NSI - Première

TD - tri insertion

qkzk

2021/06/02

Exercice 1 - Appliquer le tri par insertion

1. Faire tourner l'algorithme du tri par insertion présenté en cours sur le tableau `[5, 7, 3, 1, 9]`
2. Combien de comparaisons sont nécessaires pour trier ce tableau ?
3. Recommencer avec le tableau déjà trié `[1, 3, 5, 7, 9]`.
4. Gagne-t-on quelque chose à partir d'un tableau déjà trié ?

Exercice 2 - Programmer le tri par insertion

1. Traduire en Python l'algorithme du tri par insertion.
2. Programmer cette fonction.
3. On souhaite mesurer de manière empirique le nombre de comparaisons effectuées lors d'un tri par insertion.

Voici la démarche :

1. Avant la boucle principale introduire un compteur : `compteur = 0`
2. Repérer l'étape qui nécessite un comparaison et augmenter le compteur :

   `compteur += 1`
3. Renvoyer le compteur à la fin de la fonction.

Adapter votre fonction.

4. Essayer votre fonction sur différents ordres du tableau : `[1, 3, 5, 7]`
   - Quel ordre conduit au minimum de comparaison ?
   - Quel ordre conduit au maximum de comparaison ?

Exercice 3 - Permutations

Les sorties des instructions suivantes ont été effacées. Écrire les sorties produites.

```
>>> t = ["abe", "be", "dda", "ac", "abe"]
>>> t1 = t.copy()
>>> t1.sort()
>>> t1
>>> t2 = t.copy()
>>> t2.sort(reverse=True)
>>> t2
>>> t3 = t.copy()
>>> sorted(t3, key=len)
>>> t3
```

Exercice 4 - Permutations

On dispose de deux tableaux, mélangés dont on souhaite savoir si l'un est une permutation de l'autre.

Par exemple : `[1, 3, 2]` est une permutation de `[3, 2, 1]` mais `[1, 4, 5]` n'est pas une permutation de `[3, 2, 1]`.

Proposer une fonction python d'une ligne de code qui prend deux tableaux en paramètre et renvoie `True` s'ils sont des permutations l'un de l'autre et `False` sinon.

Exercice 5 - Top k

On dispose d'un tableau contenant les 1000 villes les populations des 1000 principales villes de France dans le désordre.

1. Créer en une ligne de code la liste des 100 premières populations, dans l'ordre décroissant.

Rappel :

```
>>> tableau = ["a", "b", "c", "d"]
>>> tableau[:2]
["a", "b"]
```

2. On utilise un tri par sélection. Modifier l'algorithme pour qu'il n'effectue que $k \times n$ comparaisons au lieu de n^2 et qu'il nous renvoie la liste des k premières villes de France.

Exercice 6 - doublons

La presence de doublon dans une grande liste est un soucis permanent. Si l'on agrège des numéros de téléphone à joindre, on souhaite éviter d'appeler plusieurs fois le même numéro !

Il existe des structures de données qui permettent d'éviter les doublons.

1. Comment utiliser un tri pour repérer les doublons dans une liste de téléphone ?
2. La structure de donnée `set` du langage Python fonctionne comme les ensembles mathématiques : elle ne contient aucun doublon.

```
>>> s = {1, 2, 3, 2}      # l'ensemble 1, 2, 3, le doublon est retiré.
>>> s
{1, 2, 3}
>>> type(s)
<class 'set'>
>>> list(s)
[1, 2, 3]                 # on converti en list
>>> len(s)
3
>>> l = [4, 2, 3, 3, 3, 1]
>>> set(l)
{4, 2, 3, 1}              # on converti un set en list
```

   Expliquer comment retirer tous les doublons d'une liste de numéros de téléphone en une liste de Python.

3. Repartons de la liste des numéros de téléphone. Expliquer comment compter les numéros distincts en une ligne de python.

Attention La méthode précédente est simple, elle tient en une ligne de code ! Pour autant, elle n'est pas forcément *efficace* !

Exercice 7 - le tri à bulle

On considère l'algorithme suivant :

```
for i in range(len(t) - 1):
    for j in range(len(t) - 1, i, -1):
        if t[j] < t[j - 1]:
            t[j], t[j - 1] = t[j - 1], t[j]
```

1. Appliquer cet algorithme à la main sur le tableau `t = [5, 3, 1, 7]`

   Quel résultat obtient-t-on ?

2. Reprendre étape par étape le tri du tableau initial en utilisant un tri par insertion et un tri par sélection.
3. Le tri à bulle est-il similaire au tri par insertion et par sélection ?

Exercice 8 - Les points les plus éloignés

1. Écrire une fonction Python qui prend un **tuple de deux nombres** en paramètres et calcule la distance qui sépare le point qu'ils forment jusqu'à l'origine.

   Exemple :

   ```
   >>> distance( (3, 4) )    # distance entre (3, 4) et l'origine
   5
   ```

2. On dispose d'une liste de points : `t = [ (1, 2), (0, -10), ..., (2, 5)]`

   Écrire, en une ligne, une instruction permettant de les trier par distance à l'origine croissante.