TP d'informatique n°6 : Tri à bulles, recherche d'un facteur dans un texte
PCSI 2023 – 2024

Ce TP a pour objectifs de manipuler les listes et tableaux, de commencer à se familiariser avec la notion de tri qui sera de nouveau abordée dans un prochain TP et de manipuler les chaînes de caractère.

I. Une petite mise en bouche

Exercice 1 :

On souhaite créer une fonction **deux_plus_proches** prenant en argument une liste ou un tableau de valeurs et qui renvoie la liste des deux éléments (éventuellement confondus) les deux plus proches (au sens de la différence en valeur absolue).

Par exemple, pour la liste suivante L = [2,5,-1,1], la fonction renverra [2,1].

Aide éventuelle, à ne regarder que si besoin : L'idée est ici de tester tous les *couples*. On pourra commencer par chercher les deux plus loin que les deux plus proches en testant toutes les combinaisons possible parce qu'il y a une petite difficulté en moins (mais pour les deux plus loin, il y aurait un algorithme plus efficace qui serait max-min, mais cela ne nous aiderait pas à adapter ensuite.)

II. Tri à bulles

Le tri à bulles est un algorithme de tri classique. On considère un tableau de nombres à trier. L'algorithme parcourt le tableau et dès que deux éléments consécutifs ne sont pas ordonnés, les échange. L'opération est répétée jusqu'à ce que le tableau soit trié. Les éléments les plus grands remontent ainsi dans la liste comme des bulles d'air remontant à la surface d'un liquide.

Vous pouvez visualiser une animation :

https://lwh-21.github.io/Algos/algorithmes%20de%20tri/bubblesort_fr.html

Exercice 2 :

Q1. Appliquer l'algorithme de tri à bulles « à la main » au tableau suivant : T = [18, 10, 15, 11, 5] en détaillant chaque étape.

Que remarquez-vous après un premier parcours du tableau ?

Q2. Supposons que l'on double la taille de la liste à trier, de façon approchée, le temps de calcul :

(a) ne change pas

(b) double

(c) quadruple

(d) autre (préciser).

Le temps de calcul d'un algorithme s'appelle la complexité en temps, notion qui sera de nouveau abordée au second semestre.

Q3. Avec cet algorithme, le cas le plus défavorable (pire cas) est lorsque le tableau est rangé dans l'ordre décroissant. Prenons l'exemple de la liste suivante : [n, n-1, n-2, n-3…]

(a) Déterminer le nombre de permutations nécessaires pour amener le nombre n à la position correcte.

(b) Déterminer ensuite le nombre de permutations nécessaires pour amener n – 1 à la bonne place ? Et pour le nombre n -2 ?

(c) Que remarquez-vous concernant le temps de calcul dans le pire des cas ?

Exercice 3 :
Écrire une fonction **est_trie** prenant en argument un tableau de valeurs T, qui retourne True ou False selon que le tableau est trié ou non.

Exercice 4 :
Écrire une fonction **tri_a_bulles** qui prend en argument en tableau de valeurs T, qui trie le tableau T par l'algorithme de tri à bulles et renvoie le tableau trié.

III. Recherche d'un facteur dans un texte

Trouver une chaîne de caractères dans un texte est un problème qu'on rencontre fréquemment dans les éditeurs de texte (le fameux Ctrl+F !).

Exercice 5 :
Écrire une fonction **egalite_mot**, qui prend en argument deux chaines de caractères de même longueur et qui renvoie True si les chaines sont les mêmes, False sinon. On codera soi-même cette opération et on s'interdit d'utiliser == sur des chaines de plus de 1 caractère (bien que cela fonctionne en python).

Exercice 6 :
Écrire une fonction **cherche_dans_chaine** prenant en argument deux chaine de caractère c1 et c2 et qui renvoie True si c1 est une sous-chaine de c2 et False sinon. Bien sûr, on n'utilisera pas «if c1 in c2», mais on le codera par soi-même. Par exemple :
cherche_dans_chaine("que","informatique")->True alors que
cherche_dans_chaine("qe","informatique")->False
Remarque : le test avec « in » de python fait déjà cela de façon efficace, mais l'idée est ici de le coder nous même pour comprendre la difficulté (c'est-à-dire le temps d'exécution) de cette opération et nous entrainer !

IV. Applications autour des anagrammes

Une anagramme d'un mot m est un mot obtenu par permutation des lettres de m. Dans une anagramme, on ignore les majuscules/minuscules, les accents, cédilles, apostrophes… Par exemple, les mots « sortie » et « rôties » sont des anagrammes. Par soucis de simplification, les mots utilisés dans les fonctions seront tous en minuscules, sans caractères spéciaux.

Exercice 7 :
Écrire une fonction **mot_to_dico**, qui prend en argument un mot m (type str) et renvoie un dictionnaire dont les clés sont les lettres de m et la valeur correspondant à une clé est le nombre d'occurrences de la lettre dans m.

Exercice 8 :
Déduire de l'exercice précédent une fonction **sont_anagrammes** qui prend en argument deux mots (type str) et qui teste s'ils sont anagrammes l'un de l'autre.

Exercice 9 :
Vous disposez d'un fichier texte d'une liste de mots.
Les lignes de code suivantes vont permettront de charger la liste des mots :

```
def charger_liste_mots(filename : str) -> list:
    f = open('filename', 'r')
    content = f.read().splitlines()
    f.close()
    return content
liste_mots = charger_liste_mots('listemotsfrancais.txt')
```

Écrire une fonction **liste_anagrammes** qui prend en argument un mot et une liste de mots, et renvoie la liste des anagrammes du mot dans la liste. Par exemple, **liste_anagrammes** ('preparation', liste_mots) doit renvoyer ['apparieront', 'preparation', 'appaireront'].

V. A faire pour la prochaine fois

Exercice 10 :
On considère une liste composée de listes de même longueur. Chaque sous-liste représente une ligne d'un tableau, chaque élément de la sous-liste une colonne. Écrire une fonction permettant d'afficher tous les éléments en parcourant par ligne. Par exemple, pour le «tableau» [[1,2,3],[4,5,6]], votre fonction doit afficher 1 2 3 4 5 6.

Exercice 11 :
Reprendre l'exercice précédent mais afficher cette fois-ci les éléments en parcourant par colonne. Cette fois-ci votre fonction affichera 1,4,2,5,3,6.

Exercice 12 :
Améliorer la fonction **tri_a_bulles2** pour que le tableau soit renvoyé dès qu'il est trié. Comparer les temps d'exécution et la complexité avec la fonction de l'exercice 5. Commenter.

Exercice 13 :
Le tri « Shaker » est un tri à bulles bidirectionnel. Son principe est identique à celui du tri à bulles sauf qu'il se fait dans les deux sens : sur un parcours, la plus grande valeur est déplacée vers la fin puis la plus petite valeur vers le début du tableau. Écrire une fonction **tri_shaker** prenant en argument un tableau de valeurs T, qui trie le tableau T et renvoie le tableau trié.

Exercice 14 :
Écrire une fonction permettant de rechercher un facteur dans un texte sans utiliser de fonction intermédiaire (c'est-à-dire la fonction **cherche_dans_chaine** codée à l'exercice 6, mais en n'utilisant pas le 5).