

**TBA – Laboratoire n°1**

**Professeur**

**M. Markus JATON**

**Assistant**

M.Grégory RUCH

**Auteurs**

Steve LIENHARD

Arnaud BURKALTHER

**Dates :** 27 Février 2010

**Temps à disposition :** 2 semainesA description...

**Date de début:** 18 Février 2010

**Date de fin :** 4 Mars

**Table des matières :**

Introduction

Dans les technologies des télécommunications, le codage a une importance cruciale, notamment quand l'on veut optimiser la vitesse et le débit d'une transmission.

Ingénieur.txt

# 3.1 Fréquence des symboles

## 3.1.1

La fonction frequency.m est compatible avec de l’ASCII étendu car la fonction retourne un tableau de 256 caractères, valeur égal au nombre de caractères de ce code.

## 3.1.2

**Code :**

*file = textread('ingenieur.txt','%s','delimiter','\n','whitespace','');*

*vector=uint8(file{1,1});*

*freq = frequency(vector);*

*[I,Symbol,Proba] = find(freq);*

*[probamax, Iprobamax] = max(Proba)*

*sprintf('%s', strcat('Le nombre de symboles dans le texte ingenieur.txt est : ', num2str(length(I))))*

**Résultat :**

Le fichier de texte Lingenieur.txt comprend 30 caractères différents.

## 3.1.3

**Code :**

*for i = 1:length(I)*

*sprintf('%s', strcat('Le caractère : ', char(Symbol(i)-1),' apparait avec une probabilité de : ', num2str(Proba(i))))*

*end;*

**Résultat :**

Le caractère ‘espace’ avec une probabilité de : 0.14937

Le caractère ‘e’ avec une probabilité de : 0.13418

Le caractère ‘s’ avec une probabilité de : 0.081013

# 3.2 Codage de Huffman – Ingénieur.txt

***Code***

*%Application de la norme de huffman (3.2)*

*[zipped,info] = norm2huff(vector);*

*[Words,Simbols] = huffcodes2bin(info.huffcodes)*

*for i = 1:length(I)*

*sprintf('%s', info)*

*end;*

## 3.2.1

**ASCII : Nb de bits total**

395

**Huffman : Nb de bits par caractère**

Words =

'001'

'011'

'0000'

'0010'

'0100'

'0110'

'1010'

'1100'

'1110'

'1111'

'01000'

'01101'

'10111'

'11101'

'010101'

'011000'

'100101'

'100111'

'0000101'

'0110101'

'0111000'

'1000101'

'1110101'

'1111000'

'00000111'

'10000111'

'001000111'

'011000111'

'101000111'

'111000111'

**Huffman : Nb de bits total**

209

**Taux de compression :**

395-209/395 = 0.471

## 3.2.2

**Longueur moyenne :** 256 caractères => 8 bits

## 3.2.3

**Longueur moyenne :** 209\*8/ 395 = 4.2329

## 3.2.4

???????

**Entropie : 4.18**

Redondance = 1-efficacité =

## 3.2.5

**Plus petit :** 3 bits

**Plus grand :** 9 bits

# 3.3 Décodage de Huffman

***Code***

vector2= huff2norm(zipped,info);

vectordouble=double(vector);

vectordouble2=double(vector2);

sum(vectordouble - vectordouble2)

## 3.3.1

Non il n’y a pas de différence. Nous avons vérifié cela en faisant la soustraction des deux vecteurs, et avons contrôlé le résultat. La valeur retournée étant 0, nous sommes certain que les deux vecteurs sont identiques.

## 3.3.2 :

Oui, par Huffman construction (préambule)

## 3.3.3 :

Fichier Ingenieur.txt : 395 bits

Fichier Ingenieur zippé : 359 bits

**Taux de compression :** (395-359)/395 = 0.0911

Par ce résultat, nous pouvons déduire que l’algorithme de compression utilisé pour zipper le fichier, est moins performant que celui défini par Huffman.

Huffman.txt

# 3.1 Fréquence des symboles

3.1.2

Le nombre de symboles dans le texte Huffman.txt est : 43

3.1.3

Le caractère : apparait avec une probabilité de :0.15443

Le caractère :' apparait avec une probabilité de :0.0045872

Le caractère :( apparait avec une probabilité de :0.0015291

3.2.1

**Huffman : Nb de caractères**

Words =

'011'

'110'

'0000'

'0001'

'0010'

'0100'

'0101'

'1010'

'1100'

'00111'

'01101'

'01111'

'11000'

'11101'

'11111'

'001001'

'011001'

'111001'

'0001000'

'0010111'

'1001000'

'1101001'

'1110111'

'00101001'

'01010111'

'000101000'

'000110111'

'001101000'

'010101000'

'010101001'

'011010111'

'011101000'

'100101000'

'100110111'

'101101000'

'110101000'

'110110111'

'111010111'

'111101000'

'0010110111'

'0110101001'

'1010110111'

'1110101001'

**Huffman : Nb de bits total**

299

**Taux de compression :**

(344-299)/344 = 0.130814

**3.2.2 :**

**Longueur moyenne :** 256 caractères => 8 bits

**3.2.3 :**

**Longueur moyenne : 299\*8**

**3.2.2 :**

**3.2.3 :**

**3.2.4 :**

**3.2.5 :**

**3.3.1 :**

**3.3.2 :**

**3.3.3 :**

# Conclusion