

Compression – méthode de Huffman

Thierry Lacroq

Université de Rouen
FRANCE

La méthode de Huffman

- consiste à remplacer les caractères les plus fréquents par des codes courts et les caractères les moins fréquents par des codes longs
- utilise la notion de code préfixe

Code préfixe

- Un code préfixe est un ensemble de mots tel qu'aucun mot de l'ensemble n'est préfixe d'un autre mot de l'ensemble
- Le décodage est alors immédiat
- Un code préfixe sur l'alphabet binaire $\{0, 1\}$ peut être représenté par un *trie* qui est en fait un arbre binaire où tous les nœuds internes ont exactement deux successeurs

Arbre

- Les feuilles sont étiquetées avec les caractères originaux, les branches par 0 ou 1 et les chemins depuis la racine jusqu'aux feuilles épellent les codes des caractères originaux
- L'utilisation d'un code préfixe assure que les codes sont bien représentés par les feuilles

Par convention, le fils gauche d'un nœud est étiqueté par 0 et le fils droit par 1

Encodage

La phase d'encodage se compose de trois étapes :

- 1 comptage des fréquences des caractères
- 2 construction du code préfixe
- 3 codage du texte

Exemple

fentrée contient

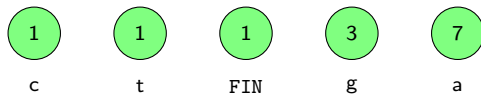
cagataagagaa

$12 \times 8 = 96$ bits

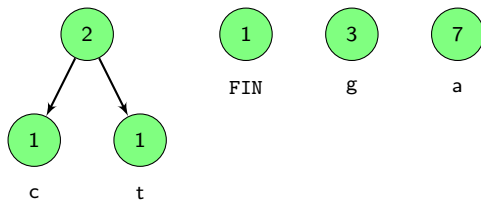
les fréquences sont

	a	c	g	t	FIN
<i>fréq</i>	7	1	3	1	1

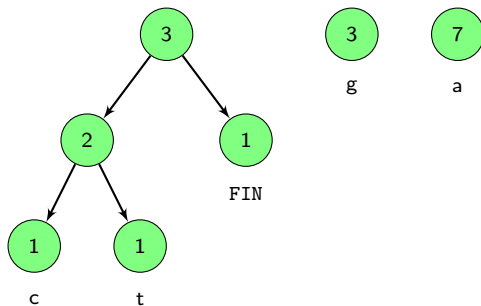
Exemple cagataagagaa



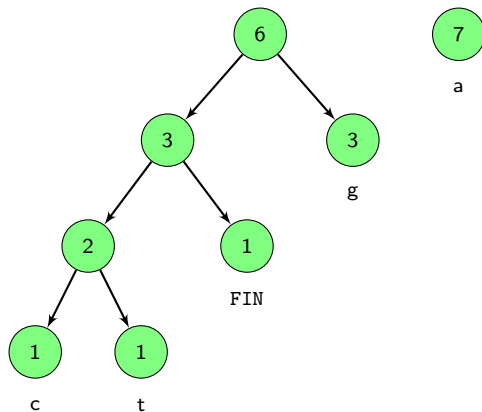
Exemple cagataagagaa



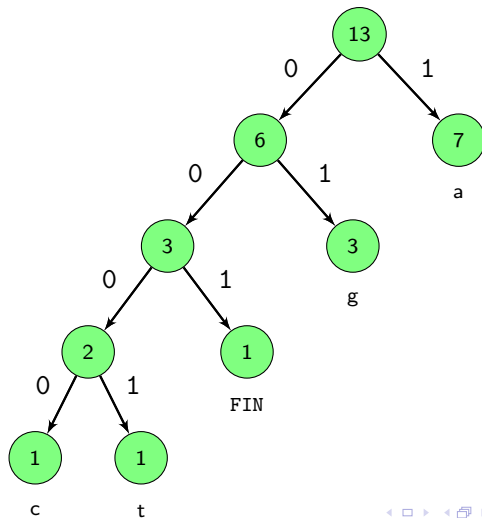
Exemple cagataagagaa



Exemple cagataagagaa



Exemple cagataagagaa



Exemple cagataagagaa

	a	c	g	t	FIN
code	1	0000	01	0001	001

codage de l'arbre :

00001 binaire(c, 9) 1 binaire(t, 9) 1 binaire(FIN, 9) 1 binaire(g, 9) 1
binaire(a, 9)

en ASCII :

00001 001000011 1 001010100 1 100000000 1 001000111 1
001000001

54 bits

Exemple cagataagagaa

codage du texte : 0000 1 01 1 0001 1 1 01 1 01 1 1 001

24 bits

total

78 bits

Comptage des fréquences de caractères

Compte(*fentrée*)

- 1 **pour** $a \in A$ **faire**
- 2 $\text{fréq}(a) \leftarrow 0$
- 3 **tantque** non $\text{fdf}(\textit{fentrée})$ et a est le prochain caractère **faire**
- 4 $\text{fréq}(a) \leftarrow \text{fréq}(a) + 1$
- 5 $\text{fréq}(\text{FIN}) \leftarrow 1$

Construction du code préfixe

Utilisation de $\text{fréq}(a)$ pour chaque $a \in A$

Const-arbre0()

- 1 créer un arbre t_a à un nœud pour chaque $a \in A$ avec $\text{poids}(a) = \text{fréq}(a)$
- 2 $i \leftarrow 0$
- 3 **répéter**
- 4 extraire les 2 arbres de poids plus faible t_1 et t_2
- 5 créer un arbre t avec t_1 et t_2 comme sous-arbre
- 6 $\text{poids}(t) \leftarrow \text{poids}(t_1) + \text{poids}(t_2)$
- 7 **jusqu'à** il ne reste qu'1 seul arbre

Algorithmes

Const-arbre()

- 1 **pour** $a \in A \cup \{\text{FIN}\}$ **faire**
- 2 **si** $\text{fréq}(a) \neq 0$ **alors**
- 3 créer un nœud t
- 4 $\text{poids}(t) \leftarrow \text{fréq}(a)$
- 5 $\text{étiq}(t) \leftarrow a$
- 6 $\text{lfeuilles} \leftarrow$ liste des nœuds dans l'ordre croissant des poids
- 7 $\text{larbres} \leftarrow$ liste vide
- 8 **tantque** $\text{LONGUEUR}(\text{lfeuilles}) + \text{LONGUEUR}(\text{larbres}) > 1$ **faire**
- 9 $(g, d) \leftarrow$ extraire les 2 nœuds de plus faible poids parmi
 les 2 premiers éléments de lfeuilles et
 les 2 premiers éléments de larbres
- 10 créer un nœud t
- 11 $\text{poids}(t) \leftarrow \text{poids}(g) + \text{poids}(d)$
- 12 $\text{gauche}(t) \leftarrow g$
- 13 $\text{droit}(t) \leftarrow d$
- 14 insérer t à la fin de larbres
- 15 **Retourner** t

Codes

Après la construction de l'arbre il est possible de retrouver le code de chaque caractère par un parcours en profondeur de l'arbre.

Const-code(t, ℓ)

```
1 si  $t$  n'est pas une feuille alors  
2    $temp[\ell] \leftarrow 0$   
3   CONST-CODE( $gauche(t), \ell + 1$ )  
4    $temp[\ell] \leftarrow 1$   
5   CONST-CODE( $droit(t), \ell + 1$ )  
6 sinon  $code(étiqu(t)) \leftarrow temp[0.. \ell - 1]$ 
```

Codage de l'arbre

La troisième étape nécessite de stocker les codes de chaque caractère avant le code du texte

Code-arbre(*fsortie*, *t*)

- 1 **si** *t* n'est pas une feuille **alors**
- 2 écrire un 0 dans *fsortie*
- 3 CODE-ARBRE(*fsortie*, *gauche*(*t*))
- 4 CODE-ARBRE(*fsortie*, *droit*(*t*))
- 5 **sinon** écrire un 1 dans *fsortie*
- 6 écrire *binnaire*(*étiq*(*t*)) dans *fsortie*

Codage du texte

On peut ensuite coder le texte

Code-texte(*fentrée*, *fsortie*)

- 1 **tantque** non $\text{fdf}(\textit{fentrée})$ et a est le prochain caractère **faire**
- 2 écrire $\textit{code}(a)$ dans *fsortie*
- 3 écrire $\textit{code}(\text{FIN})$ dans *fsortie*

Encodage complet

On peut maintenant écrire l'algorithme d'encodage complet

Encodage(*fentrée*, *fsortie*)

- 1 COMPTE(*fentrée*)
- 2 $t \leftarrow$ CONST-ARBRE()
- 3 CONST-CODE(t , 0)
- 4 CODE-ARBRE(*fsortie*, t)
- 5 CODE-TEXTE(*fentrée*, *fsortie*)

Décodage

Immédiat

- lecture de l'entête et reconstruction de l'arbre
- décodage du texte

Algorithmes

Reconst-arbre(*fentrée*, *t*)

- 1 $b \leftarrow$ lire 1 bit de *fentrée*
- 2 **si** $b = 1$ **alors**
 - ▷ feuille
- 3 $gauche(t) \leftarrow$ NIL
- 4 $droit(t) \leftarrow$ NIL
- 5 $étiq(t) \leftarrow$ symbole correspondants aux 9 prochains bits de *fentrée*
- 6 **sinon** créer un nœud g
- 7 $gauche(t) \leftarrow g$
- 8 RECONST-ARBRE(*fentrée*, g)
- 9 créer un nœud d
- 10 $droit(t) \leftarrow d$
- 11 RECONST-ARBRE(*fentrée*, d)

Algorithmes

Décode-texte(*fentrée*, *fsortie*, *racine*)

```
1  $t \leftarrow \text{racine}$ 
2 tantque  $\text{étiqu}(t) \neq \text{FIN}$  faire
3   si  $t$  est une feuille alors
4     écrire  $\text{étiqu}(t)$  dans fsortie
5      $t \leftarrow \text{racine}$ 
6   sinon  $b \leftarrow$  lire 1 bit de fentrée
7     si  $b = 0$  alors
8        $t \leftarrow \text{gauche}(t)$ 
9     sinon  $t \leftarrow \text{droit}(t)$ 
```

Décodage complet

Encodage(*fentrée*, *fsortie*)

- 1 créer un nœud *racine*
- 2 RECONST-ARBRE(*fentrée*, *racine*)
- 3 DÉCODE-TEXTE(*fentrée*, *fsortie*, *racine*)