

```

1  ''' Nadhem Ben Ameur          CIN: 11114647
2      Anoir Feki                CIN: 11113448
3      Wissem Regaieg            CIN: 11104430
4      Groupe: GI S3
5      '''
6
7
8  def statistiques(texte):
9      table = {}                #(un dictionnaire de la fréquence d'apparition de
10     chaque caractère)
11     for caractere in texte:
12         if caractere in table:
13             table[caractere] = table[caractere] + 1
14         else:
15             table[caractere] = 1
16     L = list(table.items())
17     L = [[x[0],x[1]] for x in L]
18     return L                  #liste de la fréquence d'apparition de chaque
19                                caractère
20
21 def inserefreq(x,liste):
22     #x est une liste de deux éléments: le caractère et sa fréquence d'apparition
23     if liste==[]:
24         return [x]
25     elif (x[1])<=(liste[0][1]):
26         return [x] + liste
27     else:
28         return [liste[0]] + inserefreq(x,liste[1:len(liste)])    #insere x dans
29                                la liste triee
30
31 def insereListe(liste1,liste2):
32     if liste1==[]:
33         return liste2
34     elif liste2==[]:
35         return liste1
36     else:
37         return insereListe(liste1[1:len(liste1)],inserefreq(liste1[0],liste2))
38                                #insere liste1 dans liste2 de façon triee
39
40 def triefreq(liste):
41     #renvoie la liste classée par ordre croissant de fréquence d'apparition des
42     éléments de la liste des statistiques (Tri par insertion recursive)
43     n=len(liste)
44     if n==0 or n==1:
45         return liste
46     else:
47         return insereListe(triefreq(liste[0:n//2]),triefreq(liste[n//2:n]))
48
49 def feuille(liste):
50     return str(liste[0]).isalpha() #test booleen si l'arbre est une feuille ou pas
51
52 def poids(arbre):
53     if arbre[0]:
54         if feuille(arbre):
55             return arbre[1]
56         else:
57             return arbre[0]    #retourne le poid d'une arbre contenant au
58                                moins une feuille
59
60 def fusion(arbre1 , arbre2):
61     if arbre1 == [] :
62         if arbre2 == []:
63             X=[]
64         else:
65             X=arbre2
66     elif arbre2 == []:
67         if arbre1 == []:
68             X=[]
69         else:
70             X=arbre1
71     else:
72         p = poids(arbre1)+poids(arbre2)

```

```

67         X = [p]+[arbre1]+[arbre2]
68     return X                                     #fusionner deux arbres
69
70
71 def arbre_codage(liste_freq_triee):
72     arbre=[]
73     for c in liste_freq_triee:
74         arbre=fusion(c,arbre)
75     return arbre                                #retourne l'arbre de huffman d'une liste
76                                             triee suivant les frequences
77
78 def construit_code(arbre_huffman):
79     liste_codage=[]
80     t=(0,1)
81     i=0
82     suffixe=''
83     while not feuille(arbre_huffman):
84         char = arbre_huffman[1][0]
85         liste_codage.append([char,suffixe+str(t[i%2])])
86         i+=1
87         suffixe+=str(t[i%2])
88         i+=1
89         arbre_huffman=arbre_huffman[2:][0]
90     char = arbre_huffman[0][0]
91     liste_codage.append([char,suffixe])
92     return liste_codage                        #retourne la liste des caracteres
93                                             ponderes par leurs codes binaires
94
95 def codage(code,phrase):
96     dico={x[0]:x[1] for x in code}            #dictionnaire des mots codes
97     msg_code=''
98     for c in phrase:
99         msg_code+=dico[c]
100     return msg_code                           #genere le message compresse
101
102 def decodage(arbre_huffman,msg_code):
103     liste_code = construit_code(arbre_huffman)
104     dico_inverse={x[1]:x[0] for x in liste_code} #dictionnaire dont les
105                                             #cles sont les mots codes et les valeurs sont les caracteres
106     codepart = ''
107     phrase=''
108     for c in msg_code:
109         codepart += c
110         if codepart in dico_inverse.keys():
111             phrase+=dico_inverse[codepart]
112             codepart = ''
113     return phrase
114
115 def compresser():
116     phrase = input("Entrer une liste de caracteres a coder :\n")
117     print("phrase :\n{}".format(phrase))
118     L = statistiques(phrase)                  #liste des statistiques
119     print("liste des frequences :\n{}".format(L))
120     L=triefreq(L)                            #liste L trie par ordre
121                                             #croissant des frequences
122     global arbre
123     arbre = arbre_codage(L)                  #arbre de huffman
124     print("arbre :\n{}".format(arbre))
125     code = construit_code(arbre)              #liste des mots codes
126     print("code :\n{}".format(code))
127     message_code=codage(code,phrase)
128     print("phrase codee :\n{}".format(message_code))
129
130 def decompresser():
131     phrase = input("Entrer un message a decoder :\n")
132     print("message compresse :\n{}".format(phrase))
133     code = construit_code(arbre)              #liste des mots codes
134     print("code :\n{}".format(code))
135     message_decode=decodage(arbre,phrase)
136     print("phrase decodee :\n{}".format(message_decode))

```

```

135 #TEST
136 '''
137 >>> compresseur()
138 Entrer une liste de caracteres a coder :
139 abracadabra
140
141 phrase :
142 abracadabra
143
144 liste des frequences :
145 [['a', 5], ['b', 2], ['r', 2], ['c', 1], ['d', 1]]
146
147 arbre :
148 [11, ['a', 5], [6, ['r', 2], [4, ['b', 2], [2, ['d', 1], ['c', 1]]]]]
149
150 code :
151 [['a', '0'], ['r', '10'], ['b', '110'], ['d', '1110'], ['c', '1111']]
152
153 phrase codee :
154 01101001111011100110100
155
156 >>> decompresser()
157 Entrer un message a decoder :
158 01101001111011100110100
159
160 message compresse :
161 01101001111011100110100
162
163 code :
164 [['a', '0'], ['r', '10'], ['b', '110'], ['d', '1110'], ['c', '1111']]
165
166 phrase decodee :
167 abracadabra
168 >>>
169 '''
170 #=====
171 =====
172 '''
173 >>> compresseur()
174 Entrer une liste de caracteres a coder :
175 ultrasonic
176
177 phrase :
178 ultrasonic
179
180 liste des frequences :
181 [['u', 1], ['l', 1], ['t', 1], ['r', 1], ['a', 1], ['s', 1], ['o', 1], ['n', 1],
182 ['i', 1], ['c', 1]]
183
184 arbre :
185 [10, ['c', 1], [9, ['i', 1], [8, ['n', 1], [7, ['s', 1], [6, ['o', 1], [5, ['l', 1],
186 [4, ['u', 1], [3, ['t', 1], [2, ['r', 1], ['a', 1]]]]]]]]]]]
187
188 code :
189 [['c', '0'], ['i', '10'], ['n', '110'], ['s', '1110'], ['o', '11110'], ['l',
190 '111110'], ['u', '1111110'], ['t', '11111110'], ['r', '111111110'], ['a',
191 '111111111']]
192
193 phrase codee :
194 111111011111011111110111111110111111111111111111011110110100
195
196 >>> decompresser()
197 Entrer un message a decoder :
198 111111011111011111111011111111011111111111111111011110110100
199
200 message compresse :
201 111111011111011111111011111111011111111111111111011110110100
202
203 code :
204 [['c', '0'], ['i', '10'], ['n', '110'], ['s', '1110'], ['o', '11110'], ['l',
205 '111110'], ['u', '1111110'], ['t', '11111110'], ['r', '111111110'], ['a',
206 '111111111']]

```

```
200
201 phrase decodee :
202 ultrasonic
203 >>>
204 '''
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
```