MPSI - Informatique Tronc Commun
Cours de la semaine 6 - Les algorithmes de tri

On rappelle le problème du tri :

> Entrée : Une liste l.
> Sortie : Une permutation de l triée par ordre croissant.

Un *algorithme de tri* est un algorithme résolvant le problème du tri.

1 Les questions à se poser

Pour un algorithme de tri, en plus de se poser les questions usuelles, on se pose aussi les question suivantes :

1. Est-ce que toutes les décisions de l'algorithme dépendent uniquement de *comparaisons* entre éléments de la liste ? On parle alors de tri **par comparaison** ou alternativement de **comparatif**.
2. Si plusieurs éléments ont la même valeur dans la liste en entrée, est-ce qu'ils apparaissent, dans la liste en sortie, dans le même ordre que dans la liste en entrée ? On parle alors de tri **stable**.
3. Est-ce que l'algorithme de tri a besoin de créer une nouvelle liste, ou est-ce qu'il peut agir en modifiant **en place** la liste en entrée ?

Se poser la question de la stabilité d'un tri a du sens surtout si on trie les données par rapport à seulement *une partie* de ces caractéristiques.

Exemple 1 (stabilité d'un tri) Supposons qu'on est en cours de EPS, et qu'on a une liste des temps (en minutes) que différents élèves ont mis à courir 600 mètres. Il s'agit d'une liste de paires (*prénom*, *temps mis*), qu'on veut trier par ordre croissant des temps :

```
[('Dominique', 4), ('Camille', 2), ('Gaby', 4), ('Anaël',3)]
```

Un tri **stable** renverra la liste

```
[('Camille', 2), ('Anaël',3), ('Dominique', 4), ('Gaby', 4)]
```

dans laquelle Dominique et Gaby, qui ont fait le même temps, apparaissent dans le même ordre que dans la liste en entrée.

Inversement, si un algorithme de tri renvoie

```
[('Camille', 2), ('Anaël',3), ('Gaby', 4), ('Dominique', 4)]
```

ce n'est **pas un algorithme de tri stable** car il a inversé l'ordre de Dominique et Gaby. Ce qui n'empêche pas que ce soit un algorithme de tri correct, le résultat étant en effet trié.

2 Étude du tri comptage

Si on suppose que tous les élèves ont mis un temps compris entre 0 et 10 minutes à courir 600 mètres, on peut utiliser la fonction suivante, inspirée du tri comptage vu au TD4 :

Programme 1 – Tri comptage

```python
def tri_comptage(liste):
    """
    Entrée
    ------
      Une liste de paires (prénom, temps)

    Sortie
    ------
      Une permutation de la liste en entrée,
      triée par ordre croissant des temps.

     >>> tri_comptage([('D',  4), ('C',  2), ('G',  4), ('A', 3)])
     [('C', 2), ('A', 3), ('D', 4), ('G',  4)]
    """
    eleves_par_temps = [ [] for temps in range(11) ]
    # Pour chaque temps, la liste eleves_par_temps[temps]
    # contiendra la liste des élèves ayant fait ce temps-là.

    # On commence par identifier les élèves ayant fait
    # chacun des temps possibles
    for (prenom, temps) in liste:
        eleves_par_temps[temps].append(prenom)

    # Puis on crée la liste à rendre
    resultat = []
    for temps in range(11) :
        for prenom in eleves_par_temps[temps] :
            resultat.append( (prenom, temps) )
    return resultat
```

On va se poser les questions usuelles sur ce programme :

1. Est-ce qu'il termine ?
2. Est-ce qu'il est correct ? (Rapidement)
3. Quelle est sa complexité ?

mais aussi les questions spécifiques aux algorithmes de tri :

4. Est-ce un **tri par comparaison** ?
5. Est-ce un **tri stable** ?
6. Est-ce un **tri en place** ?

3 Étude du tri à bulles

Voici une variante du tri à bulles (vu au TD6) adaptée aux listes de paires `(prenom, temps)`.

Programme 2 – Tri à bulles

```
1  def est_triee(liste):
2      """
3      Entrée
4      ------
5        Une liste de paires (prenom, temps).
6
7      Sortie
8      ------
9        True si la liste en entrée est triée par temps croissants
10       False sinon
11
12     >>> est_triee([(’D’, 4), (’C’, 2), (’G’, 4), (’A’,3)])
13     False
14     >>> est_triee([(’C’, 2), (’A’, 3), (’D’, 4), (’G’, 4)])
15     True
16     """
17     for i in range(len(liste)-1) :
18         if liste[i][1] > liste[i+1][1] :
19             return False
20     return True
21
22 def tri_a_bulles(liste):
23     """
24     Entrée
25     ------
26       Une liste de paires (prénom, temps)
27     Sortie
28     ------
29       Une permutation de la liste en entrée,
30       triée par ordre croissant des temps.
31
32     >>> tri_a_bulles([(’D’, 4), (’C’, 2), (’G’, 4), (’A’,3)])
33     [(’C’, 2), (’A’, 3), (’D’, 4), (’G’, 4)]
34     """
35     while not est_triee(liste) :
36         for i in range(len(liste) - 1) :
37             if liste[i][1] > liste[i+1][1] :
38                 intermediaire = liste[i]
39                 liste[i] = liste[i+1]
40                 liste[i+1] = intermediaire
41     return liste
```

Commençons par les questions spécifiques au tri :

1. Est-ce un **tri par comparaison** ?
2. Est-ce un **tri stable** ?
3. Est-ce un **tri en place** ?

3.1 Correction partielle versus correction totale

La notion de correction vue la semaine dernière peut être raffinée en deux notion plus précises :

Définition 1 : Correction partielle Pour un programme, on parle de correction partielle à condition que

> Si le programme termine sur une entrée donnée, alors il renvoi un résultat correct pour cette entrée.

Définition 2 : Correction totale Pour un programme, on parle de correction totale lorsque

> Pour toutes les entrées possibles, le programme termine et renvoie un résultat correct.

Dit autrement,

$$\text{correction partielle} + \text{terminaison} = \text{correction totale}$$

Question 4 : Pourquoi est-ce que notre tri à bulles est partiellement correct ? (facile).

Question 5 : Dans notre tri à bulles, pourquoi est-ce que la boucle **`while`** (lignes 35 à 40) termine ?

Variant du tri à bulles On propose le variant suivant, où les barres verticales dénotent le cardinal de l'ensemble

$$\text{variant}(liste) = \left|\left\{(i,j) \in [\![0, \text{len}(liste)-1]\!]^2 \mid i < j \wedge liste[i][1] > liste[j][1]\right\}\right|$$

Dit autrement, notre variant est le nombre de *paires d'indices dans la liste* tels que les éléments à ces indices-là sont mal ordonnés entre eux. Pour montrer que c'est un variant, on va *montrer qu'il est majoré* et *montrer qu'il croît strictement à chaque itération de la boucle* (attention : on parle de la boucle **`while`** des lignes 35 à 40).

Argument 1 : Si on appelle n la taille de la liste, on a $\text{variant}(liste) \leqslant \frac{(n-1)n}{2}$

Argument 2 : À chaque exécution de la boucle **`while`** des lignes 35 à 40, il existe au moins un indice i tel que la condition du **`if`** de la ligne 37 est vraie.

Argument 3 : Quand les lignes 38 à 40 s'exécutent (branche **alors** du **if**), le variant diminue de 1.

Argument 4 : Le variant diminue strictement à chaque exécution de la boucle **`while`** (lignes 35 à 40).

On peut utiliser le variant qu'on a trouvé pour écrire une variante du tri à bulles dans laquelle la terminaison est plus explicite :

Programme 3 – Tri à bulles, terminaison explicite

```
def tri_a_bulles2(liste):
    n = len(liste)
    for repetition in range((n-1)*n//2) :
        for i in range(len(liste) - 1) :
            if liste[i][1] > liste[i+1][1] :
                intermediaire = liste[i]
                liste[i] = liste[i+1]
                liste[i+1] = intermediaire
    return liste
```

Question 6 : Quelle complexité peut-on justifier pour le programme 3 ?

Question 7 : Quelle propriété invariante peut-on formuler pour la boucle extérieure (lignes 3 à 8) du programme 3 ?

Pour trouver une propriété invariante intéressante, regardons comment évolue la liste *en fin de boucle* lorsqu'on commence avec la liste triée par ordre décroissant suivante :

`[('E', 6), ('D', 5), ('C', 4), ('B', 3), ('A', 2)]` :

Après la première itération :

$[('D',5),('C',4),('B',3),('A',2),('\mathbf{E}',\mathbf{6})]$

Après la deuxième itération :

$[('C',4),('B',3),('A',2),('\mathbf{D}',\mathbf{5}),('\mathbf{E}',\mathbf{6})]$

Après la troisième itération :

$[('B',3),('A',2),('\mathbf{C}',\mathbf{4}),('\mathbf{D}',\mathbf{5}),('\mathbf{E}',\mathbf{6})]$

Après la quatrième itération et toutes les suivantes :

$[('A',2),('\mathbf{B}',\mathbf{3}),('\mathbf{C}',\mathbf{4}),('\mathbf{D}',\mathbf{5}),('\mathbf{E}',\mathbf{6})]$

Propriété invariante du tri à bulles On conjecture la propriété invariante suivante, qu'on ne prouvera pas :

> « Après la i$^{\text{ème}}$ itération de la boucle extérieure, les i derniers éléments de la liste sont les i plus grands éléments de la liste, triés par ordre croissant. »

De cet invariant on peut déduire les deux améliorations suivantes pour notre tri à bulles

Amélioration 1 : La boucle extérieure n'a besoin de s'exécuter qu'au plus $n-1$ fois, car au bout de $n-1$ itérations les $n-1$ plus grands éléments sont en fin de boucle, triés... donc la liste entière est triée.

Amélioration 2 : Quand la boucle intérieure s'exécute pour la i$^{\text{ème}}$ fois, elle peut s'arrêter avant les i derniers éléments, qui sont déjà triés.

On en déduit une version plus performante du programme 3 :

Programme 4 – Tri à bulles, amélioré

```python
def tri_a_bulles3(liste):
    """
    Entrée
    ------
      Une liste de paires (prénom, temps)
    Sortie
    ------
      Une permutation de la liste en entrée,
      triée par ordre croissant des temps.

    >>> tri_a_bulles3([('D', 4), ('C', 2), ('G', 4), ('A',3)])
    [('C', 2), ('A', 3), ('D', 4), ('G', 4)]
    """
    n = len(liste)
    for repetition in range((n-1)) :
        for i in range(n - 1 - repetition) :
            if liste[i][1] > liste[i+1][1] :
                intermediaire = liste[i]
                liste[i] = liste[i+1]
                liste[i+1] = intermediaire
    return liste
```


Question 8 Quelle complexité peut-on justifier pour le programme 4 ?

Origine du nom 'Tri à bulles' Le tri à bulles doit son nom à sa propriété invariante : dans le tri à bulles les plus grands éléments remontent peu à peu vers la fin de la liste, de la même façon que dans les boissons pétillantes les bulles remontent peu à peu à la surface.