

ECOLE NATIONALE POLYTECHNIQUE
GENIE INDUSTRIEL
DS & AI

COMPTE RENDU DU TP2.

Codage de Huffman et arithmétique-Compression et décompression
TINFO.

Fait par :

OUCHENE SOUHIL

souhil.ouchene@g.enp.edu.dz

1 DS & IA

Objectif du TP

Compression et décompression d'une séquence numérique, en utilisant les algorithmes de Huffman et Arithmétique.
Comparaison de l'efficacité des deux algorithmes en matière de compression.

Partie 1

Considérant l'alphabet de la source **Alpha= {1, 2, 3}** et la séquence à compresser : **Seq=3313333323**.

1. Utilisation des fonctions **huffmandict** & **huffmanenco** :

➤ CODE MATLAB

```
% initialisation:
Alpha= [1 2 3];
Seq=[3 3 1 3 3 3 3 3 2 3];
%compression avec huffman
nb_1=0;
nb_2=0;
nb_3=0;
%calcul de probabilites:
prob_occ=[0 0 0];
for i=1:length(Seq)
    if(Seq(i)==1)
        nb_1=nb_1+1;
    elseif (Seq(i)==2)
        nb_2=nb_2+1;
    else
        nb_3=nb_3+1;
    end
end
%creation du dictionnaire :
prob_occ=1/10 * [nb_1 nb_2 nb_3];
dict=huffmandict(Alpha, prob_occ);
dict;
%codage du message:
enco_huff=huffmanenco(Seq,dict);
enco_huff;
```

➤ Résultats

Le dictionnaire est comme suit :

| | |
|---|-------|
| 1 | [1,1] |
| 2 | [1,0] |
| 3 | 0 |

Le codage du message sera alors comme suit :

```
enco_huff =
    0    0    1    1    0    0    0    0    0    1    0    0
```

2. Utilisation de la fonction **huffmandeco** pour la décompression :

➤ CODE MATLAB

```
%decodage:
decode_huff=huffmandeco(enco_huff, dict);
```

➤ Résultats

```
decode_huff =
    3    3    1    3    3    3    3    3    2    3
```

Ceci est identique au message compressé avant et après décompression.

3. Comparaison :

On sait qu'on a 1 octet (8 bits) pour chaque symbole du code original, alors :

➤ CODE MATLAB

```
%comparaison
diff=length(Seq)*8-length(enco_huff);
diff;
if (diff>0)
    display("le compresseur de huffman nous a permet de gagner de l'espace")
end
```

➤ Résultats

On voit bien que la différence est de 68 bits, cad la compression de **huffman** nous a permet de gagner un espace de 68 bits .

```
diff =
"le compresseur de huffman nous a permet de gagner de l'espace"
68
```

4. Utilisation de la fonction **arithenco** :

➤ CODE MATLAB

```
%compression avec arithmetique:
counts = [1 1 8];
enco_arth=arithenco(Seq, counts);
enco_arth;
```

➤ Résultats

On aura :

```
enco_arth =
0      1      1      0      0      0      1      1      0      0      0      0      0      0      0
```

5. Décompression en utilisant la fonction **arithdeco** :

➤ CODE MATLAB

```
len=length(Seq);
decode_arth=arithdeco(enco_arth, counts, len);
decode_arth;
```

➤ Résultats

On aura alors :

```
decode_arth =
3      3      1      3      3      3      3      3      2      3
```

Ceci est identique au message compressé avant et après décompression.

6. Comparaison :

➤ CODE MATLAB

```
%comparaison 2
diff_1=length(Seq)*8-length(enco_arth);
diff_1;
if (diff_1>0)
    display("le compresseur arithmetique nous a permet de gagner de
l'espace")
end
```

➤ Résultats

On voit bien que la différence est de 65 bits en mémoire, cad la compression **arithmétique** nous a permet de gagner un espace de 65 bits.

| | |
|---|----------|
| "le compresseur arithmetique nous a permet de gagner de l'espace" | diff_1 = |
| | 65 |

7. Comparaison entre la compression de **Huffman** et la compression **arithmétique** :

Comme on ne peut pas calculer l'efficacité de ces 2 codages (on ne peut pas calculer la longueur moyenne), on va comparer ces 2 méthodes selon le nombre de bits dans chaque message codé cad calculer la différence entre la taille du code obtenu par huffman et l'arithmétique, on aura alors :

➤ CODE MATLAB

```
if(length(enco_huff)<length(enco_arth))
    display("le codage de huffman est le meilleur")
else
    display("le codage arithmetique est le meilleur")
end
```

➤ Résultats

```
"le codage de huffman est le meilleur"
```

Et si on peut aussi calculer facilement cette différence qui est de **3**.

Remarque :

On a trouvé que la compression de **Huffman est la meilleure** car on travaille avec nombre petit de symboles dans le message (10 symboles), pour le cas d'un **nombre important de symboles** (+200 symboles) la méthode **arithmétique sera la meilleure**.

Partie 2

1. Lecture du fichier texte:

CODE MATLAB

```
%lecture du fichier
TP=importdata('tp.txt');
%transformation de la chaine de caracteres en un vecteur de char
symboles = char(TP);
```

J'ai utilisé le texte du TP1, et j'ai supprimé tous les points, les virgules les caractères accentués, en laissant que les caractères de l'alphabet et les espaces (fichier.txt est en pièce jointe dans le mail). Pour la lecture (éviter tous les erreurs), il faut bien préciser le path du fichier, et puis on va avoir le résultat comme un string qui représente une seule case d'un tableau, pour cela on va la transformer en un vecteur de caractères, on aura alors :

➤ **Résultats**

```
TP =

1x1 cell array

{'on peut tout te prendre tes biens tes plus belles annees l ensemble de tes joies et l ensemble de tes merites jusqu a ta derniere chemise il te restera toujours tes reves pour reinvent
```

La conversion va donner le vecteur « symboles » suivant :

```
symboles =

'on peut tout te prendre tes biens tes plus belles annees l ensemble de tes joies et l ensemble de tes merites jusqu a ta derniere chemise il te restera toujours tes reves pour reinvent
```

Avec :

```
>> size(TP)

ans =

     1     1

>> size(symboles)

ans =

     1    216
```

2. Calcul de probabilités :

➤ **CODE MATLAB**

```
%trouver les nombres d'occurrences pour chaque caractere: on utilise count
%on a considere que a=A ...
nb_a=count(symboles,'a','ignoreCase',true);
nb_b=count(symboles,'b','ignoreCase',true);
nb_c=count(symboles,'c','ignoreCase',true);
nb_d=count(symboles,'d','ignoreCase',true);
nb_e=count(symboles,'e','ignoreCase',true);
nb_f=count(symboles,'f','ignoreCase',true);
nb_g=count(symboles,'g','ignoreCase',true);
nb_h=count(symboles,'h','ignoreCase',true);
nb_i=count(symboles,'i','ignoreCase',true);
nb_j=count(symboles,'j','ignoreCase',true);
```

```

nb_k=count(symboles,'k','ignoreCase',true);
nb_l=count(symboles,'l','ignoreCase',true);
nb_m=count(symboles,'m','ignoreCase',true);
nb_n=count(symboles,'n','ignoreCase',true);
nb_o=count(symboles,'o','ignoreCase',true);
nb_p=count(symboles,'p','ignoreCase',true);
nb_q=count(symboles,'q','ignoreCase',true);
nb_r=count(symboles,'r','ignoreCase',true);
nb_s=count(symboles,'s','ignoreCase',true);
nb_t=count(symboles,'t','ignoreCase',true);
nb_u=count(symboles,'u','ignoreCase',true);
nb_v=count(symboles,'v','ignoreCase',true);
nb_w=count(symboles,'w','ignoreCase',true);
nb_x=count(symboles,'x','ignoreCase',true);
nb_y=count(symboles,'y','ignoreCase',true);
nb_z=count(symboles,'z','ignoreCase',true);
nb_points=count(symboles,'.','ignoreCase',true);
nb_ver=count(symboles,',','ignoreCase',true);
nb_semicol=count(symboles,';','ignoreCase',true);
%pour les espaces on va utiliser la fonction : isstrprop(TEXT, type)
nb_spaces=0;
B = isstrprop(symboles, 'wspace');
for i=1:length(B)
    nb_spaces=nb_spaces+B(i);
end
nb_spaces;
%definition du vecteur de probabilités
P=1/length(symboles)*[nb_a nb_b nb_c nb_d nb_e nb_f nb_g nb_h nb_i nb_j nb_k
nb_l nb_m nb_n nb_o nb_p nb_q nb_r nb_s nb_t nb_u nb_v nb_w nb_x nb_y nb_z
nb_spaces ];

```

On aura alors comme résultat :

➤ Résultats

P =

Columns 1 through 18

| | | | | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|---|--------|--------|--------|---|--------|--------|--------|--------|--------|--------|--------|
| 0.0231 | 0.0185 | 0.0093 | 0.0231 | 0.1991 | 0.0046 | 0 | 0.0046 | 0.0370 | 0.0139 | 0 | 0.0463 | 0.0231 | 0.0602 | 0.0417 | 0.0185 | 0.0139 | 0.0556 |
|--------|--------|--------|--------|--------|--------|---|--------|--------|--------|---|--------|--------|--------|--------|--------|--------|--------|

Columns 19 through 27

| | | | | | | | | |
|--------|--------|--------|--------|---|---|---|---|--------|
| 0.0880 | 0.0787 | 0.0463 | 0.0093 | 0 | 0 | 0 | 0 | 0.1852 |
|--------|--------|--------|--------|---|---|---|---|--------|

3. Calcul de la probabilité totale :

➤ CODE MATLAB

```
%calcul de P totale
p_totale=0;
for i=1:length(P)
    p_totale=p_totale+P(i);
end
p_totale;
if (p_totale ~= 1)
    display("error");
else
    display("everything is okey");
end
```

➤ Résultats

Après avoir faire la sommation, on a obtenu que $p_{\text{totale}} = 1$

```
p_totale =
"everything is okey"
1
```

4. Calcul de l'information propre :

Il nous faut d'abord éliminer les valeurs nulles du vecteur de probabilités puis on calcule l'information propre :

➤ CODE MATLAB

```
%on doit utiliser un tableau sans valeurs nulles pour qu'on puisse calculer
%l'entropie et l'information , alors on utilise la fonction find avec la
condition P(i)<>0 et
%on stocke ces valeurs dans un vecteur P_not_null
P_not_null = P(find(P>0));
% calcul de l'information propre:
Info=-log2(P_not_null);
Info;
```

➤ Résultats

P_not_null =

Columns 1 through 18

| | | | | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 0.0231 | 0.0185 | 0.0093 | 0.0231 | 0.1991 | 0.0046 | 0.0046 | 0.0370 | 0.0139 | 0.0463 | 0.0231 | 0.0602 | 0.0417 | 0.0185 | 0.0139 | 0.0556 | 0.0880 | 0.0787 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|

Columns 19 through 21

| | | |
|--------|--------|--------|
| 0.0463 | 0.0093 | 0.1852 |
|--------|--------|--------|

>> Info

Info =

Columns 1 through 18

| | | | | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 5.4330 | 5.7549 | 6.7549 | 5.4330 | 2.3286 | 7.7549 | 7.7549 | 4.7549 | 6.1699 | 4.4330 | 5.4330 | 4.0544 | 4.5850 | 5.7549 | 6.1699 | 4.1699 | 3.5070 | 3.6674 |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|

Columns 19 through 21

| | | |
|--------|--------|--------|
| 4.4330 | 6.7549 | 2.4330 |
|--------|--------|--------|

5. Définition du dictionnaire :

On va d'abord convertir le vecteur symboles des caractères alphabétiques en un vecteur de valeurs numériques en se basant sur la table ASCII, puis on calcule les probabilités d'apparence de chaque symbole (qui sont les mêmes que celle de P) et enfin on utilise **huffmandict** pour définir notre dictionnaire :

➤ CODE MATLAB

```
%definir le vecteur de valeur possibles
alphabets=[97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113
114 115 116 117 118 119 120 121 122 32] ;
%convertir la chaine en code ascii
conv=abs(symboles);
%calcul de probabilite
P_conv=P;
%le codage de chaque symbole
M_code=huffmandict(alphabets, P_conv);
```

On aura alors les résultats suivants :

➤ Résultats

La conversion du message en ASCII va nous donner :

```
conv =

Columns 1 through 31

    111    110     32    112    101    117    116     32    116    111    117    116     32    116    101     32    112    114    101    110    100    114    101     32    116    101    115     32     98    105    101

Columns 32 through 62

    110    115     32    116    101    115     32    112    108    117    115     32     98    101    108    108    101    115     32     97    110    110    101    101    115     32    108     32    101    110    115

Columns 63 through 93

    101    109     98    108    101     32    100    101     32    116    101    115     32    106    111    105    101    115     32    101    116     32    108     32    101    110    115    101    109     98    108

Columns 94 through 124

    101     32    100    101     32    116    101    115     32    109    101    114    105    116    101    115     32    106    117    115    113    117     97     32    116     97     32    100    101    114    110

Columns 125 through 155

    105    101    114    101     32     99    104    101    109    105    115    101     32    105    108     32    116    101     32    114    101    115    116    101    114     97     32    116    111    117    106

Columns 156 through 186

    111    117    114    115     32    116    101    115     32    114    101    118    101    115     32    112    111    117    114     32    114    101    105    110    118    101    110    116    101    114     32

Columns 187 through 216

    108    101     32    109    111    110    100    101     32    113    117    101     32    108     32    111    110     32    116     97     32     99    111    110    102    105    115    113    117    101
```

Le vecteur probabilités associées sera alors :

```
P_conv =

Columns 1 through 18

    0.0231    0.0185    0.0093    0.0231    0.1991    0.0046         0    0.0046    0.0370    0.0139         0    0.0463    0.0231    0.0602    0.0417    0.0185    0.0139    0.0556

Columns 19 through 27

    0.0880    0.0787    0.0463    0.0093         0         0         0         0    0.1852
```


Alors le dictionnaire obtenu sera donc :

| | |
|-----|-----------------------------|
| 97 | [0,0,0,0,0,0] |
| 98 | [0,1,0,1,0,1] |
| 99 | [0,1,0,1,1,0,0] |
| 100 | [0,0,0,0,1,1] |
| 101 | [1,0] |
| 102 | [0,1,0,1,1,0,1,1,0] |
| 103 | [0,1,0,1,1,0,1,1,1,1,1,1] |
| 104 | [0,1,0,1,1,0,1,0] |
| 105 | [0,1,0,0,1] |
| 106 | [0,0,0,0,0,1,0] |
| 107 | [0,1,0,1,1,0,1,1,1,1,1,1,0] |
| 108 | [0,0,0,1,1] |
| 109 | [0,0,0,0,1,0] |
| 110 | [0,1,1,0] |
| 111 | [0,1,0,0,0] |
| 112 | [0,1,0,1,0,0] |
| 113 | [0,1,0,1,1,1] |
| 114 | [0,1,1,1] |
| 115 | [0,0,1,0] |
| 116 | [0,0,1,1] |
| 117 | [0,0,0,1,0] |
| 118 | [0,0,0,0,0,1,1] |
| 119 | [0,1,0,1,1,0,1,1,1,1,1,0] |
| 120 | [0,1,0,1,1,0,1,1,1,1,0] |
| 121 | [0,1,0,1,1,0,1,1,1,0] |
| 122 | [0,1,0,1,1,0,1,1,1,0] |
| 32 | [1,1] |

6. Encodage du texte :

Pour ceci, on va utiliser la fonction **huffmanenco** et puis enregistrer le code obtenu dans un fichier .txt (sera en piece jointe dans le mail) :

➤ CODE MATLAB

```
%codage du message complet
encodage=huffmanenco(symboles,M_code);
%sauvgarde du resultat dans un fichier .txt
```

Dans l'observation de la taille des fichiers, j'ai remarquée que la taille du texte original est de 1Kb mais la taille du code du texte est de 2Kb, ce qui est illogique car on a fait une compression du texte ce qui veut dire que le code doit être de taille inferieure en mémoire, par analogie a ce qu'on a fait dans la première partie le code contient 817 bits seulement mais si on considère qu'une lettre se code sur 8 bits, le message contient 216 caractères ce qui veut dire qu'il va être codé sur $216 \times 8 = 1728$ bits.

➤ Résultats

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | |
| 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |

7. Décodage avec la fonction **huffmandeco** :

Dans cette partie on utilise la fonction `huffmandeco`, on va obtenir un vecteur contenant des valeurs numériques qu'on va convertir en alphabets pour restituer notre message original.

➤ CODE MATLAB

```
%decodage de la sequence par huffmandeco:
decodage=huffmandeco(encodage, M_code);
%conversion du code numerique vers ascii
decod_ascii=char(decodage);
```

➤ Résultats

le résultat du décodage en numérique :

```
decodage =
|
Columns 1 through 29
111 110 32 112 101 117 116 32 116 111 117 116 32 116 101 32 112 114 101 110 100 114 101 32 116 101 115 32 98

Columns 30 through 58
105 101 110 115 32 116 101 115 32 112 108 117 115 32 98 101 108 108 101 115 32 97 110 110 101 101 115 32 108

Columns 59 through 87
32 101 110 115 101 109 98 108 101 32 100 101 32 116 101 115 32 106 111 105 101 115 32 101 116 32 108 32 101

Columns 88 through 116
110 115 101 109 98 108 101 32 100 101 32 116 101 115 32 109 101 114 105 116 101 115 32 106 117 115 113 117 97

Columns 117 through 145
32 116 97 32 100 101 114 110 105 101 114 101 32 99 104 101 109 105 115 101 32 105 108 32 116 101 32 114 101

Columns 146 through 174
115 116 101 114 97 32 116 111 117 106 111 117 114 115 32 116 101 115 32 114 101 118 101 115 32 112 111 117 114

Columns 175 through 203
32 114 101 105 110 118 101 110 116 101 114 32 108 101 32 109 111 110 100 101 32 113 117 101 32 108 32 111 110
```

La conversion suivant ASCII :

```
decod_ascii =
```

```
'on peut tout te prendre tes biens tes plus belles annees l ensemble de tes joies et l ensemble de tes merites jusqu a ta derniere chemise il te restera toujours tes reves pour
```

On remarque bien que c'est bien le message original qu'on a compressé et puis décompressé.

8. Calcul de l'entropie :

➤ **CODE MATLAB**

```
% calcul de l'entropie:
h=0;
for i=1:length(P_not_null)
    h=h-P_not_null(i)*log2(P_not_null(i));
end
```

➤ **Résultats** h =

On obtient le résultat suivant :

3.7232

9. Calcul de la longueur moyenne :

On doit d'abord savoir le nombre de bits pour chaque symbole cad L_i et puis multiplier chaque L_i par son P_i du vecteur de probabilités. Puis, on compare la longueur moyenne par h et on affiche le message d'aucune erreur dans le cas où h est inférieure à la longueur.

➤ **CODE MATLAB**

```
%calcul de la longueur moyenne:
%on va d'abords savoir le nombre de bits pour chaque symbole:
for i=1:length(M_code)
    longueur_par_symbole(i)=length(M_code{i,2});
end
longueur_par_symbole;
longueur=0;
for i=1:length(M_code)
    longueur=longueur+P(i)*longueur_par_symbole(i);
end
if(h>longueur)
    display("error");
else
    display("it is okay");
end
```

➤ **Résultats** "it is okay"

On obtient alors :

```
>> longueur
```

```
longueur =
```

3.7824

10. Calcul de l'efficacité et la redondance :

➤ CODE MATLAB

```
%calcul d'efficacite et redondance du code:  
%on a efficacite= H/longueur moyenne * 100  
%on redondance= 1-efficacite:
```

```
eff=h/longueur *100;  
red=100-eff;
```

➤ Résultats

```
>> eff  
  
eff =  
  
98.4357  
  
>> red  
  
red =  
  
1.5643
```

11. Le taux et le gain de compression :

➤ CODE MATLAB

```
%calcul du taux de compression:  
taille_apres_compression=length(encodage);  
%on considere qu'on code un symbole sur 8 bits  
taille_avant_compression=length(symboles)*8;  
Tc=taille_apres_compression/taille_avant_compression * 100;  
%calcul du gain de compression:  
Gain=100-Tc;
```

On aura alors les résultats :

➤ Résultats

```
>> Tc  
  
Tc =  
  
47.2801  
  
>> Gain  
  
Gain =  
  
52.7199
```