplacé en position 1, il faut déplacer d'une position vers la droite les $m-1$ éléments précédents. Chacun de ces déplacements supprime exactement *une inversion* dans le tableau. Il en résulte que le nombre total de mouvements de données est au moins égal au nombre initial des inversions, qui vaut en moyenne $\frac{n(n-1)}{4}$.

Pour s'affranchir de cette limite inférieure sans remettre en cause complètement la méthode, D.L. Shell proposa la solution suivante : au lieu d'insérer systématiquement un élément d'indice k dans le sous-tableau des éléments qui le précède, procédé qui contredit le principe d'« équilibrage » pour pouvoir mener à un algorithme efficace, on insèrera cet élément dans le sous-tableau comprenant les éléments d'indices $k-h$, $k-2h$, $k-3h$, etc. h étant une constante positive. On obtient ainsi un tableau où les h-séries disjointes d'éléments distants de h sont triées séparément :



Remarque

Il est naturel d'attendre un progrès de cette méthode qui utilise les propriétés de l'accès direct aux éléments d'un tableau, alors que l'insertion simple était une suite de parcours séquentiels, réalisable en représentation chaînée aussi bien qu'en représentation contiguë.

Méthode

Pour obtenir un tableau trié, on recommence avec une nouvelle valeur $h' < h$. Du fait de la propriété caractéristique du tri par insertion, la tâche sera plus facile que si l'on partait d'un tableau quelconque : le tri préalable « par séries de distance h » accélère le tri « par séries de distance h ». L'algorithme de Shell utilise cette propriété magique en choisissant de trier « par séries de distance h_k » puis h_{k-1}, ..., puis h_1 où $h_j, j = k, k-1, ..., 1$ est une suite décroissante d'« incréments » avec $h_1 = 1$ pour que le dernier passage soit assuré de trier définitivement le tableau.

2.2 Algorithme de Shell

Algorithme de base

```
Action trShell ( DR A : Element [ NMAX ] ; n : Entier )
Début
  |  increment <- calculIncrement ( n )
  |  // tris successifs par séries
  |  Répéter
  |    |  // trier par séries De distance "increment"
  |    |  Pour k <- 1 à increment Faire
  |    |    |  trier par insertion ( = tri bulle ) la k - ème série
  |    |  FinPour
  |    |  incrementSuivant ( increment )
  |  Jusqu'à ( increment = 0 )
Fin
```

La suite des incréments

Quelles valeurs successives $h_j, j = k, k-1, ..., 1$ doit-on choisir pour l'incrément ? Il n'y a pas de réponse absolue à cette question car les valeurs optimales dépendent de la taille du tableau à trier. L'analyse mathématique de l'algorithme, en dehors de quelques suites d'incréments particulières, est extrêmement complexe : on ne connaît pas de suites dont on puisse démontrer qu'elles sont optimales (voir [Knuth 73]). Pour des tableaux assez grands, les résultats de tests montrent qu'il est conseillé de prendre la suite h_k telle que $h_{k+1} = 3h_k + 1$ donc comprenant dans l'ordre les incréments : ... 364, 121, 40, 13, 4, 1. On partira de h_{m-2} où m est le plus petit entier tel que $h_m \geq n$. En d'autres termes h_{m-2} est le premier élément de la suite supérieur ou égal à $\lfloor n/9 \rfloor$. L'algorithme de SHELL s'écrit alors :

Algorithme de Shell

```
Action trShell ( DR A : Element [ NMAX ] ; n : Entier )
Début
  |  increment <- calculIncrement ( n )
  |  Répéter
  |    |  Pour k <- 1 à increment Faire
  |    |    |  Pour ix <- increment + k à n Pas increment Faire
  |    |    |    |  j <- ix - increment
  |    |    |    |  TantQue ( j >= 1 Et t [ ix ] < t [ j ] ) Faire
  |    |    |    |    |  permuter ( t , ix , j )
  |    |    |    |    |  ix <- j
  |    |    |    |    |  j <- j - increment
  |    |    |    |  FinTantQue
  |    |    |  FinPour
  |    |  FinPour
  |    |  incrementSuivant ( increment )
  |  Jusqu'à ( increment = 0 )
Fin
Fonction calculIncrement ( n : Entier ) : Entier
Variable h : Entier
Variable ndiv9 : Entier
Début
  |  ndiv9 <- DivEnt ( n , 9 )
```

```
  |  h <- 1
  |  TantQue ( h < ndiv9 ) Faire
  |    |  h <- 3 * h + 1
  |  FinTantQue
  |  Retourner ( h )
Fin
Action incrementSuivant ( DR h : Entier )
Début
  |  h <- DivEnt ( h , 3 )
Fin
```

2.3 Illustration du tri de Shell

La figure illustre le tri par insertion avec incréments dégressifs de D. Shell sur le tableau [7,19,24,13,31,8,82,18,44,63,5,29]. Notez que si deux éléments ont été échangés au k-ème passage alors ils restent triés dans la suite.

1è passe : $h=5$	7	19	24	13	31	8	82	18	44	63	5	29
2è passe : $h=4$	7	19	18	13	31	5	29	24	44	63	8	82
3è passe : $h=3$	7	5	18	13	31	19	8	24	44	63	29	82
4è passe : $h=2$	7	5	18	8	24	19	13	29	44	63	31	82
5è passe : $h=1$	7	8	18	13	18	19	24	29	31	63	44	82
6è passe : Triée	5	7	8	13	18	19	24	29	31	44	63	82

2.4 Complexité du tri de Shell

Nombre de comparaisons

Le nombre de comparaisons effectué par l'algorithme est fonction de la suite (h_i) des incréments. Quand il y a exactement deux incréments h et 1, il a été démontré que le meilleur choix pour h est approximativement $1.72\sqrt[3]{n}$ et qu'avec ce choix, le coût moyen temporel est proportionnel à $n^{5/3}$. En particulier pour l'incrément $h_i = 2^i - 1$, $(i = 1, \ldots, \lfloor \log n \rfloor)$, le nombre moyen de comparaisons effectuées dans le pire des cas est en $O(n^{1.5})$.

Nombre de transferts

Le nombre de transferts (déplacements) est en moyenne, quand n est grand, de l'ordre de $1.66n^{1.25}$. Les valeurs approchées de cette valeur soutient la comparaison avec des méthodes en $O(n \lg n)$.

2.5 Propriétés

Le tri de Shell n'**est pas stable** en général ce qui peut être gênant : il n'y a pas de moyen simple de corriger ce problème. Il est sensiblement meilleur que l'insertion simple. La méthode soutient la comparaison avec des méthodes en $O(n \lg n)$ jusqu'à $n = 10^4$.