Théorie de l'information : projet algorithmique autour de l'algorithme de compression `bzip2`

Michel Nguyen
Etudiant en première année de Master de mathématiques

Paris VIII - Enseignant : J. Lavauzelle

Décembre 2020

1 Introduction

Avec ce projet, nous souhaitons implémenter et expérimenter un algorithme de compression de manière simplifiée basé sur le logiciel libre `bzip2`. Pour cela, nous poserons les fonctions `Burrows-Wheeler`, `move-to-front` et `run-length encoding` en s'inspirant des notes du cours. Ainsi, la fonction `compress` effectue successivement la séquence d'algorithmes BW, MTF et RLE.

En entrée, on insère un message initial et un alphabet. Il faut faire attention à ce que les caractères du message initial soient bien compris dans l'alphabet. A la sortie, nous aurons une chaîne de caractères composée de nombres et de lettres sans aucune signification particulière. L'objectif est d'obtenir une taille inférieure à la taille du message initial, sinon, c'est un algorithme inefficace.

De plus, nous poserons les fonctions inverses pour faire la fonction `uncompress` qui effectue la séquence d'algorithmes `invRLE`, `invMTF` et `invBW` pour retrouver le message initial. Pour plus de clarté, nous avons ajouté la fonction `comparaison` qui compare la taille du message initial et du message compressé.

Rajoutons que la fonction `RLE` n'affiche aucun 1 sauf éventuellement pour le dernier élément pour un gain de place. Aussi, l'algorithme que nous proposons ici ne fonctionne qu'avec un alphabet composé de 1 caractère par élément.

2 Les fonctions en python

- Les deux fonctions utilisées dans `BW` et `MTF`

```
def rotate_str(x,i):
    return x[i:] + x[:i]

def movefirst(T,i):
    T.insert(0,T[i])
    T.pop(i+1)
    return T
```

- Les fonctions principales

```
def BW(A,x):
    Tab = []
    SortedTab = []
    y = ''

    for i in range (0,len(x)):
        Tab.append(rotate_str(x,i))
    SortedTab = sorted(Tab)

    for i in range (0,len(x)):
        y = y + SortedTab[i][len(x)-1]

    for i in range (0,len(x)):
        if SortedTab[i] == x:
            n = i
    return (n,y)


def MTF(A,x):
    Tab = list(A)
    y = []

    for i in range (0,len(x)):
        for j in range (0,len(Tab)):
            if Tab[j] == x[i]:
```

```
                y.append(j)
                Tab = movefirst(Tab,j)
    return y


def RLE(A,x):
    y = ''
    cpt = 1

    for i in range(0,len(x)):
        x[i] = (A[x[i]])

    lettre = x[0]
    for i in range (1,len(x)):
        if lettre == x[i]:
            cpt = cpt + 1
        else:
            if cpt == 1 :
                y = (y + lettre)
                cpt = 1
                lettre = x[i]
            else :
                y = (y + str(cpt) + lettre)
                cpt = 1
                lettre = x[i]
    y = (y + str(cpt) + lettre)
    return y
```

- Les fonctions inverses

```
def invRLE(A,y):
    x = ''
    cpt = ''

    for i in range (0,len(y)):
        if (y[i].isnumeric()) :
            cpt = cpt + str(y[i])
        else :
            if cpt == '' :
                x = x + y[i]
            else :
                x = x + (int(cpt) * y[i])
                cpt = ''

    x = list(x)
    for i in range(0,len(x)) :
        for j in range(0,len(A)) :
            if x[i] == A[j] :
                x[i] = int(j)
    return x


def invMTF(A,y):
    Tab = list(A)
    x = ''

    for i in range (0,len(y)):
        valeur = y[i]
        x = x + Tab[valeur]
        Tab = movefirst(Tab,valeur)
    return x

```

```
33
34  def invBW(n,y):
35      x = ''
36      Tab = []
37
38      for i in range(0,len(y)):
39          Tab.append(y[i])
40      Tab = sorted(Tab)
41
42      for i in range(1,len(y)):
43          for j in range(0,len(y)):
44              Tab[j] = y[j] + Tab[j]
45          Tab = sorted(Tab)
46      x = Tab[n]
47      return x
```

- Les fonctions `compress`, `uncompress` et `comparaison`

```
1   def compress(A,x):
2       P = (BW(A,x))[0]
3       res = RLE(A,MTF(A,(BW(A,x))[1]))
4       return (res,P)
5
6
7   def uncompress(A,res):
8       inv = invBW((compress(A,X))[1],invMTF(A,invRLE(A,(compress(A,X))[0])))
9       return inv
10
11  def comparaison(initial,compression):
12      if len(initial) >= len(compression):
13          print ('la taille initiale du message est',len(initial),'caractères;','\nla
        taille après compression est', len(compression),'caractères;','\nsoit un gain de',
        len(initial) - len(compression), 'caractère(s).\n')
14      else:
15          print ('la taille initiale du message est',len(initial),'caractères;','\nla
        taille après compression est', len(compression),'caractères;')
16          print ("la compression n'est pas efficace car il y a", len(compression)- len(
        initial), "caractère(s) qui se sont ajoutés.\n")
```

Les commentaires sont à retrouver dans l'archive.

Mise en page du code à retrouve sur le site https://borntocode.fr/latex-comment-inserer-et-customiser-du-code-source/

3 Questions

À travers les questions que nous allons répondre, nous considérons l'alphabet $\mathcal{A}$ constitué de 32 éléments avec un seul caractère par élément tel que :

$$\mathcal{A} = [A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V,$$

$$W, X, Y, Z, , ., , , ?, !, ']$$

Ici, tous les messages sont uniquement composés de caractères compris dans cet alphabet. Pour mieux comprendre le fonctionnement des algorithmes proposés, il est préférable de lire les commentaires qui sont à trouver dans l'archive.

3.1 Premières expériences

Question 1. Implanter, puis tester toutes les fonctions à implanter présentées dans la Section 1. On vérifiera notamment que la réciproque d'une procédure renvoie bien le texte initial. Pour cela, on pourra par exemple utiliser les exemples simples donnés dans le fichier `exemples.txt` de l'archive `projet-algo.zip`.

Pour vérifier que les fonctions réalisent bien la tâche demandée, nous allons essayer chaque fonction et son inverse. Si le résultat d'une fonction inverse renvoie le message sans modification, alors la fonction inverse est correcte. D'abord, on applique sur un mot sans signification de longueur 10. Enfin, nous faisons de même sur un texte en français. Les exemples sont tirés des textes du professeur.

Message initial	**T10 = "BBABBAAABA"**
Burrows-Wheeler : BW(A,T10)	(9,'BAABBBBAAA')
Inverse de BW : invBW(9,'BAABBBBAAA')	'BBABBAAABA'
move-to-front : MTF(A,'BAABBBBAAA')	[1, 1, 0, 1, 0, 0, 0, 1, 0, 0]
Inverse de MTF : invMTF(A,[1, 1, 0, 1, 0, 0, 0, 1, 0, 0])	'BAABBBBAAA'
run-length encoding : RLE(A,[1, 1, 0, 1, 0, 0, 0, 1, 0, 0])	**'2BAB3AB2A'**
Inverse de RLE : invRLE(A,'2BAB3AB2A')	[1, 1, 0, 1, 0, 0, 0, 1, 0, 0]

On observe que les fonctions principales et leurs inverses fonctionnent correctement. Pour le cas de T10, on observe que la taille du message compressé est 9 : '2BAB3AB2A', donc il y a un gain de 1 caractère. Maintenant, essayons avec un texte de longueur 407 en affichant BW, MTF et RLE à la suite pour plus de clarté.

On pose Texte1 = "IL PLEURE DANS MON COEUR COMME IL PLEUT SUR LA VILLE QUELLE EST CETTE LANGUEUR QUI PENETRE MON COEUR ? O BRUIT DOUX DE LA PLUIE PAR TERRE SUR LES TOITS ! POUR UN COEUR QUI S'ENNUIE, O LE CHANT DE LA PLUIE ! IL PLEURE SANS RAISON DANS CE COEUR QUI S'ECOEURE. QUOI ! NULLE TRAHISON ? CE DEUIL EST SANS RAISON. C'EST BIEN LA PIRE PEINE DE NE SAVOIR POURQUOI SANS AMOUR ET SANS HAINE MON COEUR A TANT DE PEINE !"

Burrows-Wheeler	(203, "EISENRRSTO.S?TENNNENRNETXETETELRS!ENEEREOREESE!?,EEEIALLLAA! RERRR.SSIIITETETEARSERA SSCENE LLL LRHRR LSDSDSSHTPS E INLCRNCLMDDTRN-DIDRLRNRLIR'PPUIP'TL' NCDOOOUOOLLOLN CAOUUUOUUUBU! VEEAOPHAAUOIIII LLL PPPEIUPPMO AOOOOUOEOIII EAEAAAAAAAAN CCCCCCUUVTCSMMMSSMPPD RUUUUUUIUUUAUT UTIURUUEBTNNNNNNE IIIEEE SSNNIESU T EIEQGQQQLLNERN QQEEEOSSEEEOEEEOEO AU")
MTF	[4, 8, 18, 2, 14, 18, 0, 3, 19, 17, 27, 3, 29, 4, 7, 7, 0, 0, 1, 1, 7, 1, 2, 3, 25, 2, 2, 1, 1, 1, 19, 5, 7, 30, 4, 7, 1, 0, 4, 1, 10, 2, 2, 0, 5, 1, 5, 9, 30, 3, 0, 0, 12, 13, 10, 0, 0, 1, 0, 6, 8, 5, 1, 0, 0, 13, 9, 0, 7, 0, 0, 12, 5, 1, 1, 1, 1, 7, 6, 5, 3, 2, 3, 30, 0, 0, 0, 4, 0, 16, 5, 14, 1, 4, 0, 11, 0, 0, 1, 1, 7, 20, 1, 0, 3, 3, 7, 18, 1, 1, 1, 0, 5, 10, 24, 3, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 10, 9, 11, 11, 3, 2, 3, 24, 12, 0, 11, 6, 6, 3, 7, 1, 3, 6, 1, 4, 1, 2, 4, 2, 11, 0, 26, 3, 2, 7, 5, 10, 0, 0, 7, 10, 9, 19, 0, 0, 9, 1, 0, 6, 0, 1, 1, 5, 6, 6, 15, 5, 6, 0, 0, 1, 1, 0, 0, 21, 1, 18, 6, 0, 27, 16, 0, 7, 7, 13, 18, 3, 0, 8, 4, 15, 0, 0, 0, 8, 0, 0, 0, 0, 0, 13, 0, 0, 1, 7, 0, 0, 8, 4, 6, 3, 0, 17, 7, 6, 0, 0, 8, 2, 0, 0, 0, 5, 1, 7, 1, 7, 0, 0, 5, 3, 5, 1, 1, 0, 0, 0, 0, 0, 0, 0, 14, 3, 0, 0, 14, 0, 0, 0, 0, 0, 7, 0, 12, 16, 3, 18, 11, 0, 0, 1, 0, 1, 12, 0, 17, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 9, 0, 0, 0, 0, 0, 13, 1, 0, 0, 12, 1, 10, 5, 0, 2, 2, 4, 2, 5, 1, 0, 13, 18, 5, 14, 0, 0, 0, 0, 0, 3, 7, 0, 0, 0, 0, 0, 0, 0, 7, 0, 0, 2, 0, 0, 2, 0, 12, 0, 4, 0, 4, 4, 3, 7, 5, 6, 1, 0, 0, 4, 5, 1, 27, 25, 1, 0, 0, 18, 0, 9, 4, 11, 2, 7, 5, 0, 4, 0, 0, 18, 11, 0, 2, 0, 0, 2, 1, 0, 0, 1, 1, 1, 4, 13, 12]
RLE	'EISCOSADTR.D?E2H2A2BHBCDZ2C3BTFH!EHBAEBK2CAFBFJ!D2AMNK2ABAGIFB2AN JAH2AMF4BHGFDCD!3AEAQFOBEAL2A2BHUBA2DHS3BAFKYDG11AIB8AMKJ2LDCDYM AL2GDHBDGBEBCECLA DCHFK2AHKJT2AJBAGA2BF2GPFG2A2B2AVBSGA.QA2HNSDA IEP3AI5AN2ABH2AIEGDARHG2AIC3AFBHBH2AFDF2B7AOD2AO5AHAMQDSL2ABABMAR I16ASJ5ANB2AMBKFA2CECFBANSFO5ADH6AH2AC2ACAMAEA2EDHFGB2AEFB.ZB2ASA JELCHFAE2ASLAC2ACB2A3BEN1M'

Inverse de RLE	[4, 8, 18, 2, 14, 18, 0, 3, 19, 17, 27, 3, 29, 4, 7, 7, 0, 0, 1, 1, 7, 1, 2, 3, 25, 2, 2, 1, 1, 1, 19, 5, 7, 30, 4, 7, 1, 0, 4, 1, 10, 2, 2, 0, 5, 1, 5, 9, 30, 3, 0, 0, 12, 13, 10, 0, 0, 1, 0, 6, 8, 5, 1, 0, 0, 13, 9, 0, 7, 0, 0, 12, 5, 1, 1, 1, 1, 7, 6, 5, 3, 2, 3, 30, 0, 0, 0, 4, 0, 16, 5, 14, 1, 4, 0, 11, 0, 0, 1, 1, 7, 20, 1, 0, 3, 3, 7, 18, 1, 1, 1, 0, 5, 10, 24, 3, 6, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 8, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 12, 10, 9, 11, 11, 3, 2, 3, 24, 12, 0, 11, 6, 6, 3, 7, 1, 3, 6, 1, 4, 1, 2, 4, 2, 11, 0, 26, 3, 2, 7, 5, 10, 0, 0, 7, 10, 9, 19, 0, 0, 9, 1, 0, 6, 0, 1, 1, 5, 6, 6, 15, 5, 6, 0, 0, 1, 1, 0, 0, 21, 1, 18, 6, 0, 27, 16, 0, 7, 7, 13, 18, 3, 0, 8, 4, 15, 0, 0, 0, 8, 0, 0, 0, 0, 0, 13, 0, 0, 1, 7, 0, 0, 8, 4, 6, 3, 0, 17, 7, 6, 0, 0, 8, 2, 0, 0, 0, 5, 1, 7, 1, 7, 0, 0, 5, 3, 5, 1, 1, 0, 0, 0, 0, 0, 0, 0, 14, 3, 0, 0, 14, 0, 0, 0, 0, 0, 7, 0, 12, 16, 3, 18, 11, 0, 0, 1, 0, 1, 12, 0, 17, 8, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 18, 9, 0, 0, 0, 0, 0, 13, 1, 0, 0, 12, 1, 10, 5, 0, 2, 2, 4, 2, 5, 1, 0, 13, 18, 5, 14, 0, 0, 0, 0, 0, 3, 7, 0, 0, 0, 0, 0, 0, 7, 0, 0, 2, 0, 0, 2, 0, 12, 0, 4, 0, 4, 4, 3, 7, 5, 6, 1, 0, 0, 4, 5, 1, 27, 25, 1, 0, 0, 18, 0, 9, 4, 11, 2, 7, 5, 0, 4, 0, 0, 18, 11, 0, 2, 0, 0, 2, 1, 0, 0, 1, 1, 1, 4, 13, 12]
Inverse de MTF	'EISENRRSTO.S?TENNNENRNETXETETELRS!ENEEREOREESE!?,EEEIALLLAA!RER RR.SSIIITETETEARSERA SSCENE LLL LRHRR LSDSDSSHTPS E INLCRNCLMDDTRN-DIDRLRNRLIRPPUIPTL NCDOOOUOOLLOLN CAOUUUOUUUBU! VEEAOPHAAUOIIII LLL PPPEIUPPMO AOOOOUOEOIII EAEAAAAAAAAN CCCCCCUUVTCSMMMSSMPPD RUUUUUUIUUUAUT UTIURUUEBTNNNNNNE IIIEEE SSNNIESU T EIEQGQQQLLNERN QQEEEOSSEEEOEEEOEO AU'
Inverse de BW	"IL PLEURE DANS MON COEUR COMME IL PLEUT SUR LA VILLE QUELLE EST CETTE LANGUEUR QUI PENETRE MON COEUR ? O BRUIT DOUX DE LA PLUIE PAR TERRE SUR LES TOITS ! POUR UN COEUR QUI S'ENNUIE, O LE CHANT DE LA PLUIE ! IL PLEURE SANS RAISON DANS CE COEUR QUI S'ECOEURE. QUOI ! NULLE TRAHISON ? CE DEUIL EST SANS RAISON. C'EST BIEN LA PIRE PEINE DE NE SAVOIR POURQUOI SANS AMOUR ET SANS HAINE MON COEUR A TANT DE PEINE !"

Même constat que le premier exemple, les trois fonctions et leurs inverses répondent bien à la tâche demandée. Ainsi, on peut définir la fonction `compress` qui suit :

```
def compress(A,x):
    P = (BW(A,x))[0] #on doit garder en mémoire le rang du message initial lors de la fonction BW
    res = RLE(A,MTF(A,(BW(A,x))[1])) #correspond au résultat par la fonction RLE
    return (res,P) #en sortie, le résultat et le rang
```

Question 2. Proposer, et valider expérimentalement par des mesures de temps, une complexité pour les algorithmes `BW`, `MTF` et `RLE` que vous avez implémentés. Cette complexité sera donnée approximativement, en fonction de la taille de l'entrée comptée en nombre de symboles.

On pose $T(n)$ qui représente la complexité d'un algortihme en fonction de la taille n. Pour calculer $T(n)$, on propose le mode de calcul suivant en posant $len(x) = n =$ *"taille de l'entrée"* :

- À chaque affection c'est-à-dire attribution de type ou de valeur , on ajoute 1 (exemple : $Tab = []$).
- À chaque calcul, on ajoute 1.
- À chaque comparaison, on ajoute 1.
- À chaque boucle, on multiplie par n.

Pour chaque boucle de type `"for i in range (0,len(x))"`, on a d'office $3n$ car il y a une affectation de i, une comparaison, et une soustraction par rapport à n (on commence de 0 jusqu'à $n-1$). On s'intéresse au pire des cas. Ainsi, on a :

```
def BW(A,x): #fonction Burrows-Wheeler
    Tab = [] #+1
    SortedTab = [] #+1
    y = '' #+1

    for i in range (0,len(x)): #pour la boucle, on a 3n + 2n (car deux affectations)
```

```
7           Tab.append(rotate_str(x,i))
8       SortedTab = sorted(Tab) #au maximum n fois
9
10      for i in range (0,len(x)): #6n fois
11          y = y + SortedTab[i][len(x)-1]
12
13      for i in range (0,len(x)): #au maximum 5n fois
14          if SortedTab[i] == x:
15              n = i
16      return (n,y)
17
18  #au total T(n) = 3n + 2n + 6n + 5n + 3 = 16n + 3
19
20
21  def MTF(A,x): #fonction move-to-front
22      Tab = list(A) #+1
23      y = [] #+1
24
25      for i in range (0,len(x)): #il y a 3n*6n=18n car il y a une boucle imbriquée dans une
        autre
26          for j in range (0,len(Tab)): #au maximum 6n fois
27              if Tab[j] == x[i]:
28                  y.append(j)
29                  Tab = movefirst(Tab,j)
30      return y
31
32  #au total T(n) = 18n + 2
33
34
35  def RLE(A,x): #fonction run-length encoding
36      y = '' #+1
37      cpt = 1 #+1
38
39      for i in range(0,len(x)):  #4n fois
40          x[i] = (A[x[i]])
41
42      lettre = x[0]  #+1
43      for i in range (1,len(x)): #on prend max (6n,8n,8n) = 8n car chaque étape est traitée
        par un des trois cas
44          if lettre == x[i]: #dans ce cas, 3n + 3n = 6n fois
45              cpt = cpt + 1
46          else:
47              if cpt == 1 : #dans ce cas, 3n + 5n = 8n fois
48                  y = (y + lettre)
49                  cpt = 1
50                  lettre = x[i]
51              else : #dans ce cas, 3n + 5n = 8n fois
52                  y = (y + str(cpt) + lettre)
53                  cpt = 1
54                  lettre = x[i]
55      y = (y + str(cpt) + lettre) #+3
56      return y
57
58  #au total T(n)= 4n + 8n + 3 + 2 = 12n + 5
```

Donc, pour BW, on propose une complexité de $16n + 3$. Pour MTF, on a $18n + 2$. Et pour RLE, on a $12n + 5$.

Question 3. Générer des chaînes de caractères tirés uniformément et indépendamment, de différentes longueurs N (par exemple $N = 10, 30, 100, 300, 1000$). Utiliser ces chaînes aléatoires pour avoir une estimation de la longueur moyenne obtenue par votre algorithme de compression `compress`. On pourra utiliser une dizaine de textes pour chaque longueur.

Notre démarche est de générer quelques chaînes de caractères de longueurs $\mathcal{N}$ où leurs caractères sont pris aléatoirement dans $\mathcal{A}$ via la fonction `randomchaine`, effectuer la fonction `compress`, et à l'aide de la fonction `comparaison`, on peut directement comparer la taille du message initial au message compressé. Voici les fonctions qu'on va utiliser :

```
import random

def comparaison(initial,compression): #fonction comparaison
    if len(initial) >= len(compression):
        print ('la taille initiale du message est',len(initial),'caractères;','\nla taille après compression est', len(compression),'caractères;','\nsoit un gain de', len(initial) - len(compression), 'caractère(s).\n')
    else:
        print ('la taille initiale du message est',len(initial),'caractères;','\nla taille après compression est', len(compression),'caractères;')
        print ("la compression n'est pas efficace car il y a", len(compression)- len(initial), "caractère(s) qui se sont ajoutés.\n")


def randomchaine(n): #fonction randomchaine
    chaine = ''
    for i in range(0,n):
        chaine = chaine + random.choice(A) #choix aléatoire parmi les éléments de l'alphabet A
    initial = chaine
    print (chaine) #afficher la chaîne de caractères
    chaine = compress(A,chaine)
    comparaison(initial,chaine[0]) #utilisation de la fonction comparaison par rapport à la chaîne générée et la chaîne compressée

for i in range(0,k):    #liste de k chaînes de caractères
    randomchaine(n)       #préciser la longueur des chaînes de caractères
```

Les résultats sont à retrouver dans le tableau ci-dessous avec plusieurs longueurs différentes.

$\mathcal{N} = 10$	MEY CYCPTP la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. ,YKCDD 'JQ la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. G NO ISSDW la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. Z?RU!VHRYC la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. ,WTFEGQTGZ la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. SECHMRTMNT la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. CNLPTWJ VH la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. !KUGUWMXXM la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. HFNPEF?NF' la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. EKILBHLCR! la taille initiale du message est 10 caractères; la taille après compression est 11 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés.

$\mathcal{N} = 30$	N?R,XXUPGK HW!T,VMRMCC,C .GY ' la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. ILGGDUYUYEYM.JWSVGBAYBY'HNR,H la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. HL'RIESKWT,'S' PQ.N.ODS,YADRV, la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. CDY,LX FXWMK.MPKWNHU.JV CAMQP la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. ENA.DU.IMYGLRXXSBO?P?WAKJXBGOL la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. !MUTKOJTUDEPCCJUK?.,GH PKFXL?F la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. XUJHAOZNCX KVC,?XBJTRBJMG!NR.S la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. FVHWFVLMPTKPMLLLGRKWUONIHRVKWF la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. BVGR NZ?OFWE!JWUNJPSJHNPYLBOBM la taille initiale du message est 30 caractères; la taille après compression est 31 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. "KYJ.WYNMCU?NAFTC WCBDCY GYXO la taille initiale du message est 30 caractères; la taille après compression est 30 caractères; soit un gain de 0 caractère(s).
$\mathcal{N} = 100$	PWZAMTHBJHOSTOQX??.PHJOTOQOXDRIAMFA.YK!JYZORQXYPHD?JMOUQJSS.C.DUO PE,SOFUC!MIRKA,?VD.JZ'MOI'ZEU?W.KY la taille initiale du message est 100 caractères; la taille après compression est 101 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. HYHM LZRD?LSZYNTK.!DYSVGPHDIC'KDASFELSR?EQ?P!,NGISXKFGPQ'WMSROBYSF'W AWZFAG!BRR!VZ!KHMBMO,CN?.AFYGBX la taille initiale du message est 100 caractères; la taille après compression est 101 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. ?YFAMNNLH?LLZUOZFI.ZIOIVB.XLF.S LFJ .QGPEHP,BEVBMDEUJOT'TBFPXVJIAOXX' NB-SCDGU'IFUNIYBQ JUUM,NPQQ,SMV la taille initiale du message est 100 caractères; la taille après compression est 101 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. 'F!JRWHHB.RASJITNIOVYU!,IEDRZR?WGI HPN,.ZIYQ?!VI,RC,PJWSF HB,TLDQYVTUKPYEMGPG,NGGRNSYC.P?TSSELGWOQSB la taille initiale du message est 100 caractères; la taille après compression est 101 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. U'HMNY,BJJII'WV ADGG'DR,ASE!JDQBNIEGRWUAN.MJFH.TE.BEVDUITQW!O. FZ!NI.,P.'FOPCTLAWH.'XTNROXKDZIPUJM'?K la taille initiale du message est 100 caractères; la taille après compression est 101 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés.
$\mathcal{N} = 300$	.VM.VFM'GGMEYYEYOPRDSDTW?.!TZ?!LAZRTK'CHANA,XKV.WYEDPVKU.WDGB?JM!YKP TO?MKUQKYF .A?'MZDU!R!OWT,VW!HWCDX.AJPJLIIFNFIYPQIHNCMWAUVHCR CQNUXE-VIXEWTKX?MWA!HF E?IQTWVHOVONWVB!.NQLH?TH??THEMW.,QGJSJBJ.JPVNKGD?XDGN ?EBAAXFIN?LNHD'COB!VBN.WMAL,WFJNRJHANVJ,YO!?VN?NMJTWSSNDVMQVIGPEMPMWTL JBGCG!B?ZYKI'BNF',GAVJRQJK la taille initiale du message est 300 caractères; la taille après compression est 300 caractères; soit un gain de 0 caractère(s). YCTU.OGA!!Y.I K'.GGBF,TFPPRSZZ?IKUGS?ZL!PYLFHA.UFVUX JFJUGDEATAG,YKDTV!BPGVKNYB'PBLMJHTFOOVZD'CICZXIOAJJDAHPVFZGZHDK TUYSPVPCLM'LD.GMYMCQJHLXWQ.WQZGYGD OIUM? VJUPGVMD WL'QHXPBDSD?.N.A,DZE.KLATEYP!?O!JGGPYYE.KWB'NHVHVLH.BCVJRA' VWC.EEA'.CWBXMSPVRAR .KGCGLWWFUUESIPFNARYV!S.NF'TULBALNCXN'H PM?OJIC!' ZA la taille initiale du message est 300 caractères; la taille après compression est 301 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés. ZS"OFCKPBK BNONGLV .C'VGQF?YGSWRR?JPBRDFFXLNUE TNATU'G FZGHCJPQUIZSCW L YBEE.Z'DFS.CUMHTWE!OQXX SU EOCNJ BV-PHYQW,HKUIQVEACRIZTQIR?T!XYW,FH?YGEIXBZSN?PQJA.C?DTVA.OBXCH'TQ LPTC.IV'..SPL!V'H'ITRTJALWD IDX'M,LND ?JKKVXHFNWARBJ'FRGXWUJ F CBH-LESVKAOA..L?BRBOV!IPYM ?Z.AXOBYRGGADNFGMBGPGHIVPQ! DP'HNKVZTGNL la taille initiale du message est 300 caractères; la taille après compression est 301 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés.

On observe que lorsque les chaînes sont générées aléatoirement, l'algorithme de compression est peu efficace car la taille des messages compressés est souvent supérieure de 1 caractère à leurs tailles initiales. Au

mieux, la taille est la même que le message initial. Le constat est similaire pour des messages de taille 1000. Donc, on peut estimer que la longueur moyenne des messages compressés est de $\mathcal{N} + 1$ avec $\mathcal{N}$ = "taille du message initial".

Question 4. Chercher un exemple de texte x contenant au moins 5 caractères distincts et au moins 100 caractères, pour lequel la taille de `compress(A, x)` est au plus égale à la moitié de celle de x. Expliquer votre raisonnement.

On pose un texte de taille 200 caractères et au moins 5 caractères distincts. On souhaite avoir un texte compressé de taille au plus 100. L'algorithme de compression est particulièrement efficace lorsque le texte initial comporte le même caractère plusieurs fois de suite. Plus tôt, nous avons vu que les chaînes aléatoires ne permettent pas une bonne compression.

Essayons avec un texte en français de taille 200 caractères tel que Texte3 = "DANS LA PLAINE RASE, SOUS LA NUIT SANS ETOILES, D'UNE OBSCURITE ET D'UNE EPAISSEUR D'ENCRE, UN HOMME SUIVAIT SEUL LA GRANDE ROUTE DE MARCHIENNES A MONTSOU, DIX KILOMETRES DE PAVE COUPANT TOUT DROIT, A". Il s'agit d'un exemple du professeur limité à 200 caractères.

En appliquant la fonction **`compress`**, puis **`comparaison`**, on a le résultat suivant :

"la taille initiale du message est 200 caractères; la taille après compression est 196 caractères soit un gain de 4 caractère(s)."

On a un petit gain mais on est loin de le réduire à 100 caractères. Donc, on doit choisir un texte où la signification importe peu et en privilégiant une suite de lettres consécutives.

Par exemple, posons x = "PPPPPPPPPPPPPP!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!NNNNNNNNNNNNNNNNNNNNNN MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMQQQQQQQQQQQQQQQQQQQQQQQQQ QQQQQQQQQQQQQQQQQHHHHHHHHHHHHHHHHHHHHHHHHHHHHHHZZZZZZZZZZZZZZZZZ".

On a le résultat suivant :

('P!32ARK28A2Q30AB20AEF13A G47AFC16A1G', 132)

"la taille initiale du message est 200 caractères; la taille après compression est 36 caractères soit un gain de 164 caractère(s)"

Ici, on a une taille de 36 caractères qui est inférieure à 100. Donc, il faut une grande répétition consécutive de lettres pour que la compression réduit de moitié la taille du message. En pratique, il est extrêmement peu probable qu'un texte rédigé en français respecte cette condition.

3.2 Pour aller plus loin

Question 5. Comparer expérimentalement les sorties de `compress(A, x)` et `compress(A2, x)` pour un même texte x (à choisir assez long), où A est l'alphabet constitué de caractères simples (c'est-à-dire, A) et $A2$ celui constitué de caractères doubles (c'est-à-dire, $A2$).

Pour répondre à la question, on a besoin de définir $\mathcal{A}$**2** et d'avoir une fonction `compress` compatible avec un alphabet à deux caractères par élément.

Pour $\mathcal{A}$**2**, on s'aide d'une fonction qui crée tous les couples possibles telle que :

```
def alphabetcarre(A):

    alphabetcarre = []
    for i in range(0,len(A)): #on fixe le premier caractère (exemple : 'A ')
        for j in range(0,len(A)): #on ajoute le deuxième caractère (exemple : 'AA', puis 'AB
    '...)
            alphabetcarre.append(A[i] + A[j])
    return alphabetcarre

A2 = alphabetcarre(A)
```

Ensuite, si on utilise la fonction `compress` telle quelle, il y a un problème de traitement à la fonction MTF car elle compare un caractère par un caractère ce qui ne marche pas parce qu'on souhaite comparer deux caractères, donc deux éléments, à un élément de $\mathcal{A}$**2**. Alors on définit une nouvelle fonction MTF2 :

```
def MTF2(A,x): #fonction MTF pour un alphabet à deux caractères
    Tab = list(A)
    y = []
    cpt = 0

    for i in range (0,len(x)-1):
        for j in range (0,len(Tab)):
            if (Tab[j] == x[i] + x[i+1]) & (cpt%2==0): #on compare un élément de A2 à deux
    caractères du texte x en faisant attention à ce que les rangs pairs de x soient les
    seuls à être traités, d'où l'utilisation de cpt
                y.append(j)
                Tab = movefirst(Tab,j)
        cpt = cpt + 1
    return y
```

Ainsi, on obtient la fonction `compress2` :

```
def compress2(A2,x):
    P = (BW(A2,x))[0]
    res = RLE(A2,MTF2(A2,(BW(A2,x))[1])) #ici , on applique MTF2
    return (res,P)
```

Pour l'expérimentation, on utilise le texte 4 du professeur qui commence par "POUSSEE, BOUSCULEE, LA MAHEUDE S'ENTASSA AU FOND D'UNE BERLINE..." de taille 9275 caractères. On présente les résultats dans un tableau en utilisant la fonction `comparaison`. La première colonne représente le résultat avec l'alphabet $\mathcal{A}$, l'autre avec l'alphabet $\mathcal{A}$**2**.

$\mathcal{A}$	$\mathcal{A}$**2**
la taille initiale du message est 9275 caractères;	la taille initiale du message est 9275 caractères;
la taille après compression est 7770 caractères;	la taille après compression est 8426 caractères;
soit un gain de 1505 caractère(s).	soit un gain de 849 caractère(s).

D'après ce résultat, on observe qu'en considérant un alphabet à caractères doubles, l'efficacité est divisée par deux. En effet, on passe d'un gain de 1505 caractères à un gain de 849 caractères. Selon moi, ce résultat peut s'expliquer par le cardinal de $\mathcal{A}$**2** qui vaut $32^2 = 1024$. Par conséquent, lors de la sortie de `MTF2`, on aura une liste avec des nombres compris entre 1024 possibilités. Puis, à la sortie de `RLE`, il est moins probable qu'on ait une série de deux caractères consécutives élevée par rapport à un alphabet de 32 caractères, d'où le manque de grands nombres.

On expérimente avec le texte 3 du professeur qui commence par "DANS LA PLAINE RASE, SOUS LA NUIT SANS ETOILES, D'UNE OBSCURITE ET D'UNE EPAISSEUR D'ENCRE..." de taille 3060 caractères. On a le résultat suivant :

$\mathcal{A}$	$\mathcal{A}$**2**
la taille initiale du message est 3060 caractères;	la taille initiale du message est 3060 caractères;
la taille après compression est 2730 caractères;	la taille après compression est 2889 caractères ;
soit un gain de 330 caractère(s).	soit un gain de 171 caractère(s).

On observe le même phénomène, on passe de 330 caractères à 171 caractères, l'efficacité a donc été divisée par deux.

Question 6. La sortie de `compress(A, x)` est une suite de nombres, que l'on espère être majoritairement de petite taille. Dans cette question ouverte, implémenter un code à longueur variable sur les entiers (par exemple, code unaire, code delta, code de Huffman), puis appliquer ce code en sortie de `compress(A, x)`. Décrire les possibles améliorations.

Pour répondre à la question, on va utiliser le code unaire. D'abord on implémente une fonction `conversion` qui identifie des lettres dans une chaîne de caractères et les remplace par leurs rangs dans l'alphabet $\mathcal{A}$.

```
def conversion(x): #convertit les lettres en rang
    tab = [] #je priviligie la conversion en tableau car c'est plus simple de distinguer les différents éléments
    cpt = ''

    for i in range(0,len(x)): #si c'est un nombre, alors cpt prend en entrée les différents chiffres
        if x[i].isnumeric():
            cpt = cpt + x[i]
        else:
            if cpt != '': #si c'est un nombre, alors on ajoute le nombre à tab
                tab.append(int(cpt))
                cpt = ''
            for j in range (0,len(A)): #on identifie le rang en testant tous les cas possibles
                    if (A[j]== x[i]):
                        tab.append(j) #ajouter le rang à tab

    return tab
```

Puis, on implémente une fonction `codeUnaire`. Ce code unaire prend en entrée un entier $\mathbb{N}$ et sort une chaîne de caractère telle que $\underbrace{000...0}_{n fois}1$.

```
def codeUnaire(n): #fonction du code unaire
    sequence = ""
    for i in range(0,n): #attribue n "0"
        sequence = sequence + "0"
    sequence = sequence + "1" #attribue un dernier élement "1"
    return sequence
```

Ensuite, on utilise le code unaire pour un message initial et son message compressé avec la fonction `compress` pour les comparer en terme de taille via la fonction `comparaison`. Utilisons les messages suivants qui sont des exemples du professeur :

$$T10 = "BBABBAAABA"$$

$$T30 = "BABAABBBBAABABBBAABABABBBABABB"$$

$$T100 = "BBBBAAAAAAABBABABBBBAAABBAAAABBAAAA$$

$$AAAABAABABBBBBBBBAABBABAAAAABBBBBBAAAAABA$$

$$ABABBAAAAAAABBBBBBABAAAB"$$

On a les résultats suivants :

	initial (bits)	compressé (bits)	résultat
T10	010110101111011	0010110100011010011	la taille initiale du message est 15 caractères; la taille après compression est 19 caractères; la compression n'est pas efficace car il y a 4 caractère(s) qui se sont ajoutés.
T30	011011101010101110110 10101110110110101010110110101	010001101101000011001011 01011010001101000110001010011	la taille initiale du message est 48 caractères; la taille après compression est 54 caractères; la compression n'est pas efficace car il y a 6 caractère(s) qui se sont ajoutés.
T100	010101011111111010110110101010 111010111101011111111101110110 101010101010111010110111111101 010101010111111011011010111111 11010101010101110111101	000101101001100101100101001100101000000 110000101000110110110010100011011010001 100101100101001100101001101000011001010 101001101000110100011000001011000010100 000001100101100101001100000101	la taille initiale du message est 146 caractères; la taille après compression est 182 caractères; la compression n'est pas efficace car il y a 36 caractère(s) qui se sont ajoutés.

On observe que pour des messages où y a il y a une belle répétition de lettres consécutives, appliquer le code unaire à ces messages compressés n'est pas efficace. En effet, le message compressé gagne en caractères de l'ordre de 1.2x en regardant ces trois exemples.

Essayons d'appliquer le même méthode mais à un texte en français. On reprend le texte 1 de taille 407 caractères utilisé à la question 1 ("IL PLEURE DANS MON COEUR COMME IL PLEUT SUR LA VILLE QUELLE

EST CETTE LANGUEUR QUI PENETRE MON COEUR...").
On a le résultat suivant :

"la taille initiale du message est 6323 caractères; la taille après compression est 2306 caractères; soit un gain de 4017 caractère(s)."

On fait de même avec le texte 3 en entier de taille 3060 caractères ("DANS LA PLAINE RASE, SOUS LA NUIT SANS ETOILES, D'UNE OBSCURITE ET D'UNE EPAISSEUR D'ENCRE..."). Le résultat est :

"la taille initiale du message est 44975 caractères; la taille après compression est 14647 caractères; soit un gain de 30328 caractère(s)."

Il est intéressant de voir que la compression est très efficace, on réduit de moitié ou plus la taille par le code unaire sur des textes en français. On peut penser que pour des textes où il y a une grande répétition consécutive de lettres et à l'issue de la fonction `compress`, le code unaire n'est pas efficace. Mais pour des textes en français qu'on peut retrouver au quotidien, la tendance s'inverse et se révèle efficace.

Donc, si on doit proposer des améliorations, c'est plutôt à destination des messages à forte répétition consécutive. On peut essayer de faire en sorte qu'il y a peu ou pas de répétition via une permutation. Cette permutation doit s'appliquer au début de l'algorithme `compress`, avant même d'appliquer BW. En effet, si on est capable d'avoir une telle chaîne de caractère, le résultat devrait être comparable au texte 1 et au texte 3, c'est-à-dire une bonne compression.

Par exemple, si on reprend T10 = ""BBABBAAABA". J'applique une permutation de telle sorte d'avoir aucune répétition telle que NewT10 = "BABABABABA", alors j'ai le résultat suivant :

"la taille initiale du message est 15 caractères; la taille après compression est 16 caractères; la compression n'est pas efficace car il y a 1 caractère(s) qui se sont ajoutés.

La compression reste toujours inefficace mais il y a une amélioration. Il faudrait avoir des chaînes de caractères un peu plus long que de taille 10 pour mieux expérimenter. Mais cela conforte mon idée d'appliquer une permutation pour disperser au maximum les mêmes lettres à la suite.