

Feuille d'exercices 1

Listes et séquences

Exercice 1

1. Ecrire une fonction `moyenne` prenant en paramètre une séquence numérique (liste ou tuple) et retournant sa moyenne.
2. Ecrire une fonction `variance` prenant en paramètre une séquence numérique (liste ou tuple) et retournant sa variance.
On rappelle que $V(X) = E((X - E(X))^2) = E(X^2) - E(X)^2$ et on fera appel à la fonction `moyenne()`.
3. Quelles sont leur complexité (en fonction de la longueur de la séquence) ?
4. Les tester en générant une liste aléatoire à l'aide de la fonction `uniform(a,b)` du module `random` qui retourne un nombre aléatoire $a \leqslant . \leqslant b$ selon une loi uniforme.

Les exercices qui suivent sont des épreuves types d'oraux de concours banque PT

Exercice 2

Recherche d'espace libre contigu.

Soit un entier naturel n non nul et une liste `t` de longueur n dont les termes valent 0 ou 1. Le but de cet exercice est de trouver le nombre maximal de 0 contigus dans `t` (c'est-à-dire figurant dans des cases consécutives). Par exemple, le nombre maximal de zéros contigus de la liste `t1` suivante vaut 4 :

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
t1[i]	0	1	1	1	0	0	0	1	0	1	1	0	0	0	0

1. Écrire une fonction `nombreZeros(t,i)`, prenant en paramètres une liste `t`, de longueur n , et un indice i compris entre 0 et $n - 1$, et renvoyant :

$$\begin{cases} 0, & \text{si } t[i] = 1 \\ \text{le nombre de zéros consécutifs dans } t \text{ à partir de } t[i] \text{ inclus,} & \text{si } t[i] = 0 \end{cases}$$

Par exemple, les appels `nombreZeros(t1,4)`, `nombreZeros(t1,1)` et `nombreZeros(t1,8)` renvoient respectivement les valeurs 3, 0 et 1.

2. Comment obtenir le nombre maximal de zéros contigus d'une liste `t` connaissant la liste des `nombreZeros(t,i)` pour $0 \leqslant i \leqslant n - 1$?
En déduire une fonction `nombreZerosMax(t)`, de paramètre `t`, renvoyant le nombre maximal de 0 contigus d'une liste `t` non vide. On utilisera la fonction `nombreZeros`.
3. Quelle est la complexité de la fonction `nombreZerosMax` construite à la question précédente ?
4. Trouver un moyen simple, toujours en utilisant la fonction `nombreZeros`, d'obtenir un algorithme plus performant.

Exercice 3. Cryptage de César.

Soit n un entier vérifiant $n \leq 26$. On souhaite écrire un programme qui code un mot en décalant chaque lettre de l'alphabet de n lettres.

Par exemple pour $n = 3$, le décalage sera le suivant :

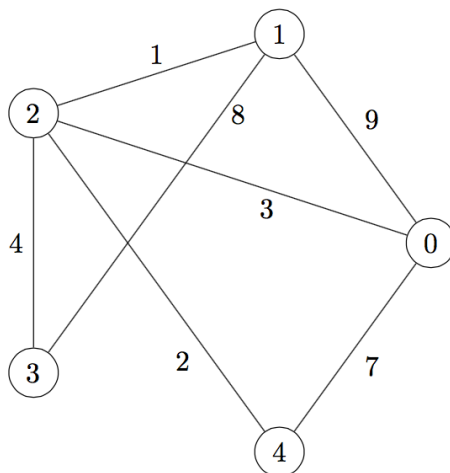
avant décalage	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	...	...	<i>x</i>	<i>y</i>	<i>z</i>
après décalage	<i>d</i>	<i>e</i>	<i>f</i>	<i>g</i>	...	...	<i>a</i>	<i>b</i>	<i>c</i>

Le mot **oralensam** devient ainsi **rudohqvdp**.

1. Définir une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique (caractères en minuscule).
2. Écrire une fonction **decalage**, d'argument un entier n , renvoyant une chaîne de caractères contenant toutes les lettres dans l'ordre alphabétique, décalées de n , comme indiqué ci-dessus.
3. Écrire une fonction **indices**, d'arguments un caractère x et une chaîne de caractères **phrase**, renvoyant une liste contenant les indices de x dans **phrase** si x est une lettre de **phrase** et une liste vide sinon.
4. Écrire une fonction **codage** d'arguments un entier n et une chaîne de caractères **phrase**, renvoyant **phrase** codé avec un décalage de n lettres.
5. Comment peut-on décoder un mot codé ?

Exercice 4. Parcours de graphe.

On considère le graphe G suivant, où le nombre situé sur l'arête joignant deux sommets est leur distance, supposée entière :



1. Construire la matrice $(M_{ij})_{0 \leq i, j \leq 4}$, *matrice de distances* du graphe G , définie par :

pour tous les indices i, j , M_{ij} représente la distance entre les sommets i et j ,
ou encore la longueur de l'arête reliant les sommets i et j .

On convient que, lorsque les sommets ne sont pas reliés, cette distance vaut -1 . La distance du sommet i à lui-même est, bien sûr, égale à 0.

2. Écrire une suite d'instructions permettant de dresser à partir de la matrice M la liste des voisins du sommet 4.
3. Écrire une fonction **voisins**, d'argument un sommet i , renvoyant la liste des voisins du sommet i .
4. Écrire une fonction **degre**, d'argument un sommet i , renvoyant le nombre des voisins du sommet i , c'est-à-dire le nombre d'arêtes issues de i .
5. Écrire une fonction **longueur**, d'argument une liste L de sommets de G , renvoyant la longueur du trajet décrit par cette liste L , c'est-à-dire la somme des longueurs des arêtes empruntées. Si le trajet n'est pas possible, la fonction renverra -1 .