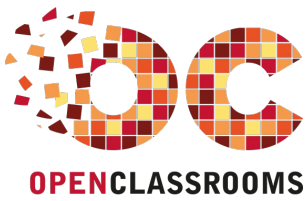
Le tri à bulles

Par Olivier Strebler (shareman)



www.openclassrooms.com

Licence Creative Commons 7 2.0
Dernière mise à jour le 6/07/2011

Sommaire

Le tri à bulles

Par  Olivier Strebler (shareman)

Mise à jour : 06/07/2011

(cc) BY-SA

Bonjour les zér0s !

Quel est le premier algorithme de tri auquel on pense intuitivement lors de nos premiers pas dans le monde du tri? On a tendance à comparer deux à deux les éléments d'un tableau ou d'une liste à trier et d'échanger leur position s'ils sont mal placés. Et bien, ce tri porte un nom : c'est le tri à bulles.

Dans son cours sur le langage C, et plus particulièrement dans le chapitre sur les tableaux, m@teo21 donne pour exercice (n°5) l'élaboration d'une méthode (d'un algorithme) pour trier un tableau. On se rend compte que la plupart des zér0s implémentent (peut-être sans le savoir) le tri à bulles. En réalité, un débutant a 99% de chances de l'implémenter dans ses débuts en algorithmique.

L'algorithme est original (mais malheureusement lent) et je vais vous le présenter ici.
Sommaire du tutoriel :

Principe du tri

Le principe de l'algorithme du tri à bulles est très simple à assimiler. Il est, et de loin, l'un des algorithmes de tri les plus simples qui soient.

Pour vous expliquer le principe, je vais d'abord vous donner une courte explication écrite puis nous allons concrètement trier une liste de nombres.

Le principe du tri à bulles est de comparer deux valeurs adjacentes et d'inverser leur position si elles sont mal placées. Alors, qu'entend-t-on par "mal placé" ? C'est très simple et surtout, c'est logique : si un premier nombre x est plus grand qu'un deuxième nombre y et que l'on souhaite trier l'ensemble par ordre croissant, alors x et y sont mal placés et il faut les inverser. Si, au contraire, x est plus petit que y, alors on ne fait rien et l'on compare y à z, l'élément suivant. C'est donc itératif. Et on parcourt ainsi la liste jusqu'à ce qu'on ait réalisé n-1 passages (n représentant le nombre de valeurs à trier) ou jusqu'à ce qu'il n'y ait plus rien à inverser lors du dernier passage.

Avec de la logique, on s'aperçoit qu'au premier passage, on place le plus grand élément de la liste au bout du tableau, au bon emplacement. Pour le passage suivant, nous ne sommes donc plus obligés de faire une comparaison avec le dernière élément ; et c'est bien plus avantageux ainsi. Donc à chaque passage, le nombre de valeurs à comparer diminue de 1.

Pour illustrer ce principe, prenons la suite de nombres suivante :

Code : Autre - liste à trier

```
6 0 3 5 1 4 2
```

Nous voulons trier ces valeurs par ordre croissant. Commençons par le commencement. Nous allons faire un premier passage.

Code : Autre - liste à trier

```
6 0 3 5 1 4 2   // On compare 6 et 0 : on inverse
0 6 3 5 1 4 2   // On compare 6 et 3 : on inverse
0 3 6 5 1 4 2   // On compare 6 et 5 : on inverse
0 3 5 6 1 4 2   // On compare 6 et 1 : on inverse
0 3 5 1 6 4 2   // On compare 6 et 4 : on inverse
0 3 5 1 4 6 2   // On compare 6 et 2 : on inverse
0 3 5 1 4 2 6   // Nous avons terminé notre premier passage
```

La question à se poser est la suivante : Devons-nous continuer ? Oui car lors du dernier passage, au moins un échange a été effectué.
Nous allons donc refaire un passage mais en omettant la dernière case.

Code : Autre - liste à trier

```
0 3 5 1 4 2 6   // On compare 0 et 3 : on laisse
0 3 5 1 4 2 6   // On compare 3 et 5 : on laisse
0 3 5 1 4 2 6   // On compare 5 et 1 : on inverse
0 3 1 5 4 2 6   // On compare 5 et 4 : on inverse
0 3 1 4 5 2 6   // On compare 5 et 2 : on inverse
0 3 1 4 2 5 6   // Nous avons terminé notre passage
```

Comme promis, on ne compare pas 5 et 6 car c'est **inutile** et ce qui est inutile est à éviter, d'autant plus que cela ralenti l'algorithme.
Je suppose que vous avez bien compris le principe alors ce que je vous propose maintenant, c'est juste de vous montrer les dernières étapes avant d'aboutir à la liste triée.

Code : Autre - liste à trier

```
0 3 1 4 2 5 6   // On compare 0 et 3 : On laisse
0 3 1 4 2 5 6   // On compare 3 et 1 : On inverse
0 1 3 4 2 5 6   // On compare 3 et 4 : On laisse
0 1 3 4 2 5 6   // On compare 4 et 2 : On inverse
0 1 3 2 4 5 6   // Nous avons terminé notre passage
```

Code : Autre - liste à trier

```
0 1 3 2 4 5 6   // On compare 0 et 1 : On laisse
0 1 3 2 4 5 6   // On compare 1 et 3 : On laisse
0 1 3 2 4 5 6   // On compare 3 et 2 : On inverse
0 1 2 3 4 5 6   // Nous avons terminé notre passage
```

Code : Autre - liste à trier

```
0 1 2 3 4 5 6   // On compare 0 et 1 : On laisse
0 1 2 3 4 5 6   // On compare 1 et 2 : On laisse
0 1 2 3 4 5 6   // Nous avons terminé notre passage
```

À ce moment-là, l'algorithme s'arrête car il n'y a plus eu d'échange lors du dernier passage et nous retrouvons notre liste belle et bien triée !

Implémentation

Dans cette sous-partie, je vais vous montrer une manière d'implémenter le tri à bulles dans un langage de programmation.

Choix du langage

En premier lieu, j'avais souhaité vous présenter une implémentation du tri à bulles en Ti-basic mais apparemment, c'était un mauvais choix. Il est vrai que tout le monde ne comprend pas ce langage (son apprentissage au lycée est facultatif) et le Ti-basic

ne s'indente pas, par conséquent les différents blocs d'instructions sont plus difficilement repérables. J'ai donc rectifié cette sous-partie et j'ai choisi le langage C++, bien plus connu des Zér0s. Même si vous ne programmez pas en ce langage, vous pourrez aisément comprendre le code et ainsi le traduire.

l'algorithme

Je vais procéder de la manière suivante : Je vais vous donner le code puis je vais l'expliquer.

Code : C++ - Algorithme du tri à bulles

```
void tri_bulles(vector<int>& tab)
{
    bool tab_en_ordre = false;
    int taille = tab.size();
    while(!tab_en_ordre)
    {
        tab_en_ordre = true;
        for(int i=0 ; i < taille-1 ; i++)
        {
            if(tab[i] > tab[i+1])
            {
                swap(tab[i],tab[i+1]);
                tab_en_ordre = false;
            }
        }
        taille--;
    }
}
```

Intéressons nous déjà à la première ligne (`void tri_bulles(vector<int>& tab)`), la fonction est de type `void` et prend en paramètre l'adresse d'un vector (STL) via une référence.
Passons à la suite :

Code : C++ - déclaration des variables

```
bool tab_en_ordre = false;
int taille = tab.size();
```

Voilà les seules variables dont nous aurons besoin, mis à part la variable de boucle. Le booléen `tab_en_ordre` permet de tester si au moins un échange a été réalisé lors du dernier passage. C'est cette variable qui va faire tourner la boucle de l'algorithme et qui permettra d'en sortir une fois le tableau trié. Pour la seconde variable, `taille` , elle stocke au départ la taille du vector mais nous en avons besoin car nous allons devoir la modifier à chaque passage du tableau (souvenez-vous du -1 à chaque passage).
A présent, analysons rapidement la fin du code :

Code : C++

```
while(!tab_en_ordre)
{
    tab_en_ordre = true;
    for(int i=0 ; i < taille-1 ; i++)
    {
        if(tab[i] > tab[i+1])
        {
            swap(tab[i],tab[i+1]);
            tab_en_ordre = false;
        }
    }
    taille--;
}
```

Il s'agit là concrètement de la boucle de l'algorithme du tri à bulles. Le **`while`** permet de réitérer les passages sur le tableau tant qu'au moins un échange a été effectué lors du dernier passage ; cette condition est testée avec `tab_en_ordre` comme dit précédemment. La boucle interne **`for`** est la boucle qui compare deux à deux les valeurs adjacentes du tableau. Si deux éléments

sont mal placés, alors on entre dans le `if` et on les inverse grâce à `std::swap` .

Complexité

Le tri à bulles est l'un des tris les plus lents, si ce n'est le plus lent. C'est pour cette raison que son utilisation se fait très rare et cet algorithme reste très critiqué. Mais pourquoi est-il si lent ? On évalue la rapidité d'un algorithme de tri en observant son nombre de comparaisons/échanges et on établi ainsi une échelle que l'on nomme la complexité. Le tri à bulles fait énormément de comparaisons/échanges pour peu de valeurs à trier. On estime mathématiquement qu'il fait en moyenne n(n-1)/4 opérations pour n valeurs à trier.

Avec la notation de Landau, on néglige le -1 et le /4 pour conserver l'opération n^2 qui croît beaucoup plus vite. C'est une croissance quadratique. En ajoutant quelques valeurs supplémentaires à trier, le rapidité de l'algorithme peut donc terriblement chuter. La complexité moyenne du tri à bulles est donc en $O(n^2)$ ce qui est extrêmement lent par rapport aux algorithmes de tri en $O(n*\log_2(n))$ tel le tri fusion.

Dans le pire des cas, la complexité du tri à bulles est aussi en $O(n^2)$. Le pire des cas est, pour ce tri, une liste triée en sens inverse. Le meilleur des cas est une liste triée. Si l'algorithme est confronté à ce cas, sa complexité s'améliore en O(n), ce qui signifie qu'il ne fait que n comparaisons pour n valeurs à trier (en réalité, pour des raisons d'intervalles, il n'en fait que n-1).

Conclusion

Le tri à bulles est l'un des plus mauvais tris en raison de sa forte complexité qui entraîne un temps d'exécution trop long. Et pourtant, cette méthode de tri ne cesse d'être utilisée et ce, sûrement, pour son originalité. Mais il faut tout de même savoir qu'elle est relativement suffisante pour des listes de petites tailles sur un ordinateur, je dis bien relativement. À partir de vingt éléments, il est déjà préférable d'utiliser un autre tri plus adapté. Il faut choisir son algorithme en fonction de ses besoins. Peut-être, un jour, aurez-vous besoin du tri à bulles.

En savoir plus

Ce tutoriel vous a intéressé ? Alors n'hésitez pas à lire cette sous-partie qui fait office d'annexe.

Le tri à bulles bidirectionnel

L'une des variantes les plus connues du tri à bulles est sans doute le tri bidirectionnel ou tri Boustrophedon. D'après Wikipédia, *"on qualifie de boustrophédon le tracé d'un système d'écriture qui change alternativement de sens ligne après ligne, à la manière du bœuf marquant les sillons dans les champs, de droite à gauche puis de gauche à droite"*. Le principe du tri à bulles bidirectionnel est justement de parcourir la liste à trier dans un sens puis dans l'autre donc en alternant le sens de passage à chaque tour. Cette technique augmente légèrement la rapidité de l'algorithme car certains éléments de la liste à trier se trouvent encore dans le buffer lors d'un passage, mais la complexité reste en $O(n^2)$.
Voici une implémentation possible du tri à bulles bidirectionnel en C++ que j'ai réalisée :

Code : C++

```
void gauche(vector<int>&,int,int);

void droite(vector<int>& tab, int deb, int fin)
{
    bool tab_en_ordre = true;
    for(int i = deb ; i < fin ; i++)
    {
        if(tab[i] > tab[i+1])
        {
            swap(tab[i],tab[i+1]);
            tab_en_ordre = false;
        }
    }
    if(!tab_en_ordre)
    {
        gauche(tab, deb, fin-1);
    }
}
void gauche(vector<int>& tab, int deb, int fin)
{
    bool tab_en_ordre = true;
    for(int i = fin ; i > deb ; i--)
    {
```

```
            if(tab[i] < tab[i-1])
            {
                swap(tab[i],tab[i-1]);
                tab_en_ordre = false;
            }
        }
        if(!tab_en_ordre)
        {
            droite(tab, deb+1, fin);
        }
    }
    void tri_bulles(vector<int>& tab){
    droite(tab, 0, tab.size()-1);}
```

Il y a certainement de meilleures implémentations mais celle-ci est ici à titre d'exemple.
Note : Cette implémentation utilise la récursivité croisée : "droite" appelle "gauche", "gauche" appelle "droite".

Liens externes

Si le tri à bulles vous intéresse, voici des liens toujours utiles :

- http://fr.wikipedia.org/wiki/Tri_%C3%A0_bulles : article Wikipédia sur le tri à bulles
- http://lwh.free.fr/pages/algo/tri/tri_bulle.htm : article et animation du tri à bulles
- http://wims.unice.fr/wims/fr_U1~algo~simububble.fr.html : Utilisez-vous même le tri à bulles !

Évidement, vous avez toutes les chances de tomber sur des sites intéressants en tapant "tri bulles" ici.
J'espère que ce chapitre vous aura appris des choses sur le monde du tri.
Je vous remercie bien cordialement de m'avoir lu,

bonne continuation et bon tri !

Si vous rencontrez une difficulté quelconque, vous pouvez toujours m'envoyer un MP.

Partager