Tris

Algorithmique

NSI1

1er mars 2022

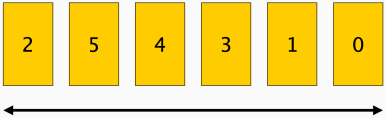
Introduction

Dans ce document nous allons

- étudier quelques tris simples ;
- déterminer leur complexité.

Tri par sélection

Illustration



Principe

On dispose d'une liste `lst`.
On trouve le plus petit élément de `lst`.
On l'échange avec le premier élément de `lst`.
Le premier élément de `lst` est en place et ne bougera plus, on recommence avec les éléments restants (du deuxième au dernier).
Quand on a bien placé l'avant dernier élément, on s'arrête car tous les éléments sont rangés.

Algorithme

On peut l'écrire ainsi :

```
fonction tri_selection(lst)
    n = longueur(lst)
    pour i allant de 0 à n - 2
        p = indice_minimum(lst,i)
        échanger lst[p] et lst[i]
```

Où `indice_minimum` est une fonction qui

- en entrée prend une liste et un entier qui est un indice de cette liste ;
- renvoie l'indice du plus petit élément de la liste situé à partir de l'indice i.

Fonction `indice_minimum`

```
fonction indice_minimum(lst,i)
    n = longueur(lst)
    mini = lst[i]
    i_mini = i
    pour j allant de i à n - 1
        si lst[j] < mini
            mini = lst[j]
            i_mini = j
    renvoyer i_mini
```

Complexité

On décide que les OPEL ne sont que les accès en lecture / écriture aux éléments de la liste.
On considère la complexité dans le pire des cas.

```
fonction indice_minimum(lst,i)
    n = longueur(lst)
    mini = lst[i]                    # 1 opel
    i_mini = i
    pour j allant de i à n - 1
        si lst[j] < mini             # 1 opel
            mini = lst[j]            # 1 opel
            i_mini = j
    renvoyer i_mini
```

Puisque la boucle est appelée n - 1 - i fois, la complexité de cette fonction est $2(n-1-i)+1$

Complexité

```
fonction tri_selection(lst)
    n = longueur(lst)
    pour i allant de 0 à n - 2
        p = indice_minimum(lst,i)   # 2(n-1-i)+1 opels
        échanger lst[p] et lst[i]   # 2 opels
```

Ce qui nous donne une complexité de

$$\sum_{0}^{n-2} 2(n-1-i)+3$$

C'est-à-dire

$$\sum_{0}^{n-2} 2(n-i)+1$$

Calcul à ne pas noter

$$\begin{aligned}
\textstyle\sum_0^{n-2} 2(n-i)+1 &= 2n(n-1) - 2\left(\textstyle\sum_0^{n-2} i\right) + (n-1) \\
&= 2n(n-1) - (n-2)(n-1) + (n-1) \\
&= (2n - n + 2 + 1)(n-1) \\
&= (n+3)(n-1) \\
&= n^2 + 2n - 3
\end{aligned}$$

Donc quand n est grand, seul n^2 prévaut et on dira que la complexité du tri par sélection est en n^2.

Tri par insertion

Principe

On considère une liste (d'entiers ou d'éléments de tout type triable) `lst`.

- la liste composée de `lst[0]` est (évidemment) triée ;
- on regarde si `lst[1]` est plus petit que `lst[0]`, si c'est le cas échange ces deux éléments ;
- on peut donc dire que la liste composée des élements `lst[0]` et `lst[1]` est triée.
- on continue ainsi : à chaque étape, la liste composée des i premiers éléments de `lst` est déjà triée, on va alors « insérer » `lst[i]` dans cette sous-liste, et ainsi on aboutira à une liste des i+1 éléments triée ;
- concrètement pour insérer `lst[i]` au bon endroit, on le compare avec l'élément précédent : tant qu'il est plus petit on l'échange avec celui-ci.

Algorithme

On peut déjà produire la fonction `place` qui

- prend en entrée une liste `lst` et un entier `i` avec les *préconditions* suivantes
  - i est inférieur à la longueur de la liste ;
  - les i premiers éléments de `lst` sont déjà triés dans l'ordre croissant ;
- ne renvoie rien mais place le i^{e} élément à la bonne place pour que les i+1 premiers éléments de `lst` soient triés.

Algorithme de la fonction `place`

```
fonction place(lst : liste, i : entier):
    tant que i > 0 et que lst[i-1] > lst[i]:
        echange lst[i] et lst[i-1]
        i = i - 1
```

Algorithme de tri par insertion

```
fonction tri_insertion( lst : liste):
    n = longueur(lst)
    pour i allant de 1 à n-1:
        place(lst,i)
```

Complexité

Tout comme le tri par sélection, on peut montrer que la complexité temporelle pire cas du tri par insertion est $2n^2 - 2n$, et donc de l'ordre de n^2.

Tri à bulles

Principe

On dispose toujours d'une liste `lst` d'éléments que l'on peut comparer.

- on parcourt la liste du premier élément à l'avant-dernier : si cet élément est plus grand que le suivant, on les échange ;
- tant qu'on a fait au moins un échange, on recommence.

Algorithme

Il est à trouver par toi-même.

Complexité

Elle est encore de l'ordre de n^2, (presque proportionnelle à n^2 quand n est grand) mais on peut baisser le « facteur de proportionnalité » en optimisant l'algorithme (voir activités).