CORRIGÉ DS INFO 01/2017 (d'après X 2017 MP/PC)

```python
# DS Info TPC-PSI 2016

###############
# Question 1 #
###############

def transforme(texte):
    # version basique
    t = []
    for car in texte:
        t.append(ord(car))
    return t

def transforme2(texte):
    # version plus compacte
    return list(map(ord, texte))

# la fonction transforme fait autant d'opérations élémentaires que la longueur
# n du texte; sa complexité est donc O(n)

###############
# Question 2 #
###############

def occurrences(t):
    occ = [0]*256
    for car in t:
        occ[car] += 1
    return occ

# cette fonction se contente de parcourir le tableau t de longueur n
# sa complexité est donc O(n)

###############
# Question 3 #
###############

def mini(t):
    occ = occurrences(t)
    mu = 0
    for k in range(1, len(occ)): # len(occ) = 256!
        if occ[k] < occ[mu]:
            mu = k
    return mu

###############
# Question 4 #
###############

def compresse(t):
    n = len(t)
    mu = mini(t)
    t0 = [mu]
    i, j = 0, 0
    # i = indice du caractère en train d'être examiné
    # les éléments entre t[i] et t[j] inclus sont tous égaux
    while i < n:
```

```
58          x = t[i] # pour éviter de l'appeler plusieurs fois
59          while j < n and t[j] == x:
60              j += 1
61          if j - i <= 3:
62              t0.extend([x]*(j-i))
63          else:
64              t0.extend([mu, j - i - 1, x])
65          i = j
66      return t0
67
68  ###############
69  # Question 5 #
70  ###############
71
72  def decompresse(t0):
73      n = len(t0)
74      mu = t0[0]
75      t = []
76      i = 1
77      while i < n:
78          if t0[i] != mu:
79              t.append(t0[i])
80              i += 1
81          else:
82              t.extend((t0[i+1]+1)*[t0[i+2]])
83              i += 3
84      return(t)
85
86  ###############
87  # Question 6 #
88  ###############
89
90  # on remarque que rot[i] est formée de la fin de la liste à partir du caractère
91  # d'indice i à laquelle on a concaténé le début de cette liste
92
93  def comparerRotations1(t, i, j):
94      # cette version a le défaut de créer 2 listes supplémentaires
95      # pour stocker les rotations d'indices i et j
96      rot_i = t[i:] + t[0:i]
97      rot_j = t[j:] + t[0:j]
98      for k in range(len(t)):
99          if rot_i[k] > rot_j[k]:
100             return 1
101         if rot_i[k] < rot_j[k]:
102             return -1
103     return 0
104
105 def comparerRotations(t, i, j):
106     # bien meilleure
107     n = len(t)
108     for k in range(len(t)):
109         # k-ième caractère des chaines obtenues après rotation
110         rot_i_k = t[(i+k)%n]
111         rot_j_k = t[(j+k)%n]
112         if rot_i_k > rot_j_k:
113             return 1
114         if rot_i_k < rot_j_k:
115             return -1
116     return 0
117
118 def triRotations(t):
```

```
    # trie les rotations comme dans l'énoncé, mais on a utilisé ici  pour
    # simplifier un tri par insertion, de complexité O(n^2)
    # Il faudrait utiliser un tri fusion ou un tri rapide pour avoir
    # une complexité en O(n ln(n))
    n = len(t)
    r = [i for i in range(n)]
    #la liste r contient les numéros des rotations
    for i in range(1,n):
        x = r[i]
        j = i-1
        while (j >= 0) and (comparerRotations(t, r[j], x) == 1):
            j -= 1
        r[j+2:i+1] = r[j+1:i]
        r[j+1] = x
    return r

###############
# Question 7 #
###############

# on remarque que la dernière lettre de rot[i] est en fait t[i-1]
#(vrai même si i=0!)
# Pour la clé, on remarque que c'est le dernier caractère du mot obtenu
# après la rotation numéro 1

def codageBW(t):
    t0 = []
    r = triRotations(t)
    for i in range(len(r)):
        if r[i] == 1:
            cle = i
        t0.append(t[r[i]-1])
    return t0, cle

# La fonction triRotations effectue O(n ln(n)) appels à la procédure
# comparerRotations qui est, elle, de complexité O(n). Elle est donc de
# de complexité O(n^2 ln(n)), et il en est de même de la fonction codageBW
# puisque celle-ci effectue en plus seulement n boucles.

###############
# Question 8 #
###############

# la fonction frequences est identique à la fonction occurrences !

def frequences(t0):
    t = t0[:len(t0)]
    freq = [0]*256
    for car in t:
        freq[car] += 1
    return freq

###############
# Question 9 #
###############

def triCarsDe(t0):
    triCars = []
    freq = frequences(t0)
    for i in range(256):
        if freq[i] != 0:
```

```
            triCars.extend(freq[i]*[i])
    return triCars

###############
# Question 10 #
###############

def trouverIndices(t0):
    n = len(t0)
    freq = frequences(t0)
    triCars = triCarsDe(t0)
    i = 0
    indices = []
    while i < n:
        k = 0 # indice qui va servir à parcourir le tableau t0
        car = triCars[i]
        m = freq[car]
        compteur = 0
        j = 1
        while j <= m:
            #on cherche ici la j-ième apparition de triCars[i] dans t0
            while k < n and compteur < j:
                if t0[k] == car:
                    compteur += 1
                    if compteur == j:
                        indices.append(k)
                k += 1
            j += 1
        i += m
    return indices

def trouverIndices2(t0):
    # version plus compacte, proposée par un élève
    t = triCarsDe(t0)
    indices = []
    for x in t:
        i = 0
        while  (x != t0[i] or i in indices):
            i += 1
        indices.append(i)
    return indices

# Si n=len(t0), la boucle extérieure est faite n fois et à chaque fois
# il faut parcourir le tableau t0 .
# La complexité de la procédure est donc O(n^2)

###############
# Question 11 #
###############

def decodageBW(t0, cle):
    indices = trouverIndices2(t0)
    pos = cle
    t = []
    for k in range(len(t0)):
        t.append(t0[pos])
        pos = indices[pos]
    return t

# Cettefonction fait appel à la fonction trouverIndices,
# puis fait ensuite une boucle de longueur n. Elle est donc aussi
```

```
# de complexité O(n^2)

# L'ensemble du processus compression/décompression sera donc de complexité
# O(n) + O(n^2) + O(n^2 ln(n)) = O(n ln(n))

# On exécute alors toutes les procédures précédentes pour vérifier!

# Une procédure utile
def list_to_str(liste):
    # Convertit une liste d'entiers entre 0 et 255 en chaine
    return "".join(chr(i) for i in liste)

texte = 'Ô mathématiques sévères, je ne vous ai pas oubliées, depuis que \
vos savantes leçons, plus douces que le miel, filtrèrent \
dans mon coeur, comme une onde rafraîchissante.\n' +\
'J\'aspirais instinctivement, dès le berceau, à boire à votre source, \
plus ancienne que le soleil, et je continue encore de fouler le parvis \
sacré de votre temple solennel, moi, le plus fidèle de vos initiés.\n' + \
'Il y avait du vague dans mon esprit, un je ne sais quoi épais comme de \
la fumée ; mais, je sus franchir religieusement les degrés qui mènent à votre \
autel, et vous avez chassé ce voile obscur, comme le vent chasse le damier.\n' +\
'Vous avez mis, à la place, une froideur excessive, une prudence consommée\
et une logique implacable.\n' +\
'À l\'aide de votre lait fortifiant, mon intelligence s\'est rapidement \
développée, et a pris des proportions immenses, au milieu de cette clarté \
ravissante dont vous faites présent, avec prodigalité, à ceux qui \
vous aiment d\'un sincère amour.\n'+\
'Arithmétique! algèbre! géométrie! trinité grandiose! triangle lumineux! \
Celui qui ne vous a pas connues est un insensé !\n'+\
'Il mériterait l\'épreuve des plus grands suplices ; car, il y a du mépris \
aveugle dans son insouciance ignorante; mais, celui qui vous connaît et vous \
apprécie ne veut plus rien des biens de la terre ; se contente de vos \
jouissances magiques; et, porté sur vos ailes sombres, ne désire plus \
que de s\'élever, d\'un vol léger, en construisant une hélice ascendante, \
vers la voûte sphérique des cieux.\n'+\
'La terre ne lui montre que des illusions et des fantasmagories morales; \
mais vous, ô mathématiques concises, par l\'enchaînement rigoureux de vos \
propositions tenaces et la constance de vos lois de fer, vous faites luire, \
aux yeux éblouis, un reflet puissant de cette vérité suprême dont on \
remarque l\'empreinte dans l\'ordre de l\'univers.\n\n'+\
'Comte de Lautréamont, Les Chants de Maldoror, Chant 10, Strophe 2.'

# tranformation du texte en liste de caractères puis de nombres
t = transforme(list(texte))
# transformation de BW
t0, cle = codageBW(t)
# compression
t_compresse = compresse(t0)
# comparaison des longueurs des textes
print('Longueur texte initial: ',len(t))
print('Longueur texte compressé: ', len(t_compresse),'\n')
# décompression
t0 = decompresse(t_compresse)
# transformation de BW inverse
t = decodageBW(t0, cle)
# on transforme la liste de nombres en chaîne
texte_retrouve = list_to_str(t)
# et on affiche
print(texte_retrouve)
```

```
Longueur texte initial:  1865
Longueur texte compressé:  1746

Ô mathématiques sévères, je ne vous ai pas oubliées, depuis que vos savantes leçons, plus douces qu
J'aspirais instinctivement, dès le berceau, à boire à votre source, plus ancienne que le soleil, et
Il y avait du vague dans mon esprit, un je ne sais quoi épais comme de la fumée ; mais, je sus fran
Vous avez mis, à la place, une froideur excessive, une prudence consomméeet une logique implacable.
À l'aide de votre lait fortifiant, mon intelligence s'est rapidement développée, et a pris des prop
Arithmétique! algèbre! géométrie! trinité grandiose! triangle lumineux! Celui qui ne vous a pas con
Il mériterait l'épreuve des plus grands suplices ; car, il y a du mépris aveugle dans son insoucian
La terre ne lui montre que des illusions et des fantasmagories morales; mais vous, ô mathématiques

Comte de Lautréamont, Les Chants de Maldoror, Chant 10, Strophe 2.
```