TP 1 – Tableaux, algorithmes, complexité
L2 – INF 1407 – Année 2018-2019

Objectifs : ce TP propose quelques rappels du langage de programmation Python. Pour ceux qui n'ont jamais utilisé ce langage, un polycopié du cours de l'année de L1 est accessible sur l'ENT (L1-PIA-x2-2018), ainsi qu'un autre document présentant les principales caractéristiques du langage (IntroductionPython3). Trois exercices permettent d'écrire et de tester des algorithmes utilisant des tableaux (implémentés par des listes). Les solutions de ces exercices devront être déposées dans l'espace prévu à cet effet avant le TP de la semaine prochaine.

Configuration de l'environnement de programmation

Installer Anaconda et spyder – *Spyder* est l'environnement de développement que nous allons utiliser pour écrire nos programmes. Pour une information plus complète, voir la doc en ligne sur le site de l'application à cette adresse : `https://docs.spyder-ide.org/`. Un ensemble Anaconda contenant Python avec Spyder peut être directement téléchargé à l'adresse suivante : `https://www.anaconda.com/download`. La version actuelle est la 3.7.

Mise en place de l'environnement

- Ouvrir un terminal.
- Récupérer le fichier *pythonenv* de la plate-forme moodle et enregistrer le dans votre répertoire principal.
- Taper dans le terminal la ligne suivante :

  ```
  source pythonenv
  ```

- Vérifier que le fichier *pythonenv* est bien dans le répertoire courant du terminal (ls, cd,...).
- Lancer *spyder* dans le terminal :

  ```
  spyder
  ```

- OU lancer **spyder** via l'icône qui est apparue sur votre bureau.

À partir de maintenant, le lancement de *spyder* se fera simplement via l'icône ou en ouvrant un terminal et en lançant directement la commande *spyder* à partir du terminal. Sur la bannière de la fenêtre *Spyder*, il est important de voir le nom Spyder (Python3.6) ou plus 3.7... Si (Python2.7) s'affiche, recommencez la mise en place de l'environnement.

IDE Spyder

La fenêtre Spyder est divisée en plusieurs zones, de haut en bas :

- la barre de menu permet d'avoir accès aux principales commandes,
- la barre d'outils montre les outils les plus courants : Nouveau fichier, Ouvrir un fichier, Exécuter le fichier, ...
- la zone d'édition à gauche contient le programme à éditer, c-à-d. l'endroit où vous allez écrire et modifier vos programmes Python
- la zone en bas à droite (dans le 1/4 inférieur droit) est la console Python. Elle permet de :
  - saisir directement des instructions Python
  - afficher les messages d'erreur (à l'interprétation de votre programme)
  - afficher les traces des exécutions de vos programmes (fonction print)
  - saisir les entrées de votre programme (fonction input)
- l'explorateur de variables dans le quart supérieur droit

Exercice 1

Écrivez les fonctions suivantes en Python :

1. *double_tab(t)* qui renvoie un tableau contenant le double de chaque valeur du tableau passé en argument ;
2. Définissez les tableaux de taille 100 suivants, qui pourront servir à tester les différents algorithmes que vous écrirez par la suite :
   - t_1 : les entiers de 0 à 99 ;
   - t_2 : les entiers de 99 à 0 ;
   - t_3 : une permutation aléatoire des entiers de 0 à 99.

   Pour créer le tableau t_3, il faut importer le module *random* et utiliser la fonction *randint* :

```
import random
k = random.randint(a,b) #renvoie un entier k compris entre a et b (inclus)
```

Exercice 2

1. Écrivez les fonctions *max(tab)* et *min(tab)* qui renvoient respectivement l'élément maximum et minimum d'un tableau d'entiers *tab*.
2. Combien de comparaisons les fonctions *max* et *min* effectuent-elles en fonction de la taille n du tableau en entrée ?
3. En utilisant les fonctions précédentes, écrivez une fonction *min_max(tab)* qui renvoie le couple *min, max* du plus petit et plus grand élément de *tab* (ça tient en deux lignes). Comptez le nombre de comparaisons effectuées par la fonction.
4. Améliorez la fonction *min_max(tab)* pour qu'elle ne fasse pas plus de 150 comparaisons sur les tableaux à 100 cases. Vous prendrez en considération les éléments suivants :

- on parcourt les éléments du tableau deux par deux ;
- on compare les deux éléments entre eux ;
- on compare le plus grand au maximum courant et le plus petit au minimum courant

Vérifiez le nombre de comparaisons effectuées. Vous ferez attention à la taille du tableau (paire ou impaire).

Exercice 3

Tri lexicographique – L'objectif ici est d'écrire un programme qui trie une liste de mots et les range dans l'ordre lexicographique (ordre des dictionnaires). Pour simplifier, vous n'utiliserez pas les caractères accentués. Vous écrirez $m_1 < m_2$ si le mot m_1 se place avant le mot m_2 suivant l'ordre lexicographique, et $m_1 = m_2$ si les deux mots sont identiques.

1. Écrivez la définition d'une variable alphabet de type **str** :
   "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz"

2. Ecrivez une fonction *ordre_ alphabetique(c1,c2)* qui prend en argument deux caractères c_1 et c_2 et renvoie 1 si c_1 est avant c_2, 2 si c_2 est avant c_1 et 0 si c_1 et c_2 sont identiques.

3. En utilisant la fonction *ordre_ alphabetique*, écrivez une fonction *ordre_ lexico(m1,m2)* qui prend en argument deux mots m_1 et m_2 et renvoie 1 si $m_1 < m_2$, 2 si $m_2 > m_1$ et 0 si $m_1 = m_2$.

4. Ecrivez une fonction *tri_ lexico* qui prend en argument un tableau de mots et renvoie le tableau trié suivant l'ordre lexicographique. Vous utiliserez la fonction *ordre_ lexico(m1,m2)* et l'un des algorithmes simples de tri vus en cours l'année dernière (par exemple le tri bulle).

Exercice à chercher à la maison

Exercice 4

Tri par comptage – Un tableau t contient n entiers naturels compris entre 0 et M, bornes comprises, avec $M \leq n$. Pour trier ces données, nous appliquons l'algorithme suivant :

- On crée un tableau *compte* formé de $M+1$ éléments tous nuls ;
- On parcourt ensuite le tableau t et pour chaque entier p rencontré on augmente d'une unité l'élément d'indice p du tableau *compte* ;
- On parcourt le tableau *compte* et pour chaque élément k d'indice i, on ajoute le nombre i, k fois, dans une liste initialement vide qui est retournée à la fin (ou dans le tableau t en remplacement des éléments initiaux).

Exemple : soit le tableau T = [5,7,1,0,1,2,3,3,9,2,4,2,9,1,2,7,2,9,9,3] comportant 20 éléments de 0 à 9 (n = 20 et M = 9). Le tableau *compte* = [1,3,5,3,1,1,0,2,0,4]. Ce tableau est constitué du nombre d'occurrences de chaque chiffre de 0 à 9.

1. Écrivez une fonction *tri_compte1*, prenant en arguments un tableau t et un entier M (valeur maximale des entiers à trier), qui trie le tableau t. Dans cette question, le tableau n'est pas modifié et une liste représentant le tableau trié est retournée.

2. Donnez la complexité de cet algorithme en fonction de la longueur n du tableau. Vous considérerez que la fonction *append* a une complexité en O(1).

3. Testez la fonction sur un tableau de 10 000 entiers choisis au hasard entre 0 et 100, bornes comprises. Utilisez la fonction *randint*.

4. Écrivez une nouvelle fonction *tri_compte2*, qui trie le tableau t, sans créer de nouvelle liste (autre que le tableau *compte*), mais en modifiant le tableau t initial.

5. Vous supposerez dans cette question que la valeur de M est inconnue. Écrivez une fonction *tri_compte3*, qui prend comme seul paramètre un tableau t et renvoie le tableau trié, sans créer de nouvelle liste (autre que le tableau *compte*).