Exercices de programmation des oraux Agroveto

Julien Reichert

Session 2022

Ce document regroupe les exercices que j'ai donnés aux oraux d'Agroveto pour la session 2022.

Des exercices ont été retirés des versions précédentes et d'autres ajoutés (souvent pour remplacer les numéros manquants).

Le nombre d'étoiles est un témoin de difficulté, et le nombre entre parenthèses est le nombre de fois que l'exercice a été posé sur un total de 151 candidats, dont onze qui n'ont pas eu d'exercice de cette liste, mais plutôt du débug sur leur projet ou un exercice unique.

Algorithmes de recherche dans une liste / chaîne de caractères / matrice

* Exercice 1 (0) : Écrire en Python une fonction qui détermine la position d'un minimum local dans une matrice composée uniquement d'entiers de `0` à un `k` en paramètre.

On rappelle qu'un minimum local a tous ses voisins (latéraux, pour ne pas alourdir) supérieurs ou égaux à lui-même. Bien entendu, on peut se contenter de rechercher le minimum global. On notera que sans propriété supplémentaire, la complexité dans le pire des cas est de l'ordre du nombre d'éléments dans la matrice.

Il est possible d'ajouter un bord de nombres « très grands » à la matrice, voire de considérer que ce bord existe.

** Exercice 2 (3) : Écrire en Python une fonction qui détermine si deux listes ont les mêmes éléments, peu importe l'ordre et le nombre d'occurrences.

** Exercice 2 bis (19) : Écrire en Python une fonction qui détermine si deux listes ont les mêmes éléments en même nombre, donc sont des permutations l'une de l'autre.

** Exercice 3 (7) : Écrire en Python une fonction qui retourne la position de départ et la longueur de la plus longue suite de valeurs identiques dans une séquence (chaîne de caractères ou liste).

* Exercice 4 (4) : Écrire en Python une fonction qui détermine l'indice dans une liste de couples d'entiers tel que le rapport entre le deuxième et le premier élément soit le plus petit possible. Le premier entier ne sera jamais nul et le deuxième sera normalement strictement positif et strictement inférieur au premier.

* Exercice 4 bis (6) : Écrire en Python une fonction qui détermine si le premier élément d'une liste de n-uplets (ou de listes... ou une structure à adapter) apparaît dans deux n-uplets (même remarque) consécutifs.

Une autre version de cet exercice revient à compter le nombre de fois que ceci se produit.

** Exercice 5 (30) : Écrire en Python une fonction prenant en entrée une liste de nombres entre 0 et k et qui retourne le nombre le plus fréquent.

*** Exercice 5 bis (6) : Écrire en Python une fonction prenant en entrée une chaîne de caractères (dont on supposera qu'ils sont tous issus de la table ASCII non étendue) et qui retourne le caractère le plus fréquent.

Pour les deux versions de l'exercice 5, le comportement est au choix en cas d'égalité.

**** Exercice 6 (1) : Écrire en Python une fonction qui recherche un motif dans une matrice. Le motif en question est représenté par une matrice contenant éventuellement des None là où l'élément de la matrice où la recherche se fait n'a pas d'importance.

***** Exercice 7 (6) : Écrire en Python une fonction qui détermine si tous les `0` de la matrice en entrée sont ensemble, c'est-à-dire sur une même composante connexe.

Presque systématiquement, la difficulté de l'exercice fait qu'on supposera une fonction adéquate faisant le parcours disponible, souvent parce qu'elle est présente dans le projet.

**** Exercice 7 bis (4) : Écrire en Python une fonction qui détermine si deux éléments (couples d'entiers) d'une liste de listes de couples d'entiers sont voisins, c'est-à-dire correspondent à des points de distance un en norme un.

Une variante de l'exercice restreint la recherche à deux éléments dont les indices ne sont pas consécutifs.

Une autre variante de l'exercice fait faire le test entre deux couples d'entiers issus chacun d'une liste de couples d'entiers, avec extension à une liste de listes de couples d'entiers (y a-t-il deux éléments distincts de cette liste vérifiant la propriété précédente ?).

*** Exercice 7 ter (0) : Écrire en Python une fonction qui détermine si tous les `True` d'une liste de booléens sont côte à côte. [Était l'exercice 4 en 2021]

Algorithmes de calcul sur une liste de nombres

** Exercice 8 (5) : Écrire en Python une fonction qui prend en entrée une liste croissante `L` de nombres et qui retourne l'indice `i` tel que l'écart entre `L[i]` et `L[i+1]` soit le plus petit écart entre deux nombres (forcément successifs) de `L`.

** Exercice 9 (0) : Écrire en Python une fonction qui prend en entrée une matrice carrée `M` de taille `n`, un indice de ligne `i` et un indice de colonne `j` et qui détermine la liste des nombres de `1` à `n` qui ne sont ni dans la ligne `i` ni dans la colonne `j`.

** Exercice 10 (2) : Écrire en Python une fonction qui prend en entrée une liste `L` et qui renvoie la liste `M` dont chaque élément est la moyenne de l'élément à la même position dans `L` et de son (cas des bords) ou ses voisins.

** Exercice 11 (13) : Écrire en Python une fonction qui détermine la position de l'élément d'une liste de couples `(x,y)` le plus proche d'un point donné avec la même syntaxe.

On pourra utiliser la norme 1 ou la norme 2 (respectivement somme des distances en abscisse et en ordonnée ou distance euclidienne), voire la norme infinie (maximum des distances en abscisses ou en ordonnée).

** Exercice 12 (0) : Écrire en Python une fonction qui détermine si une liste de nombres en entrée correspond à une suite géométrique (la valeur de retour doit être un booléen).

*** Exercice 12 bis (0) : Écrire en Python une fonction qui détermine si une liste de nombres en entrée correspond à une suite arithmétique ou géométrique (la valeur de retour doit être une chaîne de caractères répondant aux deux questions).

*** Exercice 13 (1) : Écrire en Python une fonction qui détermine si une liste est triée par ordre croissant ou décroissant (la valeur de retour doit être une chaîne de caractères répondant aux deux questions).

*** Exercice 13 bis (5) : Écrire en Python une fonction qui vérifie si une liste est d'abord croissante puis décroissante.

Algorithmes divers

*** Exercice 14 (12) : Écrire en Python une fonction prenant en entrée une matrice carrée et renvoyant la matrice obtenue en faisant une rotation d'un quart de tour dans le sens horaire de la matrice de départ.

*** Exercice 14 bis (6) : Écrire en Python une fonction prenant en entrée une matrice de dimensions `(2*n, n)` et renvoyant la matrice carrée de dimensions `(2*n, 2*n)` obtenue en disposant la matrice de départ dans un damier. On pourra aussi écrire une fonction réciproque.

*** Exercice 15 (3) : Écrire en Python une fonction qui simule une promenade aléatoire de `n` pas sur une grille de dimensions `(2*L,2*l)` centrée en `(0,0)`, qui est le point de départ. Le programme retourne un code indiquant si on est resté sur la grille ou de quel côté on est sorti (ce qui aurait interrompu la promenade).

**** Exercice 15 bis (2) : Écrire en Python une fonction qui simule une promenade aléatoire de `n` pas sur un graphe donné en tant que liste de listes d'adjacence.

**** Exercice 16 (1) : Écrire en Python une fonction prenant en entrée quatre couples de flottants `a`, `b`, `c` et `d` et déterminant si les segments entre les points de coordonnées `a` et `b` d'une part et `c` et `d` d'autre part se chevauchent.

**** Exercice 17 (4) : Écrire en Python une fonction qui vérifie si un cercle de centre `(x,y)` et de rayon `r` touche un rectangle parmi ceux donnés dans une liste en tant que couples de couples `((xg,yg),(xd,yd))`, où `(xg,yg)` sont les coordonnées du coin en bas à gauche et `(xd,yd)` sont les coordonnées du coin en haut à droite. On pourra faciliter l'exercice en considérant un rectangle et des rectangles, voire un cercle et des cercles.