

## Corrigé de la feuille d'exercices 4

### Les dictionnaires

#### Exercice 1

On constitue un dictionnaire de clés tous les nombres apparaissant et de valeur leur nombre d'occurrence, puis on effectue une recherche d'une clé de valeur maximale dans le dictionnaire.

À la question 2), on insère les clés de valeur maximale dans une liste durant la recherche de maximum, en n'oubliant pas de réinitialiser la liste dès qu'on trouve une occurrence strictement supérieure.

```
# 1.
def plus_frequent(L):
    # Constitution d'un dictionnaire
    D = {}
    for l in L:
        for x in l:
            if x in D:
                D[x] += 1
            else:
                D[x] = 1
    # Recherche element + frequent
    m = 0
    elt = None
    for c in D:
        if D[c] > m:
            m = D[c]
            elt = c
    return elt
```

```
# 2.
def liste_plus_frequent(L):
    # Constitution d'un dictionnaire
```

```
D = {}
for l in L:
    for x in l:
        if x in D:
            D[x] += 1
        else:
            D[x] = 1
# Recherche liste elements + frequent
m = 0
liste = []
for c in D:
    if D[c] > m:
        m = D[c]
        liste = [c]
    elif D[c] == m:
        liste.append(c)
return liste
```

#### Exercice 2

```
# 2)
nom = input("Saisir le nom du fichier : ")
texte = input("Saisir le texte du fichier : ")
nom = nom + ".txt"
objet_fichier = open(nom, 'w')
objet_fichier.write(texte)
objet_fichier.close()
```

```
## 2.a)
def caracteres(fichier):
    # Lecture du contenu du fichier
    nom = fichier + '.txt'
```

```

objet_fichier = open(nom, 'r')
texte = objet_fichier.read()  # Texte du fichier
objet_fichier.close()
# Constitution du dictionnaire
Dic = {}
for c in texte:
    if c in Dic:
        Dic[c] += 1
    else:
        Dic[c] = 1
return Dic

# 2.b)
nom = input("Nom du fichier ? ")
Dic = caracteres(nom)
S = 0
for c in Dic:
    S += Dic[c]
print(nom, "contient", S, "caracteres")

## 2.c)
def caracteresOrdonnees(fichier):
    Dic = caracteres(fichier)
    L = []
    for c in Dic.keys():
        L.append((c, Dic[c]))
    # Tri par insertion sur L
    N = len(L)
    for i in range(2, N):
        k = i
        x = L[i]
        while k > 0 and L[k-1][1] < x[1]:
            L[k] = L[k-1]

```

```

        k = k-1
        L[k] = x
    return L

## 3)
def mots(fichier):
    # Lecture du contenu du fichier
    nom = fichier + '.txt'
    objet_fichier = open(nom, 'r')
    texte = objet_fichier.read()  # Texte du fichier
    objet_fichier.close()
    # Constitution du dictionnaire
    Dic = {}
    nouveau_mot = ""
    non_vider = False
    for c in texte:
        if c not in (' ', '\t', '\n'):
            nouveau_mot += c
            non_vider = True
        else:
            if non_vider:
                if nouveau_mot in Dic:
                    Dic[nouveau_mot] += 1
                else:
                    Dic[nouveau_mot] = 1
            nouveau_mot = ""
            non_vider = False
    return Dic

```

**Exercice 3**

$$f(5) = 5 \quad f(28) = 1 \quad f(19) = 1 \quad f(15) = 6 \quad f(20) = 2$$

$$f(33) = 6 \quad f(12) = 3 \quad f(17) = 8 \quad f(10) = 1$$

Ce qui donne la table de hachage :

Indices	0	1	2	3	4	5	6	7	8
Alvéoles	[ ]	[28,19,10]	[20]	[12]	[ ]	[5]	[15,33]	[ ]	[17]

**Exercice 4**

1)

$$f(5) = 5 \quad f(28) = 1 \quad f(19) = 1 \rightarrow 2 \quad f(15) = 6 \quad f(20) = 2 \rightarrow 3$$

$$f(33) = 6 \rightarrow 7 \quad f(12) = 3 \rightarrow 4 \quad f(17) = 8 \quad f(10) = 1 \rightarrow 0$$

Ce qui donne la table de hachage :

Indices	0	1	2	3	4	5	6	7	8
Alvéoles	10	28	19	20	12	5	15	33	17

2)

```
omega = 10
```

```
D = [None] * omega
```

```
def f(m):
    return m % omega
```

```
def valeur(D, c):
    i = f(c)
    for k in range(i, i+len(D)):
        ind = k % omega
        if D[ind][0] == c:
            return D[ind][1]
```

```
def ajout(D, c, v):
    i = f(c)
    while D[i] != None:
        i = i + 1 % omega
    D[i] = (c, v)
```

```
def recherche(D, c):
    i = f(c)
    for k in range(i, i+len(D)):
        ind = k % omega
        if D[ind] != None:
            if D[ind][0] == c:
                return True
    return False
```

3) Après suppression de l'association de clé  $c$  il faut parcourir les éléments suivant pour décaler ceux dont la clé a même valeur de hachage que  $c$  à la position précédemment libérée.

```
def supprime(D, c):
    """On suppose la cle presente"""
    i = f(c)
    dernier = False
    for k in range(i, i+len(D)):
        ind = k % omega
        if D[ind][0] == c:
            D[ind] = None
            dernier = ind
        elif dernier != False and f(D[ind][0]) == i:
            D[dernier] = D[ind]
            D[ind] = None
            dernier = ind
```

**Exercice 5**

1.

```
def str2int(ch):  
    n = len(ch)  
    S = 0  
    for k in range(n-1,-1,-1):  
        S = 256*S + ord(ch[k])  
    return S
```

2. Pour optimiser le coût on reprend le calcul en prenant chaque résultat intermédiaire modulo  $\Omega$ . Cela évite de calculer avec des nombres  $> \Omega$  et le modulo étant compatible avec  $+$  et  $\times$  on obtient le même résultat que si l'on faisait `return str2int(ch) % omega`.

```
omega = 100  
  
def hachage(ch):  
    n = len(ch)  
    S = 0  
    for k in range(n-1,-1,-1):  
        S = (256*S + ord(ch[k])) % omega  
    return S
```