Auteur externe

Zeste de savoir

Le tri par sélection

12 août 2019

Table des matières

Parmi les nombreux algorithmes de tri existants, celui dont je vais vous parler aujourd'hui a l'avantage d'être un des plus faciles à mettre en œuvre. Même si je l'implémenterai ici avec une liste d'entiers, il fonctionne parfaitement avec n'importe quelle entité que l'on peut comparer (caractères, flottants, structures, etc...).

1. Principe

L'idée est simple : rechercher le plus grand élément (ou le plus petit), le placer en fin de tableau (ou en début), recommencer avec le second plus grand (ou le second plus petit), le placer en avant-dernière position (ou en seconde position) et ainsi de suite jusqu'à avoir parcouru la totalité du tableau.

Pour la suite du tuto ainsi que pour les différentes implémentations que je donnerai, **j'appliquerai l'algorithme en recherchant l'élément le plus grand du tableau**, et non le plus petit.

Cette décision est importante car à chaque fois que je déplacerai un élément en fin de tableau, je serai certain qu'il n'aura plus à être déplacé jusqu'à la fin du tri.

Regardons ensemble ce que donne l'algorithme appliqué à un exemple :

1. Soit le tableau d'entiers suivant :

6	2	8	1	5	3	7	9	4	0

2. L'élément le plus grand se trouve en 7ème position (si on commence à compter à partir de zéro) :

1. Principe

6	**2**	**8**	**1**	**5**	**3**	**7**	***9***	**4**	**0**

3. On échange l'élément le plus grand (en 7ème position) avec le dernier :

6	**2**	**8**	**1**	**5**	**3**	**7**	***0***	**4**	***9***

4.

```
Le dernier élément du tableau est désormais forcément le plus grand.
    On continue donc en considérant le même tableau, en ignorant
    son dernier élément :

+-+-+-+-+-+-+-+-+-+-----+
|6|2|8|1|5|3|7|0|4|**9**|
+-+-+-+-+-+-+-+-+-+-----+

[[attention]]
```

| Toute l'astuce de l'algorithme est là : on ignore volontairement dans la suite du traitement les éléments que l'on a déplacés à la fin du tableau.

1. De même, on repère l'élément le plus grand en ignorant le dernier et on l'échange avec l'avant dernier :

6	**2**	**4**	**1**	**5**	**3**	**7**	**0**	**8**	**9**

2. Et ainsi de suite, en ignorant à chaque fois les éléments déjà triés (en gras).

6	**2**	**4**	**1**	**5**	**3**	**0**	**7**	**8**	**9**

3.

```
+-+-+-+-+-+-----+-----+-----+-----+
|0|2|4|1|5|3|**6**|**7**|**8**|**9**|
+-+-+-+-+-+-----+-----+-----+-----+
```

1.

```
+-+-+-+-+-+-----+-----+-----+-----+-----+
|0|2|4|1|3|**5**|**6**|**7**|**8**|**9**|
+-+-+-+-+-+-----+-----+-----+-----+-----+
```

1.

```
+-+-+-+-----+-----+-----+-----+-----+-----+-----+
|0|2|3|1|**4**|**5**|**6**|**7**|**8**|**9**|
+-+-+-+-----+-----+-----+-----+-----+-----+-----+
```

1.

```
+-+-+-+-----+-----+-----+-----+-----+-----+-----+
|0|2|1|**3**|**4**|**5**|**6**|**7**|**8**|**9**|
+-+-+-+-----+-----+-----+-----+-----+-----+-----+
```

1.

```
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|0|1|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**|
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
```

1.

```
+-+-----+-----+-----+-----+-----+-----+-----+-----+-----+
|0|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**|
+-+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

1. Et on a enfin trié notre tableau !

On sait que notre tableau est trié lorsque le nombre d'éléments non triés est égal à 1.

2. Implémentations

2.1. Implémentation du tri d'un tableau

Maintenant que vous connaissez l'algorithme et que vous avez vu sur un exemple son fonctionnement, nous pouvons passer à son implémentation !

Mais avant cela, on remarque qu'il est possible de décomposer l'algorithme en plusieurs « sous-fonctions », ce qui facilitera notre travail :

— La recherche de l'élément le plus grand ;
— L'échange de deux éléments ;
— La réalisation du tri.

2.1.1. La fonction max()

Le fonctionnement de cette fonction (qui prend en paramètre un tableau et sa taille pour renvoyer l'indice de l'élément le plus grand) est simple : on se contente de parcourir l'intégralité du tableau pour à chaque fois comparer l'élément actuel avec le maximum provisoire. J'ai choisi de ne conserver que l'indice du maximum provisoire, que je définis par défaut comme étant celui de la première valeur du tableau.

Le choix de cette valeur de départ est important ! En effet, si vous définissez directement une valeur telle que 0 et que votre tableau est du type {-6, -3, -2, -18}, votre algorithme renverra un maximum erroné !

```
1  /**
2  *   Renvoie l'indice du plus grand élément du tableau
3  *
4  *   int tab[] :: tableau dans lequel on effectue la recherche
5  *   int taille :: taille du tableau
6  *
7  *   return int l'indice du plus grand élément
8  **/
9  int max(int tab[], int taille)
10 {
11     // on considère que le plus grand élément est le premier
12     int i=0, indice_max=0;
13
14     while(i < taille)
15     {
16         if(tab[i] > tab[indice_max])
17             indice_max = i;
18         i++;
19     }
20
21     return indice_max;
22 }
```

2.1.2. La fonction echanger()

Le but ici est d'échanger deux éléments (dont on connait les indices) d'un tableau. On agit de la même manière que lorsqu'on souhaite échanger le contenu de deux verres d'eau : on prend un troisième verre pour stocker temporairement un des contenus à échanger (l'image peut paraitre futile ou puérile, mais c'est exactement le comportement que reproduit cette petite fonction).

```c
/**
*   Échange deux éléments d'un tableau
*
*   int tab[] :: tableau dans lequel on effectue l'échange
*   int x :: indice du premier élément
*   int y :: indice du second élément
*
*   return void
**/
void echanger(int tab[], int x, int y)
{
    int tmp;

    tmp = tab[x];
    tab[x] = tab[y];
    tab[y] = tmp;
}
```

2.1.3. La fonction tri_selection()

Petit exo du jour, bonjour ! (Eh oui, je ne vais quand même pas tout faire ... si ?) Aujourd'hui et de manière totalement inopinée, je vais vous demander d'implémenter un algorithme qui vous est totalement inconnu ! Il est le suivant :

— Tant que la taille du tableau est supérieure à 0 : - Rechercher l'indice de l'élément le plus grand ;
  — Échanger cet élément avec le dernier du tableau ;
  — Décrémenter la taille.

Car oui, implémenter l'algorithme de tri par sélection n'est pas plus compliqué que cela. La preuve, même vous, débutant, allez y parvenir !

Contenu masqué n°1

J'ai aussi codé une version récursive de ce tri (qui me parrait plus "naturelle", mais ne diffère en rien ou presque de la version itérative) :

Contenu masqué n°2

Vous noterez que dans les deux versions du tri (récursive ou pas), aucune optimisation n'a été apportée. Je ne vérifie par exemple pas si j'ai effectivement besoin de réaliser l'échange (si max(...) == taille-1, pas besoin d'échanger quoi que ce soit) ... je laisse cela à votre charge ! =)

Pour vous entrainer, essayez de coder le tri par sélection en recherchant non plus l'élément le plus grand, mais l'élément le plus petit !

2.2. Implémentation du tri d'une liste

Eh oui, bien que je vous parle depuis le début du tutoriel du « cas particulier » des tableaux, il faut aussi savoir cet algorithme fonctionne parfaitement sur d'autres structures de données, dont les listes !

Cependant, bluestorm ayant déjà traité cette partie du sujet dans son tutoriel sur l'algorithmique , je me contenterai de vous rediriger vers ce dernier (deux implémentations sont proposées : une en OCaml et l'autre en C).

3. Complexité

Vous l'aurez remarqué, le tri par sélection, à l'opposé du tri à bulles, effectue beaucoup de comparaisons de deux éléments et relativement peu d'échanges. On privilégie donc cette méthode lorsque la comparaison est peu coûteuse en ressources mais que l'échange ne l'est pas.

3.0.1. Calcul (grossier) de la complexité

Minute minute ! La complexité, qu'est-ce que c'est ?

Si vous vous posez cette question, je vous invite à lire le tutoriel de bluestorm sur l'Algorithmique pour l'apprenti programmeur , et plus précisément la partie sur la notion de complexité . De même, si vous n'êtes pas très à l'aise avec cette notion ou que les formules mathématiques vous donnent des boutons, je vous recommande de lire son paragraphe sur la complexité du tri par sélection !

Tentons de raisonner ... À la première itération, on effectue $n-1$ comparaisons. À la $i^{\text{ème}}$ itération, on effectue donc $n-i$ comparaisons (puisque à chaque itération on décrémente la taille du tableau). Le nombre total de comparaisons pour trier un tableau de taille n est donc la somme de $n-i$ pour i allant de 1 à $n-1$, soit en langage mathématique : $\sum_{i=1}^{n-1}(n-i) = \frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$

On s'aperçoit donc que la complexité (en comparaisons) de notre algorithme est quadratique (en $O(n^2)$), ce qui n'est pas très bon. Pour faire simple et être plus concret, à titre d'exemple, si vous doublez la taille d'un tableau, il vous faudra quatre fois plus de temps pour le trier.

En effet, la simplicité de cet algorithme fait qu'on le qualifie d'algorithme « naïf ». Cela ne veut pas pour autant dire qu'il est incorrect, il est juste trop simpliste pour être réellement efficace

(jetez un œil du côté de l'algorithme de tri rapide, ou quicksort, vous verrez que ce n'est pas la même simplicité d'implémentation).

En résumé, lorsque on utilise le tri par sélection :

— On effectue environ $\frac{n(n-1)}{2}$ comparaisons ;
— On effectue environ n échanges ;
— La complexité moyenne et dans le pire des cas est quadratique.

Contenu masqué

Contenu masqué n°1

```c
/**
*   Trie le tableau donné selon l'algorithme de tri par sélection
*
*   int tab[] :: tableau à trier
*   int taille :: taille du tableau
*
*   return void
**/
void tri_selection(int tab[], int taille)
{
    int indice_max;

    // à chaque tour de boucle, on va déplacer le plus grand élément
    // vers la fin du tableau, on diminue donc à chaque fois sa taille
    // car le dernier élément est obligatoirement correctement
    // placé (et n'a donc plus besoin d'être parcouru/déplacé)

    for(; taille > 1 ; taille--) // tant qu'il reste des éléments non triés
    {
        indice_max = max(tab, taille);

        echanger(tab, taille-1, indice_max); // on échange le dernier élément avec le plus grand
    }
}
```

Retourner au texte.

Contenu masqué

Contenu masqué n°2

```c
/**
*   Trie le tableau donné selon l'algorithme de tri par sélection
*
*   VERSION RÉCURSIVE
*
*   int tab[] :: tableau à trier
*   int taille :: taille du tableau
*
*   return void
**/
void tri_selection_recursif(int tab[], int taille)
{
    // un tableau d'un seul élément ou moins n'a pas besoin d'être trié
    if(taille <= 1)
        return;

    echanger(tab, taille-1, max(tab, taille)); // on échange le dernier élément avec le plus grand

    // on rappelle la fonction en diminuant la taille du tableau
    // on peut faire cela car on est certain que le dernier élément
    // est le plus grand (donc plus besoin de le déplacer)
    return tri_selection_recursif(tab, taille-1);
}
```

Retourner au texte.