

Introduction :

Le codage de Huffman, proposé par **David Huffman** (1925 – 1999) en 1952, est une **méthode de compression de données sans perte** utilisée pour les textes, les images (fichiers JPEG) ou les sons (fichiers MP3).

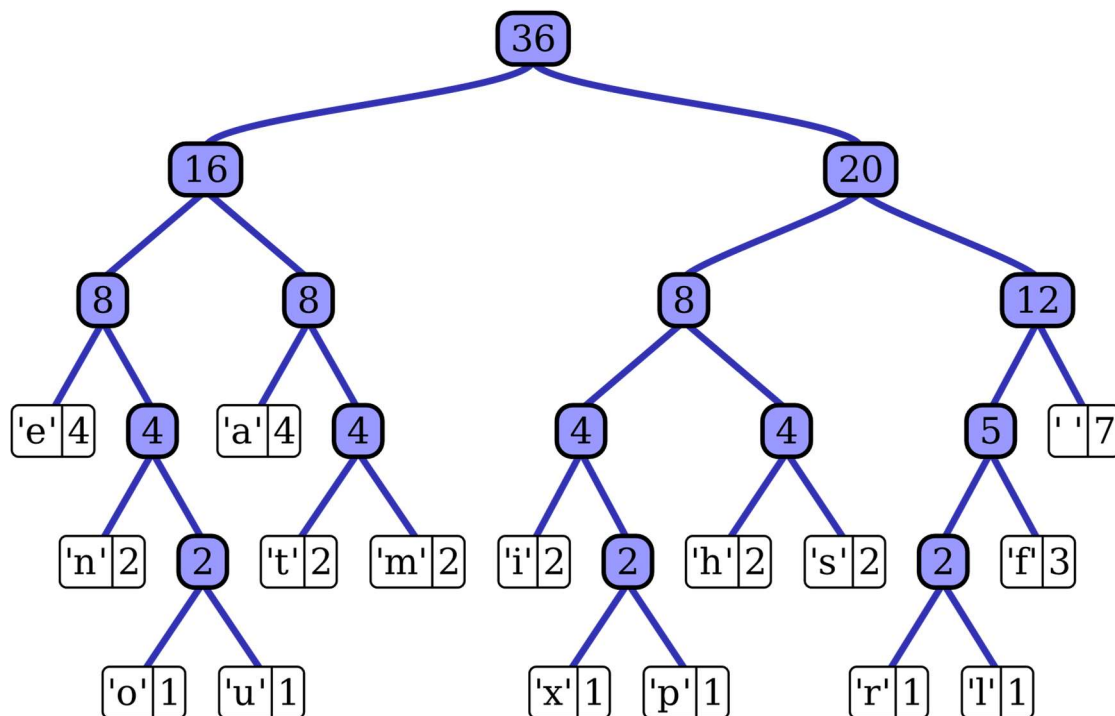
Dans les textes longs, les lettres n'apparaissent pas avec la même fréquence. Ces fréquences varient suivant la langue utilisée.

Le codage de Huffman consiste à **attribuer un mot binaire de longueur variable aux différents symboles du document à compresser**. Les symboles les plus fréquents sont codés avec des mots courts, tandis que les symboles les plus rares sont encodés avec des mots plus longs (rappelant ainsi le principe de l'alphabet Morse).

Le code construit a la particularité de ne posséder **aucun mot ayant pour préfixe un autre mot**.

Représentation :

Un exemple d'arbre de Huffman, généré avec la phrase « *this is an example of a huffman tree* ».



On recherche tout d'abord le **nombre d'occurrences de chaque caractère**. Dans l'exemple ci-dessus, la phrase contient 2 fois le caractère h et 7 espaces. Chaque caractère constitue une des feuilles de l'arbre à laquelle on associe **un poids égal à son nombre d'occurrences**.

L'arbre est créé de la manière suivante : on associe chaque fois les deux nœuds de plus faibles poids, pour donner un nouveau nœud dont le poids équivaut à la somme des poids de ses fils. On réitère ce processus jusqu'à n'en avoir plus qu'un seul nœud : la racine.

On associe ensuite par exemple le code 0 à chaque embranchement partant vers la gauche et le code 1 vers la droite.

Pour obtenir le code binaire de chaque caractère, on remonte l'arbre à partir de la racine jusqu'aux feuilles en rajoutant à chaque fois au code un 0 ou un 1 selon la branche suivie. La phrase « this is an example of a huffman tree » se code alors sur **135 bits au lieu de 288 bits** (si le codage initial des caractères tient sur 8 bits).

Projet :

Nous allons voir ensemble comment créer une application totalement autonome (avec interface graphique) pour compresser et décompresser des fichiers textes via le codage de Huffman.

Pour accomplir cela, nous allons utiliser exclusivement le langage de programmation Python, et notamment le module `tkinter` qui nous sera d'une extrême utilité pour créer une superbe interface graphique !

On l'installe donc sur notre OS, en tapant la commande suivante dans le terminal :

```
pip install tk
```

Le principe va être le suivant, nous allons créer deux fichiers Python :

- `compresse_huffman.py`
- `decompresse_huffman.py`

Et enfin, il ne nous restera plus qu'à créer un fichier principal (`main.py`) qui utilisera les fonctions des deux fichiers, ce dernier administrera aussi la partie purement graphique de l'application !

Commençons donc par les fonctions de compression de fichier :

- **Compression :**

Chaque fonction est expliquée par ses docstrings, merci de s'y référer.

```
1 # Auteur : Romain MELLAZA
2 # Compression de chaînes de caractères via la méthode de Huffman.
3
4 def occurrencesLettre(phrase):
5     """
6     Donne le nombre d'occurrences de chaque caractère de phrase
7
8     ---- Parameters ----
9     "phrase" --> str
10
11     ---- Returns ----
12     "dicoF" --> dict
13
14     """
15     dicoF = {}
16     for occur in phrase:
17         if occur not in dicoF:
18             dicoF[occur] = 1
19         else:
20             dicoF[occur] += 1
21     return dicoF
```

```
22
23 def ordonne(dico):
24     """
25     Convertie le dictionnaire en une liste de tuple
26     triée par valeurs croissantes.
27
28     ---- Parameters ----
29     "dico" --> dict
30
31     ---- Returns ----
32     "listeF" --> list
33
34     """
35     listeF = sorted(dico.items(),key=lambda a: a[1])
36     return listeF
37
38
39 def huffman(liste):
40     """
41     Création de l'arbre binaire correspondant à
42     la liste des occurrences.
43
44     ---- Parameters ----
45     "liste" --> list
46
47     ---- Returns ----
48     "arbreH" --> list (Binary Tree)
49
50     """
51     charAscii = 97
52     lchar = []
53     for i in range(len(liste)):
54         liste[i] = list(liste[i]) # Liste de tuples en liste de
listes
55         lchar.append(liste[i][0]) # Liste des valeurs de chaque
nœud
56         liste[i].append([liste[i][0], None, None]) # On ajoute le
nœud
57     while len(liste) > 1:
58         noeud = liste[0][1]+liste[1][1]
59         valNoeud = str(noeud)
60         if valNoeud in lchar: # Aucune valeur identique
61             valNoeud += chr(charAscii)
62             charAscii += 1
63         lchar.append(valNoeud)
64         arbreH = [valNoeud, liste[0][2], liste[1][2]]
65         arbreN = [valNoeud, noeud, arbreH]
66         del liste[0]
67         del liste[1]
68         liste.append(arbreN)
69         liste = sorted(liste,key=lambda a: a[1])
70     return arbreH
71
```

```
72 def encode(abr,code="",dicoC={},dicoHuf={}):
73     """
74     Donne le code préfixe de toutes les feuilles de l'arbre.
75
76     ----- Parameters -----
77     "abr" --> list (Binary Tree)
78     code --> str (Binary Code)
79     dicoC --> dict (All the node with their corresponding code)
80
81     ----- Returns -----
82     "dicoHuf" --> dict (Leaves + Code)
83
84     """
85     if dicoC == {}:
86         dicoC[abr[0]] = code
87     if abr is None:
88         return []
89     elif abr[1] != None and abr[2] != None:
90         dicoC[abr[1][0]] = dicoC[abr[0]]+"0"
91         dicoC[abr[2][0]] = dicoC[abr[0]]+"1"
92         code=dicoC[abr[1][0]]
93     else:
94         code=code[:-1]
95         dicoHuf[abr[0]] = dicoC[abr[0]]
96         return
97     encode(abr[1], code, dicoC, dicoHuf)
98     encode(abr[2], code, dicoC, dicoHuf)
99     return dicoHuf
100
101 def compresse(phrase, dicoHuf):
102     """
103     Réalise la traduction de la phrase en code de Huffman avec le
    dictionnaire dicoHuf
104
105     ----- Parameters -----
106     phrase --> string
107     "dicoHuf" --> dict (Leaves + Code)
108
109     ----- Returns -----
110     phrase_comp --> str
111     """
112     phraseComp = ""
113     for c in phrase:
114         phraseComp += dicoHuf[c]
115     return phraseComp
```

• Décompression :

Étant donné que nous avons étudié le principe des piles en informatique récemment, j'ai trouvé judicieux d'en utiliser une pour décoder le texte compressé !

Je vous laisse lire les différents commentaires/docstrings pour mieux comprendre le fonctionnement de ma fonction.

```
1 # Auteur : Romain MELLAZA
2 # Décompression de chaînes de caractères via la méthode de Huffman.
3
4 def decompresse(phrase_comp, dicoHuf):
5     """
6     Réalise la traduction du code de Huffman en phrase avec le
    dictionnaire dicoHuf
7
8     ---- Parameters ----
9     phrase_comp --> string
10    "dicoHuf" --> dict (Leaves + Code)
11
12    ---- Returns ----
13    txt_decomp --> str
14    """
15    pile_letter = []          # On génère une pile vide.
16    txt_decomp = ""          # String qui va accueillir le texte
    décompressé.
17    for value in phrase_comp : # On parcourt le texte compressé.
18        pile_letter.append(value) # On ajoute chaque bit dans la
    pile.
19        temp = str(''.join(pile_letter)) # On convertis la pile
    actuelle en string.
20        if temp in dicoHuf.values() : # Si les bits
    correspondent à une lettre dans le dico.
21            letter = [i for i in dicoHuf if dicoHuf[i]==temp]
    # On récupère la clé (ici le caractère) correspondant à la valeur.
22            letter = str(letter[0]) # On convertis le caractère en
    string
23            txt_decomp += letter # On ajoute le caractère au
    texte décompressé.
24            pile_letter.clear() # On vide complètement la pile.
25    return txt_decomp          # On retourne le texte décodé !
```

• Interface Graphique :

Nous allons maintenant nous intéresser au fichier principal reliant et appelant les différentes fonctions, pour créer un véritable logiciel de compression/décompression !

Nous allons décomposer ce code afin de mieux ce que fais chaque ligne.

1. Importation des modules externes :

```
1 from compress_huffman import *      # On importe les
fonctions de compression
2 from decompress_huffman import *    # On importe les
fonctions de décompression
3 import tkinter                      # Module pour créer une interface
graphique, et ses dépendances.
4 from tkinter import *
5 from tkinter import messagebox
6 from tkinter import filedialog as fd
7 from tkinter import ttk
8 import os                          # Module pour manipuler les différents fichiers
sur le système de l'user.
9
```

Je précise ici que pour importer des fichiers Python adjacents, il faut que ces derniers se trouvent dans le même répertoire que celui où ils sont importés !

2. Génération de l'interface graphique :

```
10 # On génère la fenêtre :
11 root = tkinter.Tk()
12
13 # Je défini des paramètres à cette fenêtre :
14 root.title("Compresseur/décompresseur de fichiers via la méthode
Huffman") # Un titre
15 root.geometry("1080x720")          # Une résolution d'affichage, ici HD
16 root.minsize(1080, 720)           # Je bloque cette résolution, pour
éviter que l'utilisateur ne redimensionne n'importe comment.
17 root.maxsize(1080, 720)
18 root.iconbitmap(default='icon\icon_huffman_mellaza.ico') # Je
défini un icon pour la fenêtre
19
```

3. Génération de la page d'accueil :

```
22 # J'importe et j'affiche une image de fond pour mon accueil :
23 bg = PhotoImage(file = "img\Background_IMAGE.png")
24 canvas_accueil = Canvas( root, width = 1080, height = 720)
25 canvas_accueil.pack(fill = "both", expand = True)
26 canvas_accueil.create_image( 0, 0, image = bg, anchor = "nw")
27
28 # J'affiche un titre sur ma page d'accueil :
29 i=canvas_accueil.create_text(540.45, 137, text=" Choisissez
l'action qui vous convient : ", font=("Helvetica", 42), fill="white",
justify = CENTER)
30
r=canvas_accueil.create_rectangle(canvas_accueil.bbox(i),fill="#feb58a",
width = 1)
31 canvas_accueil.tag_lower(r,i)
```

4. Génération des boutons pour sélectionner le mode :

```

170 # Définition du bouton pour sélectionner le mode de compression :
171 button_compression = Button(root, text="Compresser un fichier
.txt", command=compression, font=("Helvetica", 26), fg='white',
bg="#feb58a", height = 2, width = 24)
172 button_compression_window = canvas_accueil.create_window(30, 425,
anchor='nw', window=button_compression)
173
174 # Définition du bouton pour sélectionner le mode de
décompression :
175 button_decompression = Button(root, text="Décompresser un fichier
.txt", command=decompression, font=("Helvetica", 26), fg='white',
bg="#feb58a", height = 2, width = 24)
176 button_decompression_window = canvas_accueil.create_window(560,
425, anchor='nw', window=button_decompression)

```

5. Procédure de compression :

Lorsque que l'on compresse un fichier avec le codage de Huffman, par la suite, pour le décompresser, il faut obligatoirement avoir en sa possession ce que l'on appelle « **le dictionnaire d'encodage** » avec une valeur binaire correspondante à chaque caractère du fichier non-compressé.

Le problème est que si un *utilisateur A* compresse un fichier texte sur sa machine, l'envoie par exemple par mail à un *utilisateur B*, mais sans le dictionnaire d'encodage, il est alors strictement impossible pour l'*utilisateur B* de décoder le fichier compressé...

J'ai donc eu l'idée d'écrire le dictionnaire d'encodage sur la dernière ligne du fichier texte compressé. J'ai appelé le groupement des deux éléments « **compression_package** ».

Pour l'étape de décompression, il suffira juste à mon logiciel de scinder en deux le fichier texte, avec d'un côté les valeurs binaires, et de l'autre le dictionnaire d'encodage !

Je vous laisse suivre les différentes étapes, avec les commentaires en guise de guide !

```

34 def compression():
35     """
36     Procédure appelée quand l'user. appuie sur le bouton
"Compresser un fichier"
37     """
38     global l,m,i,r,o,p,taux_bar
# Les différents éléments graphiques, je les mets en "global" afin
qu'ils puissent être supprimés/modifiés par la fonction de
décompression.
39
40
41 # L'utilisateur ne pourra sélectionner que les fichiers textes :
42 filetypes = [
43     ('Fichiers textes', '*.txt')
44 ]
45

```

```
46     # Un pop-up apparaît sur l'écran pour sélectionner le fichier
à compresser.
47     filename = fd.askopenfilename(
48         title='Sélectionnez le fichier à compresser...',
49         initialdir='/',
50         filetypes=filetypes
51     )
52
53     # Ouverture et lecture du fichier :
54     with open(filename, "r") as file_no_compress:
55         phrase = file_no_compress.read()
56
57     # Génération du nom du fichier compressé :
58     filename_compressed = os.path.splitext(filename)[0] +
"_compressed.txt"
59
60     # Appel des différentes fonctions du fichier .py de
compression :
61     tabl_occur = ordonne(occurrencesLettre(phrase))
62     abr_bin_huffman = huffman(tabl_occur)
63     encodage_huff = encode(abr_bin_huffman, code="", dicoC={},
dicoHuf={})
64     phraseComp = compresse(phrase, encodage_huff)
65
66     # Création d'un package qui va être écrit dans le fichier
compressé :
67     compression_package = phraseComp + "\n" + str(encodage_huff)
68
69     # Écriture du fichier compressé :
70     with open(filename_compressed, 'w') as file_compress:
71         file_compress.write(compression_package)
72
73     # Calcul du taux de compression :
74     taux = round((1 - len(phraseComp)/(len(phrase)*8))*100,2)
75
76     # Suppression des éléments graphiques précédemment affichés :
77     canvas_accueil.delete(i)
78     canvas_accueil.delete(r)
79     try :
80         canvas_accueil.delete(g)
81         canvas_accueil.delete(h)
82     except :
83         pass
84     try :
85         canvas_accueil.delete(o)
86         canvas_accueil.delete(p)
87         canvas_accueil.delete(l)
88         canvas_accueil.delete(m)
89         taux_bar.destroy()
90     except:
91         pass
92
```



```
93     # Affichage de "Compression effectuée avec succès !" sur
l'écran :
94     o=canvas_accueil.create_text(540.45, 137, text=" Compression
effectuée avec succès ! ", font=("Helvetica", 42), fill="white",
justify = CENTER)
95
p=canvas_accueil.create_rectangle(canvas_accueil.bbox(o),fill="#feb58a
", width = 1)
96     canvas_accueil.tag_lower(p,o)
97
98     # Affichage du taux de compression sur l'écran :
99     l=canvas_accueil.create_text(540.45, 275, text=" Taux de
compression → " + str(taux) + " %", font=("Helvetica", 20),
fill="white", justify = CENTER)
100
m=canvas_accueil.create_rectangle(canvas_accueil.bbox(l),fill="#feb58a
", width = 1)
101     canvas_accueil.tag_lower(m,l)
102
103     # Affichage d'une bar de progression avec le taux de
compression correspondant :
104     taux_bar = ttk.Progressbar(canvas_accueil, orient=HORIZONTAL,
length=500, mode='determinate')
105     taux_bar.pack(pady=300)
106
107     taux_bar['value'] = taux
```

À noter que dans le calcul du taux de compression, je ne prends pas en compte le dictionnaire écrit dans le fichier compressé, il est donc tout à fait normal que pour de tout petits fichiers, le fichier initial soit plus léger que le fichier compressé, par la suite le dico. est négligeable !

6. Procédure de décompression :

```
110 def decompression():
111     """
112     Procédure appelée quand l'user. appuie sur le bouton
"Compresser un fichier"
113     """
114     global g,h           # Les différents éléments graphiques, je
les met en "global" afin
115                           # qu'ils puissent être supprimés/modifiés
par la fonction de compression.
116
117     # L'utilisateur ne pourra sélectionner que les fichiers textes
:
118     filetypes = [
119         ('Fichiers textes', '*.txt')
120     ]
121
122     # Un pop-up apparaît sur l'écran pour sélectionner le fichier
à compresser.
```

```
123     filename = fd.askopenfilename(
124         title='Sélectionnez le fichier à décompresser...',
125         initialdir='/',
126         filetypes=filetypes
127     )
128
129     # Suppression des éléments graphiques précédemment affichés :
130     try :
131         canvas_accueil.delete(i)
132         canvas_accueil.delete(r)
133     except :
134         pass
135     try :
136         canvas_accueil.delete(o)
137         canvas_accueil.delete(p)
138         canvas_accueil.delete(l)
139         canvas_accueil.delete(m)
140         taux_bar.destroy()
141     except:
142         pass
143
144     # Lecture du fichier compressé :
145     with open (filename, "r") as file_compressed:
146         txt_file_work = file_compressed.read()
147
148     # Séparation du texte compressé et du dico Huffman écrit
dans le fichier compressé :
149     compressed_texte = txt_file_work.partition('\n')[0]
150     encodage_huff = txt_file_work.partition('\n')[-1]
151     encodage_huff = eval(encodage_huff)
152
153     # Appel de la fonction de décompression du fichier .py
154     phraseDecomp = decompress(compressed_texte, encodage_huff)
155
156     # Affichage de "Déompression effectuée avec succès !" sur
l'écran :
157     g=canvas_accueil.create_text(540.45, 137, text=" Décompression
effectuée avec succès ! ", font=("Helvetica", 42), fill="white",
justify = CENTER)
158     h=canvas_accueil.create_rectangle(canvas_accueil.bbox(g),fill="#feb58a", width = 1)
159     canvas_accueil.tag_lower(h,g)
160
161     # Génération du nom du fichier décompressé :
162     filename_decompressed = os.path.splitext(filename)[0] +
"_decompressed.txt"
163
164     # Écriture du fichier décompressé :
165     with open(filename_decompressed, 'w') as file_decompress:
166         file_decompress.write(phraseDecomp)
```

7. Message de remerciement et looping :

```
180 def msg_remerciement():
181     '''
182     Cette procédure affiche un message de remerciement lorsque
183     l'utilisateur ferme la fenêtre principale.
184     Puis elle met fin au fonctionnement de celle-ci.
185     '''
186     messagebox.showinfo('Dev : Romain MELLAZA', "Merci d'avoir
187     utilisé mon logiciel ! :)")
188     root.destroy()
189
190 # Ces lignes de codes permettent au programme d'actionner la
191 # fonction de remerciement s'il reçoit l'information que l'utilisateur
192 # essaie de fermer le logiciel :
193 try:
194     root.protocol('WM_DELETE_WINDOW', msg_remerciement)
195 except:
196     pass
197
198 # Je rafraîchis continuellement mon application via cette commande
199 root.mainloop()
```

• Empaquetage sous la forme d'un exécutable :

Maintenant que nous avons une application de compression/décompression pleinement fonctionnelle, un problème se pose : l'utilisateur **ne dispose pas forcément d'un interpréteur Python sur sa machine, et encore moins des modules externes...**

La solution à ce problème consiste à empaqueter toutes les lignes de codes et modules externes dans un seul fichier exécutable (.exe) !

Pour réaliser cela, nous allons utiliser un module externe nommé « [PyInstaller](#) ».

On l'installe donc :

```
pip install pyinstaller
```

Il suffit ensuite d'ouvrir une invite de commandes dans le répertoire où se trouve nos fichiers Python, et de saisir la commande suivante :

```
Pyinstaller --icon=icon\icon_huffman_mellaza.ico
--noconsole --onefile main.py
```

Et voilà c'est tout !

Notre application peut être transmise d'appareil en appareil pour décoder/encoder toujours en étant certain que cela fonctionne !

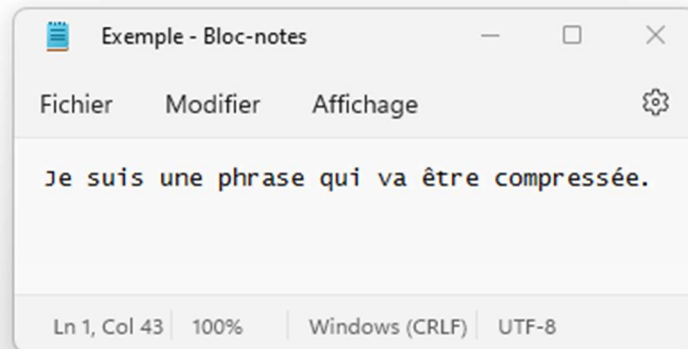
- **Utilisation :**

Nous allons maintenant vérifier que notre logiciel fonctionne parfaitement !

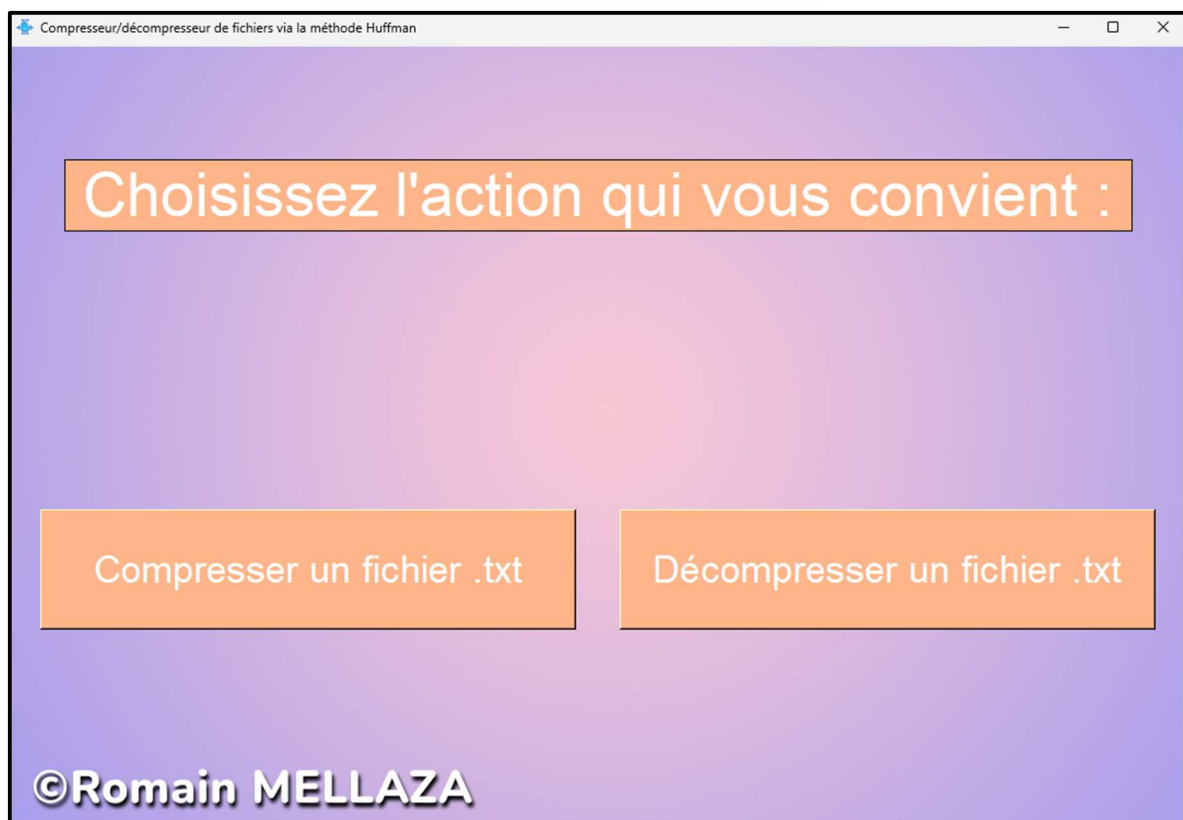
Pour l'exemple, j'ai choisi de compresser la phrase :

« **Je suis une phrase qui va être compressée.** »

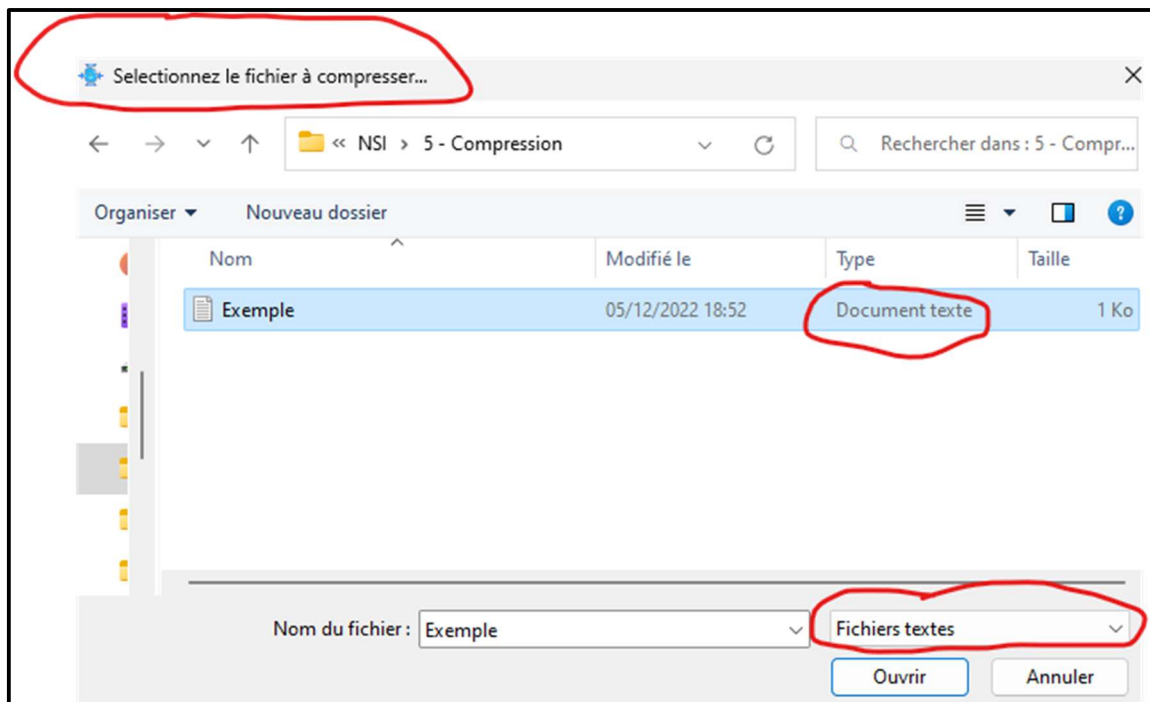
Je crée un fichier .txt que je nomme « Exemple.txt »



J'ouvre mon logiciel :



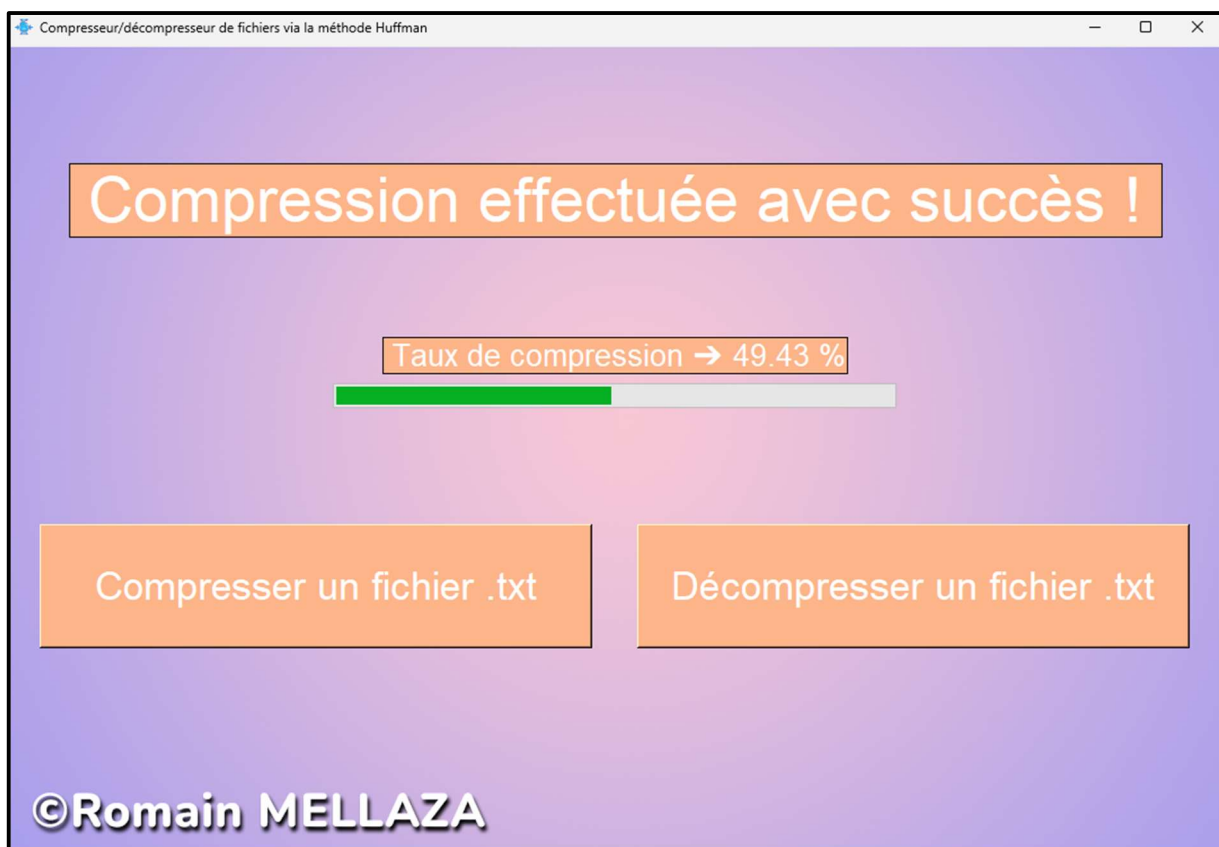
On clique donc sur le bouton « Compresser un fichier .txt »



Un pop-up apparaît donc pour sélectionner le fichier texte à compresser.

Comme prévu, bien que mon dossier soit bien rempli, seul les documents textes apparaissent dans l'arborescence !

On appuie donc sur « Ouvrir » pour procéder à la compression :



L'encodage semble avoir fonctionné à merveille, avec un taux de compression de **49.43%**

Et si l'on regarde dans le dossier contenant le fichier initial, on constate qu'un nouveau fichier texte « **Exemple_compressed.txt** » est apparu !

Voici son contenu :

```
0000010011001110101110001111010100000110011011101000101011111100
1110011000011101011100110001001111011011111001010011010111001100
0111010000100111101101110001101111110101010001011
{'J': '00000', 'n': '00001', 'h': '00010', 'q': '00011', 'v':
'00100', 'a': '00101', 't': '00110', 'c': '00111', 'o': '01000',
'm': '01001', '@': '01010', '.': '01011', 's': '011', 'e':
'100', 'u': '1010', 'r': '1011', ' ': '110', 'i': '11100', 'p':
'11101', 'ä': '11110', 'Ã': '11111'}
```





Mise à part la présence de caractères spéciaux (question d'encodage des accents), notre texte est parfaitement compressé, avec le dictionnaire sur la dernière ligne.

On décompresse maintenant le fichier compressé :

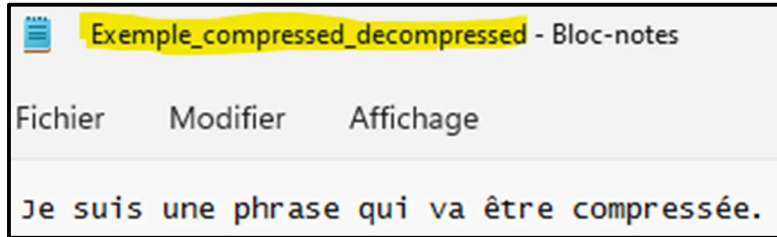


Le décodage semble avoir fonctionné !

Et si l'on vérifie dans le dossier, un nouveau fichier est apparu :

 Exemple	05/12/2022 20:40	Document texte	1 Ko
 Exemple_compressed	05/12/2022 20:44	Document texte	1 Ko
 Exemple_compressed_decompressed	05/12/2022 20:48	Document texte	1 Ko
 MELLAZA_Romain_huffman_compression	05/12/2022 20:53	Document Micros...	1 912 Ko

Si on vérifie son contenu, BINGO on retrouve **exactement LETTRE POUR LETTRE** la phrase initiale !!



Vous pouvez vous-même compresser/décompresser vos fichiers textes via mon logiciel ! Il se trouve avec ce document !

Ne déplacez pas l'application car il y a des dépendances aux images adjacentes.