Les algorithmes de tri

par Romuald Perrot

Date de publication : 13/05/2006

Dernière mise à jour : 13/05/2006

Présentation des algorithmes de tris.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

Les algorithmes de tri sont à la base de la programmation. En effet, il s'agit des premiers algorithmes fondamentaux que l'on apprend. Il occupent une place majeure dans la programmation puisqu'on les trouve partout, dès qu'on a besoin de trier des données, on fait appel à eux.

Cet article vise à étudier quelques uns des algorithmes de tri.

I - Tri par insertion

I-A - Principe

Le tri par insetion est un tri que l'on pratique lorsque l'on joue aux cartes. En effet ce tri repose sur le principe suivant :

On dispose des i premiers éléments du tableau triés, on place le i+1 ème élement à sa bonne place dans la partie du tableau trié. On réitère ce procédé jusqu'à avoir trié tous les éléments du tableau.

La preuve que ceci marche est évidente. Si on place un élément dans un tableau trié en respectant les relations d'ordre, on obtient toujours un tableau trié.

Voici un petit exemple de l'algorithme :

TABLEAU 10 élements. En bleu, vous pouvez apercevoir la partie triée du tableau, on cherche à placer le premier élément qui est juste après la partie triée du tableau.

On insère donc ... à sa position.

TABLEAU_I+1 On recommence le procédé avec ...

TABLEAU_I+2 En réitérant sur tous les éléments on obtient le tableau trié :

TABLEAU_I_N

I-B - Pseudo code

Maintenant que nous avons vu le principe de l'algorithme, il est temps de dégager un algorithme formel.

Cet algorithme va en fait se décomposer en deux procédures principales.

La première insère un élément à sa place dans un tableau trié et la deuxième va appeler la première procédure pour tous les éléments du tableau.

Deux questions se posent maintenant : l'initialisation et la terminaison. Pour ce qui est de l'initialisation, on fait appel à une propriété évidente : un tableau à un élément est un tableau trié. A partir de cette propriété, on peut simplement commencer l'algorithme en considérant la partie trié du tableau comme étant le sous tableau contenant le premier élément.

Pour ce qui est de la fin de l'algorithme, c'est plutôt simple : l'algorithme se termine lorsque l'on a passé tous les éléments du tableau.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

A partir de ces deux conditions, on peut établir la procédure principale de l'algorithme:

```
procedure tri_insertion( T : Tableau )
debut
     pour i allant de 2 à n faire
          placer(T,T[i],i-1);
fin procedure
```

Voici l'algorithme. Nous avons considéré que le tableau était composé de n éléments indèxés de 1 à n.

La procédure placer place l'élément T[i] dans le tableau trié (qui est trié entre 1 et i-1). Il ne nous reste plus qu'à écrire cette procédure.

Pour se faire, il faut savoir comment nous pouvons insérer un élément dans un tableau trié. Ceci est fait simplement en décallant les éléments du tableau trié sur la droite tant que l'on a pas trouvé l'endroit où on devait insérer notre élément.

On arrête le décalage des éléments dès que l'on a trouvé une position qui respecte les relations d'ordre. Ceci peut être résumer ainsi : si l'élément en cours est plus grand que l'élément à insèrer, on décalle. Sinon, on arrête le décallage et on insère.

A partir de ceci nous pouvons écrire la procédure :

```
procedure placer( T : Tableau; elt : Element; fin : Entier )
debut
     i <-fin

     tant que i > 1 et T[i] > elt faire
          T[i+1] <- T[i];
          i <- (i - 1);
     fin tant que

         T[i+1] <- elt
fin procedure
```

Voilà pour la procédure placer, si nous résumons voici le pseudo code de l'algorithme de tri par insertion.

```
procedure placer( T : Tableau; elt : Element; fin : Entier )
debut
     i <-fin

     tant que i > 1 et T[i] > elt faire
          T[i+1] <- T[i];
          i <- (i - 1);
     fin tant que

         T[i+1] <- elt
fin procedure

procedure tri_insertion( T : Tableau )
debut
     pour i allant de 2 à n faire
          placer(T,T[i],i-1);
fin procedure
```

I-C - Complexité

Autant vous le dire tout de suite, ce algorithme bien que simple sur le papier est assez catastrophique en terme de complexité. Il ne sera donc utilisé qu'à des fins pédagogique ou alors pour de très faibles jeux de données.

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

Etudions donc de plus près la complexité en temps de l'algorithme.

Si nous résumons brièvement, nous avons une boucle allant de 2 à n qui appelle une fonction placer. Pour un début, nous pouvons donc dire que la complexité est (n-2) fois la complexité de la procédure placer.

Pour ce qui est de la complexité de la procédure placer, celle ci n'est pas fixe, en effet, nous ne savons pas combien d'éléments nous devrons décaler avant d'insérer. Nous allons donc dégager trois cas : le cas favorable (celui qui minimise le nombre d'élément à décaler), le cas défavorable (celui qui maximise le nombre d'éléments à décaler) et enfin le cas général.

Le cas favorable est le cas ou on a le moins de décallage à faire. Celui ci intervient lorsque le tableau est déjà trié. En effet, lorsuq le tableau est trié, l'élément à placer est déjà à sa bonne position. Dans ce cas, chaque appel à placer est en temps constant O(1). Du coup, l'algorithme du tri sera en temps linéaire : O(n) (en fait en théta de n pour être plus précis)

Le cas défavorable intervient lorsque l'on doit décaler tous les éléments à chaque fois. Ceci arrive lorsque le tableau est trié à l'envers. Dans ce cas, le nombre de décallage est 1 puis 2, 3, 4, ... jusqu'à n-1. On a alors la somme de Gauss : (n-1)(n-2)/2. Ce qui fait donc que la complexité totale est en (n^2)/2+n/2. Ce qui se résumera en O(n^2).

Maintenant le cas général s'obtient en considérant que l'on décalle seulement la moitié des éléments du tableau trié lors d'une insertion. Ce qui fait en complexité pour l'algorithme: (n^2)/4 + n/4 soit donc toujours O(n^2).

On a donc une complexité en temps de l'ordre de O(n^2) pour le cas général. Pour vou donner un ordre d'idée, celà veut dire que si on double le nombre d'élement à trier, on multiplie par 4 le temps d'éxécution, ce qui est considérable.

I-D - Code source

Voici un petit exemple en C du tri par insertion :

```c
#include <stdio.h>
#include <stdlib.h>

void affiche( int * tab, char * ch, unsigned n)
{
    unsigned i;
    printf("%s : \n",ch);
    for( i = 0 ; i < n ; i++ )
    {
        printf("%d ",tab[i]);
    }

    printf("\n");
}

void placer( int * tab, int elt, int ind, int n)
{
    int i = ind;

    while( ( i >= 0 ) && tab[i] > elt )
    {
        tab[i+1] = tab[i];
        i--;
    }

    tab[i+1] = elt;
}

void tri_insertion(int * tab, int n)
{
    int i;
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

```
    for( i = 1 ; i < n ; i++ )
    {
        placer(tab,tab[i],i-1,n);
    }
}


int main ( void )
{
    int tab[] ={10,5,19,38,47,50,68,8,5};
    affiche(tab,"avant le tri",9);
    tri_insertion(tab,9);
    affiche(tab,"apres le tri",9);

    return EXIT_SUCCESS;
}
```


Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

II - Tri par sélection

II-A - Principe

II-B - Pseudo code

II-C - Complexité

II-D - Code source

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

III - Tri à bulle

III-A - Principe

III-B - Pseudo code

III-C - Complexité

III-D - Variante : le tri shakker

III-E - Code source

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

IV - Tri par fusion

IV-A - Principe

IV-B - Pseudo code

IV-C - Complexité

IV-D - Code source

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

V - Tri par tas

V-A - Principe

V-B - Pseudo code

V-C - Complexité

V-D - Code source

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/

VI - Tri rapide (quick sort)

VI-A - Principe

VI-B - Pseudo code

VI-C - Complexité

VI-D - Code source

Les sources présentés sur cette pages sont libres de droits, et vous pouvez les utiliser à votre convenance. Par contre la page de présentation de ces sources constitue une oeuvre intellectuelle protégée par les droits d'auteurs. Copyright © 2006 - Romuald Perrot. Aucune reproduction, même partielle, ne peut être faite de ce site et de l'ensemble de son contenu : textes, documents, images, etc sans l'autorisation expresse de l'auteur. Sinon vous encourez selon la loi jusqu'à 3 ans de prison et jusqu'à 300 000 E de dommages et intérêts.
http://rperrot.developpez.com/article/algo/theorie/tris/