Algorithmes itératifs de tri par comparaison

Compétences

- Manipuler des données de taille importante pour tester différents algorithmes de tri.
- Implémenter les tris : insertion, sélection, permutation.

Kaggle

Sur Kaggle un utilisateur peut :

- Trouver et publier des bases de données.
- Explorer et construire des modèles sur un environnement web adapté.
- Travailler avec d'autres professionnels et passionnés.
- Participer des compétitions pour se challenger sur des sujets innovants.

Nous allons nous servir de Kaggle comme d'une base de données sur les adresses en France. Vous devez commencer par récupérer le fichier *france.csv* sur la page principale OpenAddresses-Europe. Attention ce fichier une fois décompressé est très volumineux $\simeq 2.2Go$ avec 27,381,996 lignes. N'essayer pas de l'ouvrir avec un logiciel de type tableur ou éditeur de texte sous peine de devoir tuer le processus pour reprendre la main.

Les fichiers avec Python

Ouvrir un fichier texte

Pour cela Python dispose de la fonction **open** qui renvoie un objet fichier. Elle prend en paramètre, le chemin (absolu ou relatif) menant au fichier à ouvrir et le mode d'ouverture. Le mode est donné sous la forme d'une chaîne de caractères. Voici les principaux modes :

- **r** en lecture (read)
- **w** en écriture (write)
- **a** en allongement (append)

Dans le mode **w**, le fichier est créé automatiquement s'il n'existe pas.

```
try:
    reader = open('filename.txt', "r")
    #Further file processing goes here
finally:
    reader.close()
```

Le répertoire courant de travail

Pour obtenir le répertoire courant dans lequel vous avez enregistré votre programme Python vous pouvez utiliser la fonction **getcwd**

```
import os   #operating system
os.getcwd() #current working directory
```

La fonction **chdir** du module **os** permet de modifier de manière permanente le répertoire de travail

Lire le contenu d'un fichier texte

Il existe trois modes de lecture des informations

- **read(size=-1)** : lit une certaine quantité de données et la renvoie sous forme de chaîne de caractères.
- **readline(size=-1)** : lit une ligne entière si aucun argument n'est passé et la renvoie sous la forme de chaîne de caractères
- **readlines()** : lit toutes les lignes jusqu'à la fin du fichier, et renvoie le résultat sous forme d'une liste de lignes sous forme de chaîne de caractères

Pour s'entraîner

1. Écrire une fonction `readLines(filename, n)` qui prend en paramètres le nom de fichier, le nombre de lignes à lire et qui renvoie un tableau contenant les **n** premières lignes du fichier.
2. Afficher les trois premières lignes et expliciter les attributs utilisés.
3. Modifier la fonction `readLines` pour qu'elle renvoie un tableau contenant les **n** premières lignes du fichier à **l'exception** de la première.
4. Évaluer expérimentalement l'évolution du temps de chargement en fonction du nombre **n** de lignes demandées. Pour cela vous utiliserez la fonction `runTime` suivante et présenterez vos résultats sous la forme d'un graphique avec `matplotlib`

```
def runTime(f, filename, n):
    "Renvoie le résultat en
     milliseconde"
    debut = time.process_time()
    f(filename, n)
    fin = time.process_time()
    return round((fin - debut)*1e3)
```

5. Combien de temps faudrait-il pour charger dans un tableau la totalité du fichier *france.csv*?

Trier le tableau contenant le fichier

L'objectif est maintenant d'appliquer les algorithmes de tri : sélection, insertion et permutation pour essayer de trier le tableau représentatif de notre fichier en fonction d'un de ses attributs par exemple *NUMBER* ou *POSTCODE*. La fonction `readLines` renvoie un tableau **t** dont chaque élément est une ligne de valeurs des attributs au format *str*. Il est donc difficile d'accéder à ces valeurs individuellement. Pour simplifier l'accessibilité aux valeurs la fonction `transforme` ci-dessous transforme chaque ligne du tableau **t** en un sous tableau dont chaque élément est une valeur d'attribut. Par exemple pour la première ligne nous avons :

`t[1] = "4.9356384,46.1290477,5098,..."`
transformée en :
`t[1] = ['4.9356384','46.1290477','5098',...]`
De cette manière on peut accéder à toutes les valeurs des attributs grâce à la notation `t[i][j]` avec **i** pour les lignes et **j** pour les colonnes.
Par exemple pour obtenir la valeur de l'attribut *NUMBER* sur la ligne d'indice `(i=1)` il suffit d'écrire `t[1][2]`

```
1  def transforme(t, n, delimiter = ","):
2
3      for i in range(n):
4          t[i] = t[i].split(delimiter)
```

1. Écrire une fonction `selectSort2D(t, n, c)` qui prend en argument un tableau **t** de taille **n** et un indice **c** de la colonne correspondant à l'attribut sur lequel on veut faire le tri.
2. Évaluer la complexité expérimentale de votre algorithme en faisant varier le nombre de lignes de votre tableau. Les résultats seront présentés sous forme graphique.
3. Reprendre les deux questions précédentes pour une fonction `insertSort2D(t, n, c)`
4. Quel est le tri le plus efficace pour trouver les 1000 plus petites valeurs de l'attribut *NUMBER* sur la totalité du fichier? Justifier.