Informatique PCSI
Prérequis TP 2 suite : boucles imbriquées

Un algorithme de tri

Introduction

Trier des données, c'est les ranger suivant un ordre défini au préalable. Par exemple avec des données numériques, nous pouvons trier ces données en utilisant l'ordre défini en mathématiques : a est avant b si $b-a$ est positif ($a \leq b$ si $b-a \geq 0$). Si nous trions l'ensemble de nombres $\{3,8,5,2\}$, nous obtenons l'ensemble $\{2,3,5,8\}$ et nous disons que les nombres sont rangés suivant l'ordre croissant. Si nous les rangeons dans l'ordre inverse, nous parlons d'ordre décroissant.

Le tri à bulles

L'algorithme du tri à bulles consiste à parcourir une liste plusieurs fois jusqu'à ce qu'elle soit triée, en comparant à chaque parcours les éléments consécutifs et en procédant à leur échange s'ils sont mal triés. Les étapes sont :

- on parcourt la liste et si deux éléments consécutifs sont rangés dans le désordre, on les échange ;
- si à la fin du parcours au moins un échange a eu lieu, on recommence l'opération ;
- sinon, la liste est triée, on arrête.

Les éléments les plus grands remontent ainsi dans la liste comme des bulles d'air qui remontent à la surface d'un liquide. Commençons par un exemple avec la liste : `[12, 17, 14, 11, 8]`. Les éléments permutés sont en gras.

Parcours 1 : [12, **14**, **17**, 11, 8], puis [12, 14, **11**, **17**, 8], puis [12, 14, 11, **8**, **17**].

Parcours 2 : [12, **11**, **14**, 8, 17], puis [12, 11, **8**, **14**, 17].

Parcours 3 : [**11**, **12**, 8, 14, 17], puis [11, **8**, **12**, 14, 17].

Parcours 4 : [**8**, **11**, 12, 14, 17].

Parcours 5 : [8, 11, 12, 14, 17].

Lors du parcours 5, aucune permutation n'est effectuée. La liste est donc triée.

Si une liste de n éléments est déjà triée, il y a un seul parcours de la liste. Nous en déduisons qu'il y a $n-1$ comparaisons. C'est le cas le plus favorable.

On considère la liste : [n, $n-1$, …, 3, 2, 1].

La valeur n se déplace après chaque comparaison jusqu'à la fin de la liste. Donc le nombre de permutations nécessaires pour amener n à la position correcte est $n-1$.

De manière similaire, pour amener $n-1$ à la bonne place il faut $n-2$ permutations et pour amener $n-2$ à la bonne place il en faut $n-3$.

Le nombre total de permutations nécessaires pour trier une liste de n éléments rangés initialement en ordre décroissant est donc $(n-1)+(n-2)+\ldots+1 = \dfrac{n(n-1)}{2}$. Nous disons que le coût de l'algorithme en fonction de n, ou sa complexité, est quadratique.

Un programme :

```
def tri_bulles(liste):
    for j in range(1, len(liste)):
        trier = True
        for i in range(len(liste)-j):
            if liste[i] > liste[i+1]:
                liste[i], liste[i+1] = liste[i+1], liste[i]
                trier = False
        if trier:
            break
```

Il n'est pas nécessaire de parcourir la liste jusqu'à la fin à chaque parcours. En effet, après le premier parcours, l'élément le plus grand est placé à la dernière place. Après le second parcours, les deux plus grands éléments sont rangés à leur place définitive. Et ainsi de suite.

La fonction ne renvoie. On dit que la liste passée en paramètre est triée en place.

On pourrait remplacer l'instruction `break` par l'instruction `return True` ou simplement `return` qui ne renvoie rien (plus précisément, c'est la valeur `None` qui est renvoyée).

On peut aussi utiliser une boucle `while` :

```
def tri_bulles(liste):
    j = len(liste)-1
    while j > 0:
        tri = True
        for i in range(j):
            if liste[i] > liste[i+1]:
                liste[i], liste[i+1] = liste[i+1], liste[i]
                tri = False
        j = j - 1
        if tri:
            j = 0
```

Nous pouvons remarquer que dans le cas le plus défavorable, il y a autant de permutations de valeurs que de comparaisons. Des tris plus performants seront présentés plus tard dans l'année.

Notion de validité d'un algorithme

- **Terminaison**
  La question est de prouver que l'exécution du programme s'arrête après un nombre fini d'étapes. Il nous faut étudier les deux boucles. La boucle interne est une boucle `for` donc le nombre de passages est déterminé et fini. La boucle externe est une boucle `while`. Les valeurs prises par la variable `j` constituent une suite d'entiers strictement décroissante avec des valeurs allant de $n-1$ à 0, si n est la longueur de la liste. Il y a donc exactement $n-1$ passages dans la boucle `while`. L'expression égale à `j` s'appelle *un variant de boucle*.
- **Correction**
  Il s'agit de prouver que le résultat obtenu à la fin est bien celui attendu.
  On démontre pour cela qu'une propriété est vraie après chaque passage dans la boucle, propriété qu'on appelle *un invariant de boucle* : « pour chaque valeur de `j`, la liste est une permutation de la liste initiale et la liste `liste[j:len(liste)]` est triée ».