Algorithmique et programmation

Exercice 1

➔ **Écrire** une fonction `tri_selection` qui prend en paramètre une liste `tab` de nombres entiers et qui renvoie le tableau trié par ordre croissant.

On utilisera l'algorithme suivant :

- ✗ on recherche le plus petit élément du tableau, et on l'échange avec l'élément d'indice 0 ;
- ✗ on recherche le second plus petit élément du tableau, et on l'échange avec l'élément d'indice 1 ; on continue de cette façon jusqu'à ce que le tableau soit entièrement trié.

Exemple :

```
>>> tri_selection([1,52,6,-9,12])
[-9, 1, 6, 12, 52]
```

Exercice 2

➔ **Écrire** une fonction `occurrence_max` prenant en paramètres une chaîne de caractères `chaine` et qui renvoie le caractère le plus fréquent de la chaîne. La chaîne ne contient que des lettres en minuscules sans accent.

On pourra s'aider du tableau
`alphabet=['a','b','c','d','e','f','g','h','i','j','k','l','m','n','o,','p','q','r','s','t','u','v','w','x','y','z']`
et du tableau occurrence de 26 éléments où l'on mettra dans `occurrence[i]` le nombre d'apparitions de `alphabet[i]` dans la chaîne. Puis on calculera l'indice `k` d'un maximum du tableau occurrence et on affichera `alphabet[k]`.

Exemple :

```
>>> ch='je suis en terminale et je passe le bac et je souhaite poursuivre des etudes pour devenir expert en informatique'
>>> occurrence_max(ch)
'e'
```

Exercice 3

Le but de l'exercice est de compléter une fonction qui détermine si une valeur est présente dans un tableau de valeurs triées dans l'ordre croissant.

- ✗ L'algorithme traite le cas du tableau vide.
- ✗ L'algorithme est écrit pour que la recherche dichotomique ne se fasse que dans le cas où la valeur est comprise entre les valeurs extrêmes du tableau.

On distingue les trois cas qui renvoient `False` en renvoyant `False,1` , `False,2` et `False,3`.

➔ **Compléter** l'algorithme de dichotomie donné ci-après.

```
def dichotomie(tab, x):
    """
    tab : tableau trié dans l'ordre croissant
    x : nombre entier
    La fonction renvoie True si tab contient x et False sinon
    """
    # cas du tableau vide
    if ... :
        return False,1
    # cas où x n'est pas compris entre les valeurs extrêmes
    if (x < tab[0]) or ... :
        return False,2
    debut = 0
    fin = len(tab) - 1
    while debut <= fin:
        m = ...
        if x == tab[m]:
            return ...
        if x > tab[m]:
            debut = m + 1
        else:
            fin = ...
    return ...
```

Exemples :

```
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],28)
True
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],27)
(False, 3)
>>> dichotomie([15, 16, 18, 19, 23, 24, 28, 29, 31, 33],1)
(False, 2)
>>> dichotomie([],28)
(False, 1)
```

Exercice 4

Soit une image binaire représentée dans un tableau à 2 dimensions. Les éléments `M[i][j]`, appelés pixels, sont égaux soit à 0 soit à 1.
Une composante d'une image est un sous-ensemble de l'image constitué uniquement de 1 et de 0 qui sont côte à côte, soit horizontalement soit verticalement.
Par exemple, les composantes de

M =

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

sont

M =

0	0	1	0
0	1	0	1
1	1	1	0
0	1	1	0

On souhaite, à partir d'un pixel égal à 1 dans une image `M`, donner la valeur `val` à tous les pixels de la composante à laquelle appartient ce pixel.
La fonction `propager` prend pour paramètre une image `M`, deux entiers `i` et `j` et une valeur entière `val`. Elle met à la valeur `val` tous les pixels de la composante du pixel `M[i][j]` s'il vaut 1 et ne fait rien s'il vaut 0.

Par exemple, `propager(M,2,1,3)` donne :

M =

0	0	1	0
0	3	0	1
3	**3**	3	0
0	3	3	0

Compléter le code récursif de la fonction propager donné ci-dessous

```
def propager(M, i, j, val):
    if M[i][j]== ...:
        return
    M[i][j]=val
    # l'élément en haut fait partie de la composante
    if ((i-1) >= 0 and M[i-1][j] == ...):
        propager(M, i-1, j, val)
    # l'élément en bas fait partie de la composante
    if ((...) < len(M) and M[i+1][j] == 1):
        propager(M, ..., j, val)
    # l'élément à gauche fait partie de la composante
    if ((...) >= 0 and M[i][j-1] == 1):
        propager(M, i, ..., val)
    # l'élément à droite fait partie de la composante
    if ((...) < len(M) and M[i][j+1] == 1):
        propager(M, i, ..., val)
```

Exemple :

```
>>> M = [[0,0,1,0],[0,1,0,1],[1,1,1,0],[0,1,1,0]]
>>> propager(M,2,1,3)
>>> M
[[0, 0, 1, 0], [0, 3, 0, 1], [3, 3, 3, 0], [0, 3, 3, 0]]
```