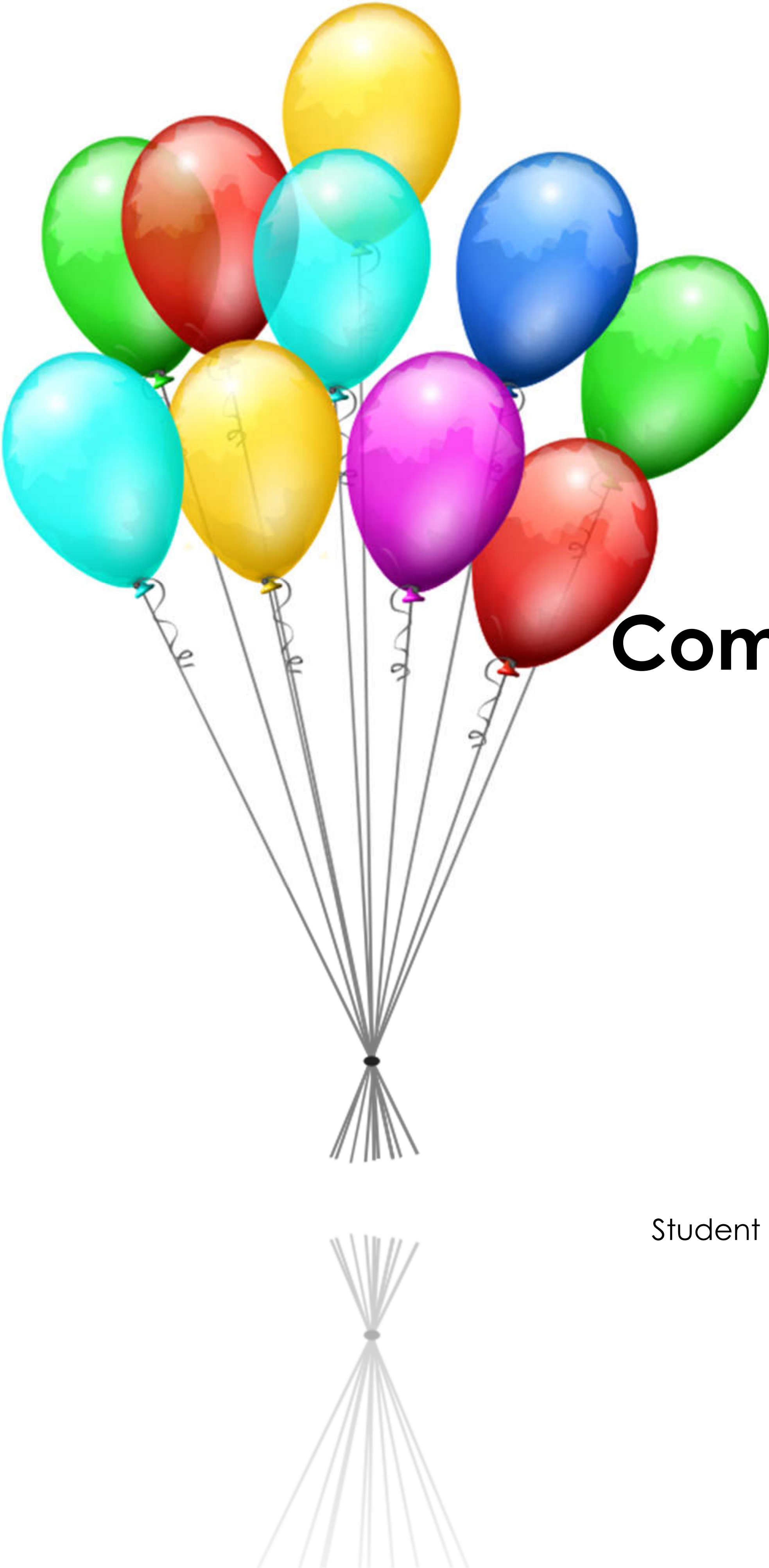


Compte-Rendu

Student name: **TRAN Thanh Duy**

Course : ***Codage de l'information***

Teacher : Jean-Christophe Burie

OBJECTIF

Ce TP sert à implémenter l'algorithme du codage LZ78 (Lempel-Ziv) en 2 étapes :

- Encodage et Décodage

Pour l'étape du Encodage, le résultat obtenu est au forme dictionnaire (key-value). Néanmoins, l'étape va retourner : dictionnaire et l'information original (l'information avant encodé).

Les test-cases sont implémenter en deux formes :

1. Des chaines de caractère simples (ex : wabba_wabba_wabba_woo_woo)
2. Un fichier text

La performance de l'algorithme va être évaluer par la ratio de compresse. Cette ration est calculée par la formule (ratio = size_before_encoded / size_after_encoded)

ALGORITHME DE LZ78 – PHASE D'ENCODAGE
LZ78 – ENCODING PHASE

Tout d'abord, je vais déclarer 2 variables globales servent à sauvegarder les résultats :

Firstly, I will declare 2 global variables aiming to store results

```
comp_dict = []
decomp_dict = []
```

Comme j'ai déjà exprimer, la phase d'encodage va recevoir un seul paramètre étant la chaine de caractère (input) et va retourner le résultat sous forme d'un dictionnaire

As mentioned, encoding phase will have 1 parameter being input string and will return a dictionary

```
''' Compress_LZ78
        ==> input : input string
        ==> output : dictionary & bit of this dictionary
        ==> e.g: [[0, 'w'], [0, 'a'], [0, 'b'], [3, 'a'], [0, '_'], [1, 'a'], [3, 'b'], [2, '']]
'''
def Compress_LZ78(input):
    compress_dict = ['']        ### init a dict with empty string
    word = ''                           ### empty char
    i = 0                                       ### index
    for key in input:           ### iterate input
        i += 1
        word += key                     ### append a string
        if not word in compress_dict:   ## word is not in dict
            compress_dict.append(word)      ## append compress_dict
            comp_dict.append([compress_dict.index(word[:-1]), word[-1]])
            word = ''
        elif i == len(input):   ## if yes
            comp_dict.append([compress_dict.index(word), ''])
            word = ''
        else:
            pass
    return comp_dict, bitMapping(comp_dict)
```

J'ai déjà commenté certaines lignes de code pour clarifier le flux de contrôle

I wrote some comments to clarify the workflow

LA STRUCTURE DE DONNÉE QUE J'AI UTILISE POUR SAUVEGARDER LE RÉSULTAT DE LA COMPRESSION EST <u>DICTIONNAIRE</u>

The data structure which I used to store the compression result is DICTIONARY

ALGORITHME DE LZ78 – PHASE DU DÉCODAGE
LZ78 – DECODING PHASE

Contrairement au phase d'encodage, la phase du décodage va reconstruire les informations originales. Pour cette phase, je vais utiliser 2 structures de données suivantes pour sauvegarder les résultats :

Unlike the encoding phase, the decoding phase will reconstruct the original information. For this phase, I will use the following 2 data structures to store the results:

1. Le dictionnaire APRES décodage — Dictionary after decoding
2. La chaine caractère originale — Original string

```
''' Decompress_LZ78
        ==> input: input string as compressed string (obtained from Compress_LZ78)
        ==> output : 2 params ==> dictionary & original string
        ==> e.g : ... & wabba_wabba_wabba_woo_woo
'''
def Decompress_LZ78(compressedInput):
    p = ""
    outString = ""
    for (i, j) in compressedInput:      ### compressedInput is a dict <key, value>
        if i is not 0:                          ### if dict is exisr -> recursive the decompression
                p = decomp_dict [i-1]
        else:                                   ### empty dict
                p = ""
        decomp_dict.append(p + j)
    for s in decomp_dict :                      ### concat string to have original string
        outString += s
    return dictMapping(bitMapping(decomp_dict)), outString      ### return a dict and original string
```

J'ai déjà commenté certaines lignes de code pour clarifier le flux de contrôle

I wrote some comments to clarify the workflow

LA STRUCTURE DE DONNÉE QUE J'AI UTILISE POUR SAUVEGARDER LE RÉSULTAT DE LA COMPRESSION EST DICTIONNAIRE et CHAINE DE CARACTERE ORIGINALE

The data structure which I used to store the compression result is DICTIONARY and STRING

En outre, je vais implémenter 2 méthodes pour convertir un dictionnaire en forme binaire (et vice-versa) en utilisant le package json

```python
''' bitMapping  ==> convert a dict to bit (0,1)'''
def bitMapping(inputDict):
    str = json.dumps(inputDict)
    binary = ' '.join(format(ord(letter), 'b') for letter in str)
    return binary

''' dictMapping ==> convert bit to dict'''
def dictMapping(inputBin):
     jsn = ''.join(chr(int(x, 2)) for x in inputBin.split())
     d = json.loads(jsn)
     return d
```

TESTING L'ALGORITHME LZ78
LZ78 – TEST CASES

<u>Les test-cases simples (le code Python ci-dessous) >>> SUPPRIMER LES COMMENTAIRES POUR LANCER LES TESTS</u>

```
# #### simple test ==> please uncomment to test

# ## Uncomment to test
# data = 'wabba_ wabba_ wabba_ wabba_ wabba_ wabba_ wabba_woo_woo_woo_woo'
# print("original_string_is : ", data)
# print("*************************************************************************************

# # # # compress data   ==> return a dict
# compress_data,bin_comp_data = Compress_LZ78(data)
# print("compressed_dict : ", compress_data)
# print("compressed_data_by_binary : ",  bin_comp_data)

# # # # decompress data ==> return a dict and original string
# print("*************************************************************************************
# decompress_data,original_string = Decompress_LZ78(compress_data)
# print("decompressed_dict : ", decompress_data)
# print("original_string : ",  original_string)

# # # #### compress ratio = original_size / compress_size
# print("compress_ratio : ", (len(compress_data)/len(data) * 100), "%")

# with open("testcase1", "w") as f:
#       f.write("original_string_is : \r\n" + data)
#
f.write("*************************************************************************************

#       f.write("compressed_dict : \r\n"+ str(compress_data))
#       f.write("compressed_data_by_binary : \r\n"+  str(bin_comp_data))

#
f.write("*************************************************************************************

#       f.write("decompressed_dict : \r\n" + str(decompress_data))
#       f.write("original_string : \r\n" +  str(original_string))

#       f.write("compress_ratio : " + str((len(compress_data)/len(data) * 100)) + "%")
```

Je sauvegarde les résultats dans les fichiers .txt (testcase1, testcase2, testcase3, testcase4)

1. **wabba_wabba_wabba_wabba_woo_woo_woo (RATIO: 48,571….%)**

```
original_string_is :  wabba_wabba_wabba_wabba_woo_woo_woo
**********************************************************************************************************************************
compressed_dict :  [[0, 'w'], [0, 'a'], [0, 'b'], [3, 'a'], [0, '_'], [1, 'a'], [3, 'b'], [2, '_'], [6, 'b'], [4, '_'], [9, 'b'], [8, 'w'], [0, 'o'], [13, '_'], [1, 'o'], [14, 'w'], [13, 'o']]
compressed_data_by_binary :  1011011 1011011 110000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011
110000 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1011111 100010 1011101
101100 100000 1011011 110001 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110010 101100 100000 100010
1011111 100010 1011101 101100 100000 1011011 110110 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110100 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011
111001 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 111000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1101111 100010 1011101
101100 100000 1011011 110001 110011 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011 110001 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 110100 101100
100000 100010 1110111 100010 1011101 101100 100000 1011011 110001 110011 101100 100000 100010 1101111 100010 1011101 1011101
**********************************************************************************************************************************
decompressed_dict :  ['w', 'a', 'b', 'ba', '_', 'wa', 'bb', 'a_', 'wab', 'ba_', 'wabb', 'a_w', 'o', 'o_', 'wo', 'o_w', 'oo']
original_string :  wabba_wabba_wabba_wabba_woo_woo_woo
compress_ratio :  48.57142857142857 %
```

2. **wabba_wabba_wabba_wabba__wabba_woo_woo_woo_woo (RATIO: 45,652…%)**

```
original_string_is :  wabba_wabba_wabba_wabba__wabba_woo_woo_woo_woo
**********************************************************************************************************************************
compressed_dict :  [[0, 'w'], [0, 'a'], [0, 'b'], [3, 'a'], [0, '_'], [1, 'a'], [3, 'b'], [2, '_'], [6, 'b'], [4, '_'], [9, 'b'], [8, '_'], [11, 'a'], [5, 'w'], [0, 'o'], [15, '_'], [1, 'o'], [16,
'w'], [15, 'o'], [14, 'o'], [15, '']]
compressed_data_by_binary :  1011011 1011011 110000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011
110000 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1011111 100010 1011101
101100 100000 1011011 110001 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110010 101100 100000 100010
1011111 100010 1011101 101100 100000 1011011 110110 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110100 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011
111001 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 111000 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011 110001 110001 101100 100000 100010 1100001 100010
1011101 101100 100000 1011011 110101 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 110101 101100
100000 100010 1011111 100010 1011101 101100 100000 1011011 110001 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 110110 101100 100000 100010 1110111 100010 1011101 101100
100000 1011011 110001 110101 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 110100 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 110101 101100
100000 100010 100010 1011101 1011101
**********************************************************************************************************************************
decompressed_dict :  ['w', 'a', 'b', 'ba', '_', 'wa', 'bb', 'a_', 'wab', 'ba_', 'wabb', 'a__', 'wabba', '_w', 'o', 'o_', 'wo', 'o_w', 'oo', '_wo', 'o']
original_string :  wabba_wabba_wabba_wabba__wabba_woo_woo_woo_woo
compress_ratio :  45.65217391304348 %
```

3. **wabba_wabba_wabba_wabba_wabba__wabba_woo_woo_woo_woo_woo (RATIO: 42,857…%)**

```
original_string_is :  wabba_wabba_wabba_wabba_wabba__wabba_woo_woo_woo_woo_woo
**********************************************************************************************************************************
compressed_dict :  [[0, 'w'], [0, 'a'], [0, 'b'], [3, 'a'], [0, '_'], [1, 'a'], [3, 'b'], [2, '_'], [6, 'b'], [4, '_'], [9, 'b'], [8, 'w'], [2, 'b'], [10, '_'], [11, 'a'], [5, 'w'], [0, 'o'], [17,
'_'], [1, 'o'], [18, 'w'], [17, 'o'], [16, 'o'], [20, 'o'], [17, '']]
compressed_data_by_binary :  1011011 1011011 110000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011
110000 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1011111 100010 1011101
101100 100000 1011011 110001 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110010 101100 100000 100010
1011111 100010 1011101 101100 100000 1011011 110110 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110100 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011
111001 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 111000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110010 101100 100000 100010 1100010 100010 1011101
101100 100000 1011011 110001 110000 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011 110001 110001 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110101 101100
100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 110111 101100 100000 100010 1011111 100010 1011101 101100
100000 1011011 110001 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110001 111000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110001 110111 101100 100000
100010 1101111 100010 1011101 101100 100000 1011011 110001 110110 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110010 110000 101100 100000 100010 1101111 100010 1011101 101100
100000 1011011 110001 110111 101100 100000 100010 100010 1011101 1011101
**********************************************************************************************************************************
decompressed_dict :  ['w', 'a', 'b', 'ba', '_', 'wa', 'bb', 'a_', 'wab', 'ba_', 'wabb', 'a_w', 'ab', 'ba__', 'wabba', '_w', 'o', 'o_', 'wo', 'o_w', 'oo', '_wo', 'o_wo', 'o']
original_string :  wabba_wabba_wabba_wabba_wabba__wabba_woo_woo_woo_woo_woo
compress_ratio :  42.857142857142854 %
```

4. **wabba_ wabba_ wabba_ wabba_ wabba_ wabba_ wabba_woo_woo_woo_woo (RATIO: 44,44…%)**

```
original_string_is :  wabba_ wabba_ wabba_ wabba_ wabba_ wabba_ wabba_woo_woo_woo_woo
**********************************************************************************************************************************
compressed_dict :  [[0, 'w'], [0, 'a'], [0, 'b'], [3, 'a'], [0, '_'], [0, ' '], [1, 'a'], [3, 'b'], [2, '_'], [6, 'w'], [2, 'b'], [4, '_'], [10, 'a'], [8, 'a'], [5, ' '], [7, 'b'], [12, ' '], [16,
'b'], [9, ' '], [18, 'a'], [5, 'w'], [0, 'o'], [22, '_'], [1, 'o'], [23, 'w'], [22, 'o'], [21, 'o'], [22, '']]
compressed_data_by_binary :  1011011 1011011 110000 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011
110000 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110011 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1011111 100010 1011101
101100 100000 1011011 110000 101100 100000 100010 100000 100010 1011101 101100 100000 1011011 110001 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110011 101100 100000 100010
1100010 100010 1011101 101100 100000 1011011 110010 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011 110110 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011
110010 101100 100000 100010 1100010 100010 1011101 101100 100000 1011011 110100 101100 100000 100010 1011111 100010 1011101 101100 100000 1011011 110001 110000 101100 100000 100010 1100001 100010
1011101 101100 100000 1011011 111000 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110101 101100 100000 100010 100000 100010 1011101 101100 100000 1011011 110111 101100 100000
100010 1100010 100010 1011101 101100 100000 1011011 110001 110010 101100 100000 100010 100000 100010 1011101 101100 100000 1011011 110001 110110 101100 100000 100010 1100010 100010 1011101 101100
100000 1011011 111001 101100 100000 100010 100000 100010 1011101 101100 100000 1011011 110001 111000 101100 100000 100010 1100001 100010 1011101 101100 100000 1011011 110101 101100 100000 100010
1110111 100010 1011101 101100 100000 1011011 110000 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110010 110010 101100 100000 100010 1011111 100010 1011101 101100 100000
1011011 110001 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110010 110011 101100 100000 100010 1110111 100010 1011101 101100 100000 1011011 110010 110010 101100 100000 100010
1101111 100010 1011101 101100 100000 1011011 110010 110001 101100 100000 100010 1101111 100010 1011101 101100 100000 1011011 110010 110010 101100 100000 100010 100010 1011101 1011101
**********************************************************************************************************************************
decompressed_dict :  ['w', 'a', 'b', 'ba', '_', ' ', 'wa', 'bb', 'a_', ' w', 'ab', 'ba_', ' wa', 'bba', '_ ', 'wab', 'ba_ ', 'wabb', 'a_ ', 'wabba', '_w', 'o', 'o_', 'wo', 'o_w', 'oo', '_wo', 'o']
original_string :  wabba_ wabba_ wabba_ wabba_ wabba_ wabba_ wabba_woo_woo_woo_woo
compress_ratio :  44.44444444444444 %
```

<u>Les test-cases complexes (fichiers texte) – Le code Python ci-dessous</u>

```python
# ### complex test with text file
file = "text.txt"
compress_data = []

with open(file, "r") as file:
        data = file.read()
        data_size = len(data)
        _,bin_comp_data = Compress_LZ78(data)

        with open("lz78_coded_test.txt","w") as f:
                compress_data,_ = Compress_LZ78(data)
                f.write(str(compress_data))

        with open("lz78_decoded_test.txt", "w") as f:
                _,decompress_data = Decompress_LZ78(compress_data)
                f.write(str(decompress_data))
```

J'ai déjà sauvegarder les résultats dans 3 fichiers texte :

1. « lz78_code.txt » sauvegarde le dictionnaire d'encodage
2. « lz78_decoded.txt » sauvegarde les informations originaux
3. « testcase5.txt » sauvegarde la ratio de compression

Open lz78_coded_test.txt ~/Desktop/Tran-Duy

```
[[0, 'S'], [0, 'o'], [0, ' '], [0, 'f'], [0, 'a'], [0, 'r'], [3, 'I'], [3, 'h'], [5, 'v'], [0, 'e'], [3, 'b'], [10, 'e
[0, 'u'], [2, 't'], [17, 'f'], [6, 'o'], [0, 'm'], [3, 't'], [0, 'h'], [17, 'r'], [10, 'p'], [2, 'r'], [18, 's'], [3,
'w'], [5, 'r'], [0, 'd'], [10, 'd'], [3, 'd'], [21, 'r'], [0, 'i'], [13, 'g'], [0, '\n'], [18, 'h'], [10, 's'], [17, '
'l'], [2, 'c'], [0, 'k'], [3, 'H'], [2, 'l'], [25, 'e'], [0, 's'], [0, '.'], [3, 'N'], [2, 'w'], [0, ','], [8, 'o'], [
[6, 'i'], [77, 'd'], [14, 't'], [14, ' '], [0, 'p'], [2, 'i'], [13, 't'], [3, 'i'], [13, ' '], [25, 'y'], [48, 'n'], [
[2, 'm'], [83, 'e'], [16, 'l'], [43, ' '], [18, 'o'], [14, 'b'], [5, 'n'], [42, 'o'], [87, 't'], [33, 's'], [3, 'm'],
```

lz78_coded.txt

Activities Text Editor Fri 09:12

Open lz78_decoded_test.txt ~/Desktop/Tran-Duy

```
So far I have been able to quote from the reports which I have forwarded during
these early days to Sherlock Holmes. Now, however, I have arrived at a point in my
narrative where I am compelled to abandon this method and to trust once more to
my recollections, aided by the diary which I kept at the time. A few extracts from
the latter will carry me on to those scenes which are indelibly fixed in every detail
upon my memory. I proceed, then, from the morning which followed our abortive
chase of the convict and our other strange experiences upon the moor.

October 16th. A dull and foggy day with a drizzle of rain. The house is banked in
with rolling clouds, which rise now and then to show the dreary curves of the moor,
with thin, silver veins upon the sides of the hills, and the distant boulders gleaming
```

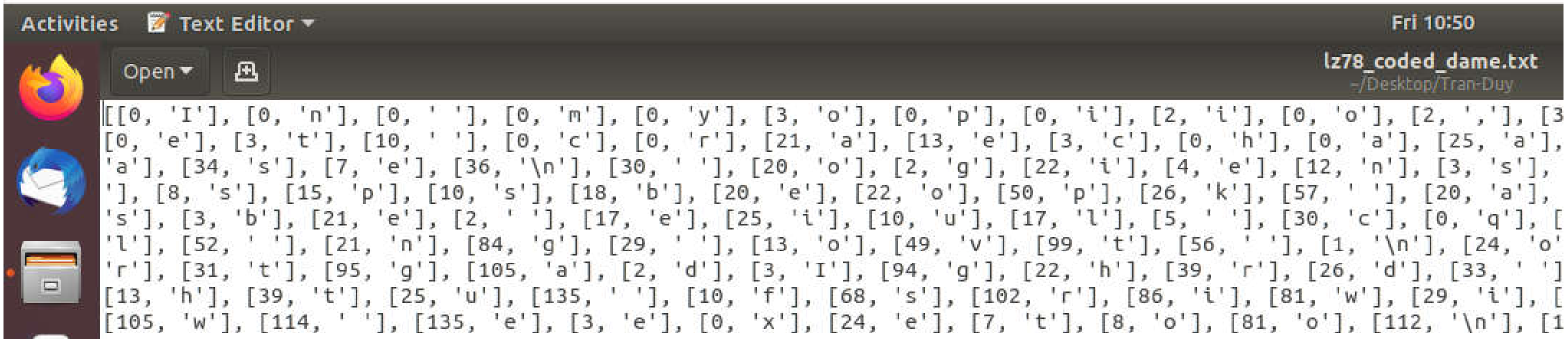
lz78_decoded.txt

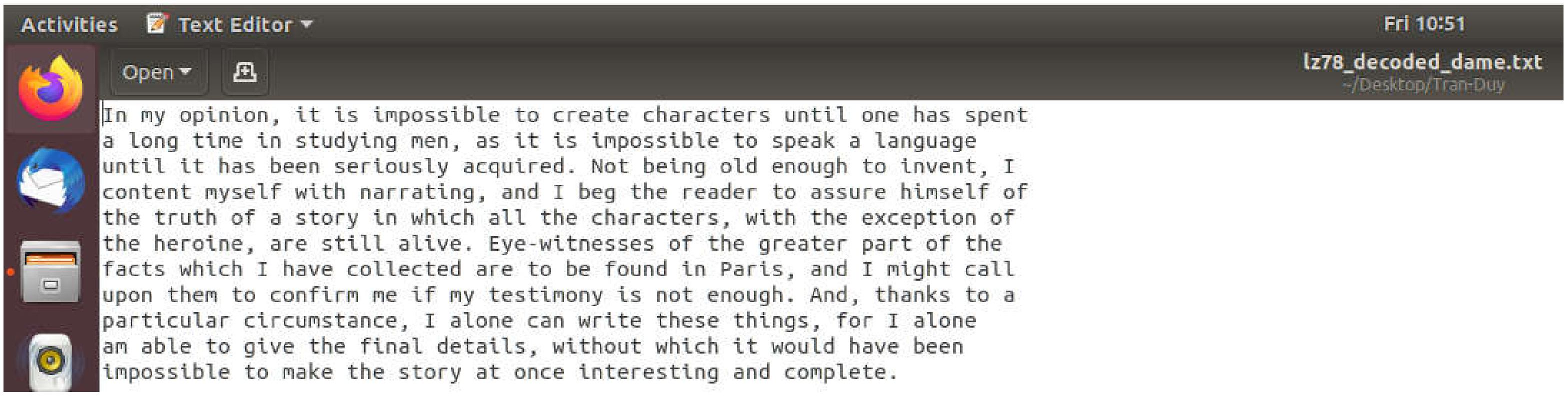
Open testcase5.txt ~/Desktop/Tran-Duy

```
ratio is : 50.96472320067294%
```

La ratio de compression

J'ai fait une autre test pour un autre texte (chapitre 1 – La Dame au Camelia) [je l'ai mis ce fichier dans la même répertoire). Les résultats que je vais obtenir sont :





CONCLUSION

L'algorithme de LZ78 que j'ai déjà implémenté a marché bien en 2 cases : simple et complexe. **Cependant, je pense que ces algorithmes ne sont pas très efficace (la complexité est grande car j'ai déjà utilisé la récursive) => la difficulté est comment annuler la récursive (je pense pour faire cela, je peux utiliser l'approche de l'algorithme de gourmand).**

UNE PETITE OBSERVATION POUR LES TEST CASES

1. **POUR LES CAS DE TEST SIMPLE : LA RATIO EST ENTRE 40 – 50%**
2. **POUR LES CAS DE TEST COMPLEXE : LA RATIO EST ENTRE 50 – 60%**