Chapitre 15 :
Algorithmes de tri 2

I Rappels

Un algorithme de tri est un algorithme qui trie des données selon un ordre donné, l'ordre par défaut étant l'ordre croissant.

Les propriétés des algorithmes de tri sont les suivantes :

- Un algorithme de tri est dit comparatif s'il compare deux à deux certains éléments et modifie les données en fonction de l'élément le plus grand.
- Un algorithme de tri est dit stable si, lorsque deux éléments sont égaux, l'algorithme ne modifie pas leur ordre.
- Un algorithme de tri est dit en place si la mémoire utilisée est constante, autrement dit si on ne crée pas de copie des données à trier. Ainsi, dans un algorithme en place, les données initiales sont perdues car elles ont été modifiées par l'algorithme, il y a donc un effet de bord.

II Exemples d'algorithmes de tri et calcul de leur complexité

2.1 Tri à bulles

L'algorithme du tri à bulles est simple mais assez coûteux.

Le principe de ce tri est de faire remonter les éléments les plus grands à la fin de la liste, comme des bulles qui remontent à la surface d'un liquide.

Pour cela, on va comparer les éléments 2 à 2 et les échanger si besoin.

On considère la liste l=[1,7,9,5]. Cette liste est de longueur 4.

- On considère les 4 premiers éléments de la liste.
  - [①,⑦,9,5] comme $1 \leq 7$, on ne fait rien,
  - [1,⑦,⑨,5] comme $7 \leq 9$, on ne fait rien,
  - [1,7,⑨,⑤] comme $9 > 5$, on échange.
- On obtient [1,7,5,9]. Le dernier élément est trié, on ne considère que les 3 premiers éléments de la liste.
  - [①,⑦,5,9] comme $1 \leq 7$, on ne fait rien,
  - [1,⑦,⑤,9] comme $7 > 5$, on échange.
- On obtient [1,5,7,9]. Les 2 derniers éléments sont triés, on ne considère que les 2 premiers éléments de la liste.
  - [①,⑤,7,9] comme $1 \leq 5$, on ne fait rien.

  On a obtenu la liste triée qui est : [1,5,7,9].

L'algorithme du tri à bulles est :

```
def TriBulles(l) :
    n=len(l)
    for i in range(n-1) :
        for j in range(0,n-i-1) :
            if l[j]>l[j+1] :
                l[j],l[j+1]=l[j+1],l[j]
    return l
```

La fonction TriBulles est comparative, stable et en place.

On effectue :

- 1 affectation,
- une boucle for avec $i \in [\![0, n-2]\!]$ avec, à chaque itération, 1 boucle for de longueur $n-i-1$ avec 1 comparaison et 1 affectation,
- 1 renvoi.

Le nombre total d'opérations est donc :

$$1 + \sum_{i=0}^{n-2} (n-i-1) + 1 = 2 + \sum_{k=1}^{n-1} k = 2 + \frac{n(n-1)}{2} = O(n^2).$$

La complexité est donc $O(n^2)$, c'est-à-dire quadratique.

2.2 Tri par sélection

Le principe du tri par sélection est le suivant :

- on cherche le plus petit élément de la liste,
- on le place en première position en effectuant un échange,
- on cherche le plus petit élément de liste à partir du deuxième élément,
- on le place en première position en effectuant un échange,
- on réitère le procédé.

Par exemple, pour $l = [1,5,7,4,2,6]$:

- le plus petit élément de la liste est 1, il est déjà en première position mais on l'échange avec lui même, on obtient : $l = [1,5,7,4,2,6]$,
- le plus petit élément de la liste à partir du rang 2 est 2, on l'échange avec le deuxième élément, on obtient : $l = [1,2,7,4,5,6]$,
- le plus petit élément de la liste à partir du rang 3 est 4, on l'échange avec le troisième élément, on obtient : $l = [1,2,4,7,5,6]$,
- le plus petit élément de la liste à partir du rang 4 est 5, on l'échange avec le quatrième élément, on obtient : $l = [1,2,4,5,7,6]$,
- le plus petit élément de la liste à partir du rang 5 est 6, on l'échange avec le cinquième élément, on obtient : $l = [1,2,4,5,6,7]$,
- la liste est triée.

L'algorithme du tri par sélection est :

```
def TriSelection(l) :
    n=len(l)
    for i in range(n) :
        imin=i #recherche du minimum à partir du rang i
        m=l[i]
        for j in range(i+1,n) :
            if l[j]<m :
                imin=j
                m=l[j]
        l[imin],l[i]=l[i],l[imin] # on place le minimum en position i
    return l
```

La fonction TriSelection est comparative, instable et en place.

On effectue :

- 1 affectation,
- une boucle for avec $i \in [\![0, n-1]\!]$ avec, à chaque itération : 2 affectations, 1 boucle for de longueur $n-i-1$ avec 1 comparaison et au plus 2 affectations, puis 1 échange,
- 1 renvoi.

Le nombre total d'opérations est donc :

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

La complexité est donc $O(n^2)$, c'est-à-dire quadratique.

2.3 Tri par insertion

Le principe du tri par insertion est le suivant :

- on suppose que les éléments indexés de 0 à i-1 sont triés,
- on considère l[i] comme étant une clé que l'on va insérer parmi les éléments déjà triés,
- on compare la clé aux éléments déjà triés en partant du plus grand,
- on décale d'un rang vers la droite les éléments plus grand que la clé,
- on place la clé lorsqu'on arrive à un élément plus petit que la clé.

Par exemple, pour $l = [1,5,7,4,2,6]$ et $i = 3$, la clé est $l[i] = 4$ et :

- on décale 7 : $[1,5,7,7,2,6]$,
- on décale 5 : $[1,5,5,7,2,6]$,
- on ne décale pas 1 donc on insère la clé : $[1,4,5,7,2,6]$.

L'algorithme du tri par sélection est :

```
def TriInsertion(l) :
    n=len(l)
    for i in range(n) :
        k=i
        cle=l[i]
        while k>0 and cle<l[k-1] :
            l[k]=l[k-1] # décalage vers la droite des valeurs plus grandes que la clé
            k-=1
        l[k]=cle# insertion de la clé
    return l
```

La fonction TriInsertion est comparative, stable et en place.

On effectue :

- 1 affectation,
- une boucle for avec $i \in [\![0, n-1]\!]$ avec, à chaque itération : 2 affectations, 1 boucle while de longueur au plus i avec 2 comparaisons et 2 affectations, puis 1 affectation,
- 1 renvoi.

Le nombre total d'opérations est donc au plus :

$$1 + \sum_{i=0}^{n-1} (2 + 4i + 1) + 1 = O(n^2).$$

La complexité est donc $O(n^2)$, c'est-à-dire quadratique.

2.4 Tri par comptage

Le tri par comptage ne s'applique qu'aux listes d'entiers naturels dont on connaît un majorant. On considèrera une telle liste l et un majorant m.

On considère la fonction comptage qui prend comme argument l et m et qui renvoie une liste L le longueur $m+1$ tel que, pour tout $k \in [\![0, m]\!]$, L[k] soit égal au nombre d'occurrences de k dans l :

```
def comptage(l,m) :
    L=[0]*(m+1)#initialisation avec une liste de taille m+1
    for k in l :
        L[k]+=1#on a trouvé une occurrence de k
    return L
```

On effectue :

- 1 affectation d'une liste de longueur $m+1$,
- une boucle for de longueur n avec, à chaque itération, 1 incrémentation,
- 1 renvoi.

Le nombre total d'opérations est donc au plus :

$$m+1+n+1 = O(n+m).$$

La complexité de comptage est donc $O(n+m)$.

Le principe du tri par comptage est parcourir la liste L afin de rajouter les éléments qui sont dans l autant de fois qu'ils apparaissent dans l.

L'algorithme du tri par comptage est :

```
def TriComptage(l,m) :
    L=comptage(l,m)
    T=[]
    for k in range(m+1) :
        T=T+[k]*L[k]#on ajoute L[k] fois l'élément k
    return T
```

La fonction TriComptage est non comparative et non en place.

On effectue :

- 1 appel à comptage de complexité $O(n+m)$,
- 1 affectation de liste vide,
- une boucle for avec pour $k \in [\![0, m]\!]$, à chaque itération, une concaténation de $L[k]$ éléments,
- 1 renvoi.

Le nombre total d'opérations est donc au plus :

$$O(n+m)+1+\sum_{k=0}^{m} L[k] = O(n+m)+1+n = O(n+m).$$

La complexité est donc $O(n+m)$.