Cours 4 : algorithmes de codage de source adaptatifs

Plan du cours

1. Motivation ;
2. Huffman adaptatif ;
3. Méthode de Gallager et Knuth ;
4. Méthode par dictionnaire : Lempel-Ziv 78, Lempel-Ziv-Welsh, Lempel-Ziv 77.

1. Motivation

Une application naïve nécessite deux passes sur le fichier :

1. une passe pour déterminer les probabilités ;
2. une passe pour le codage proprement dit.

De plus, il faut transmettre une information auxiliaire, à savoir
- soit les probabilités ;
- soit le dictionnaire.

que ce soit pour le code de Huffman ou pour le codage arithmétique.

Codage de source universel

algorithme de codage de source universel : ne suppose rien sur la source à compresser.

Idée : Construire à chaque instant un modèle dynamique de la source ayant pu produire le texte jusqu'à ce point et coder la lettre suivante dans ce modèle.

Illustration :

- l'algorithme de Huffman adaptatif (modèle sans mémoire),
- l'algorithme de codage arithmétique adaptatif,
- l'algorithme de Lempel-Ziv, et ses variantes (utilisation des dépendances entre les lettres).

2. Huffman adaptatif – Principe

Supposons que nous ayons déja lu n caractères dans le texte, correspondant à K lettres distinctes.

- nous fabriquons X_n une source de $K+1$ lettres formée de K lettres déjà apparues auxquelles on attribue une probabilité proportionnelle au nombre d'occurrences, et d'une $(K+1)$-ème lettre « fourre-tout » (vide) à laquelle on attribue la probabilité 0,
- on construit un code de Huffman T_n de cette source,
- la $n+1$-ème lettre est lue et est codée
  - par son mot de code s'il existe,
  - par le $K+1$-ème mot de code suivi du code ascii sinon.

Huffman adaptatif – Codage

L'arbre initial est constitué d'une unique feuille, celle de la lettre vide. À chaque fois qu'une lettre x est lue dans le texte source

- si elle est déjà apparue
  - on imprime son mot de code,
  - on met à jour l'arbre,
- sinon
  - on imprime le mot de code de la lettre vide suivi de x non codé (son code ascii en base 2 par exemple),
  - on ajoute une feuille dans l'arbre,
  - on met à jour l'arbre.

Huffman adaptatif – Décodage

L'arbre initial est constitué d'une unique feuille, celle de la lettre vide. Jusqu'à épuisement, on parcourt l'arbre en lisant les bits du texte codé ('0' à gauche, '1' à droite) jusqu'à arriver à une feuille

- s'il s'agit d'une lettre non vide
  - on imprime la lettre,
  - on met à jour l'arbre,
- sinon, il s'agit de la lettre vide
  - on lit les 8 bits suivants pour obtenir le code ascii d'une lettre que l'on imprime,
  - on ajoute une feuille dans l'arbre,
  - on met à jour l'arbre.

Huffman adaptatif – Définitions préliminaires

Nous identifions un code préfixe d'une source discrète X à un arbre binaire à $|X|$ feuilles étiquetées par les lettres de X.

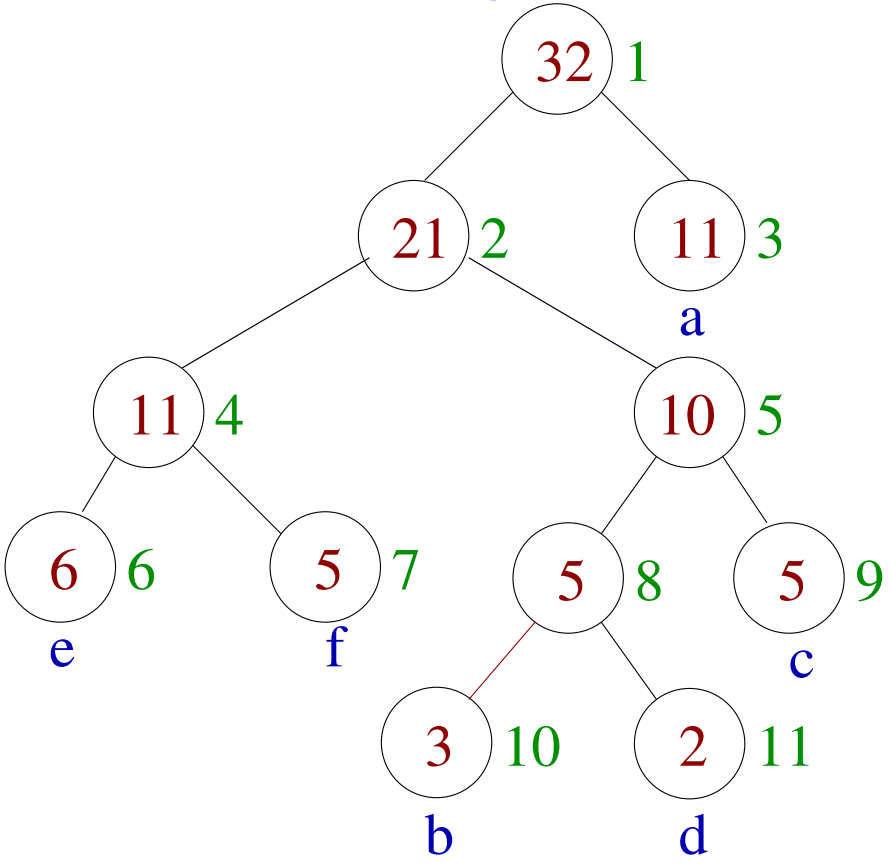
Rappel : Le code est dit irréductible, si chaque nœud possède 0 ou 2 fils.

Définition Soit un code préfixe d'une source discrète
- le *poids d'une feuille* est la probabilité de sa lettre
- le *poids d'un nœud* est la somme du poids de ses fils

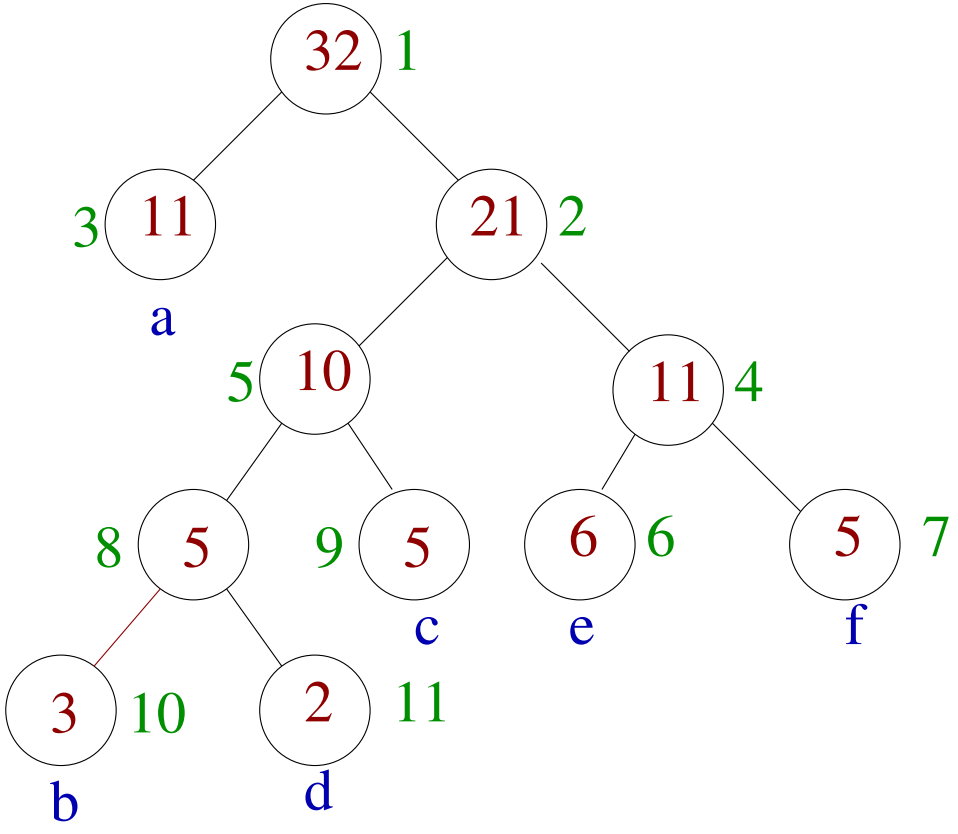
Définition Un *ordre de Gallager* $u_1, \ldots, u_{2K-1}$ sur les nœuds d'un code préfixe irréductible (d'une source de cardinal K) vérifie

1. les poids des u_i sont décroissants,
2. u_{2i} et u_{2i+1} sont frères pour tout i, $1 \leq i < K$.

Exemple



Exemple



Huffman adaptatif – Propriétés

Théorème 1. *[Gallager] Soit T un code préfixe d'une source X. T est un arbre de Huffman de X si et seulement si il existe un ordre de Gallager sur les sommets de T.*

Preuve

T Huffman $\Longrightarrow$ T admet un ordre de Gallager.

Les deux mots de poids le plus faible sont frères $\Rightarrow$fusionne pour obtenir un arbre avec une feuille de moins, soit $2(K-1)-1 = 2K-3$ nœuds.

L'arbre obtenu est un arbre de Huffmann, qui admet un ordre de Gallager (récurrence) $u'_1 \geq \cdots \geq u'_{2K-3}$.

Et le nœud fusionné est quelque part dans la liste. On prend ses deux fils, et on les met à la fin, ce qui donne un ordre de Gallager

$$u'_1 \geq \cdots \geq u'_{2K-3} \geq u_{2K-2} \geq u_{2K-1}$$

Preuve

T admet un ordre de Gallager $\implies$ T Huffman.

T a un ordre de Gallager $\implies$ nœuds ordonnés comme

$$u_1 \geq \cdots \geq u_{2K-3} \geq u_{2K-2} \geq u_{2K-1}$$

où u_{2K-2} et u_{2K-1} = feuilles frères de poids minimal.

Soit T' l'arbre $u_1, \ldots, u_{2K-3}$ il admet l'ordre $u_1 \geq \cdots \geq u_{2K-3}$. C'est donc un arbre de Huffman (récurrence).

Par l'algorithme de Huffman, l'arbre $u_1, \ldots, u_{2K-1}$ est de Huffman (car l'un des nœuds est la fusion de u_{2K-2} et de u_{2K-1}).

Principe de la mise à jour

Proposition 1. *Soient X_n la source au rang n et T_n un arbre de Huffman de cette source.*

Soit x la $n+1$-ème lettre du texte et soit $u_1, \ldots, u_{2K-1}$ un ordre de Gallager sur les nœuds de T_n.

Si $x \in X_n$ et si tous les nœuds $u_{i_1}, u_{i_2}, \ldots, u_{i_\ell}$ du chemin entre la racine et x sont les premiers de leur poids dans l'ordre de Gallager, alors T_n est un arbre de Huffman de X_{n+1}.

Preuve Il suffit de conserver le même ordre de Gallager.

Huffman adaptatif – Mettre à jour l'arbre

Soit T_n un arbre de Huffman au rang n et $u_1, \ldots, u_{2K-1}$ un ordre de Gallager sur ses nœuds.

Hypothèse : $x \in T_n$ (au noeud u).

Répéter tant que u n'est pas la racine

— soit $\tilde{u}$ le premier nœud classé avec le poids de u,

— échanger les noeuds u et $\tilde{u}$,

— échanger u et $\tilde{u}$ dans l'ordre de Gallager,

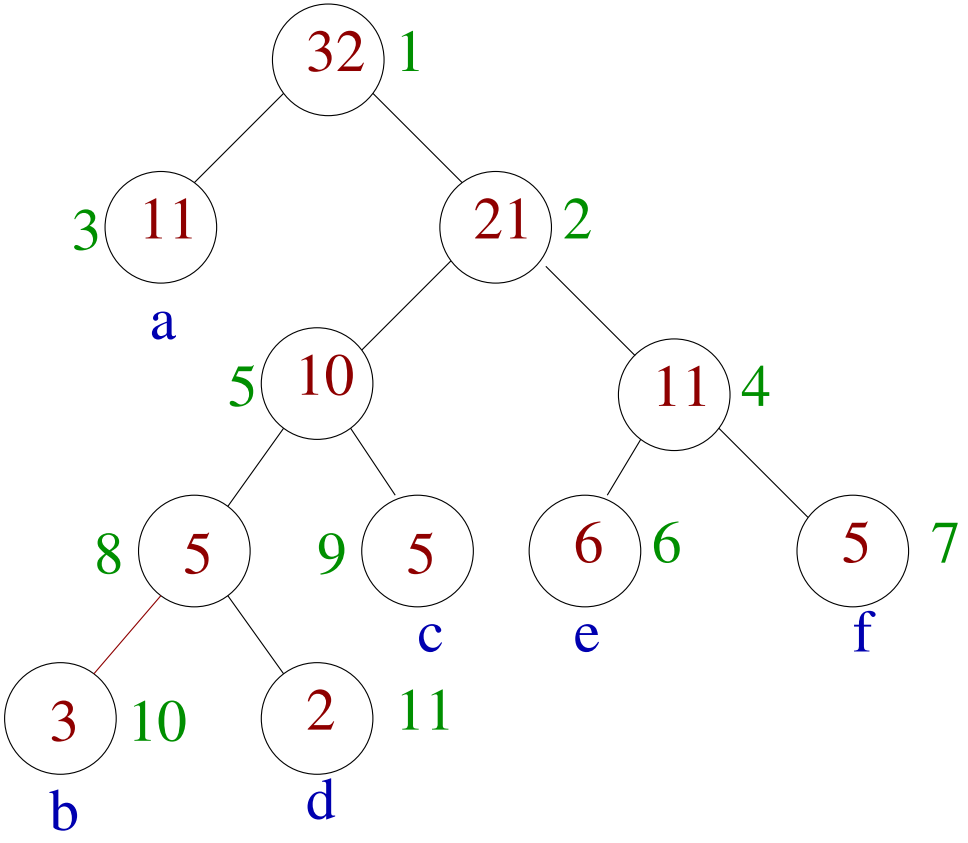
— incrémenter le poids de u *(poids = nb occurrences)*

— u devient le père de u

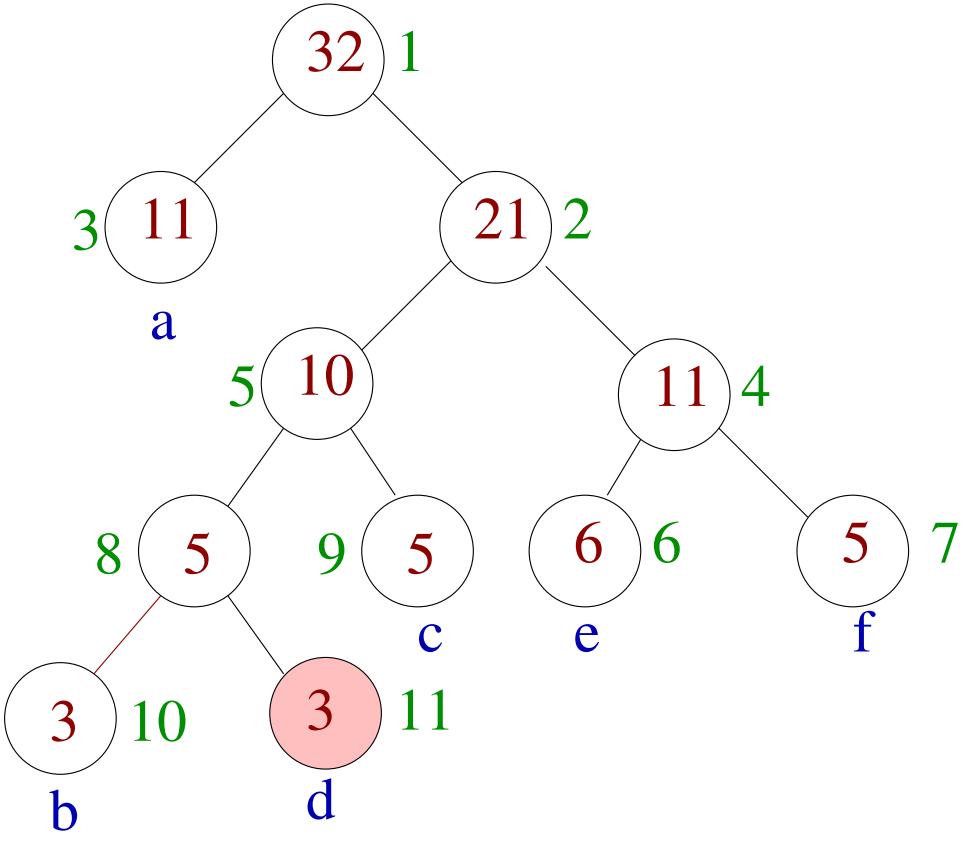
On est donc ramené au cas précédent.

Cet algorithme est du à D. Knuth.

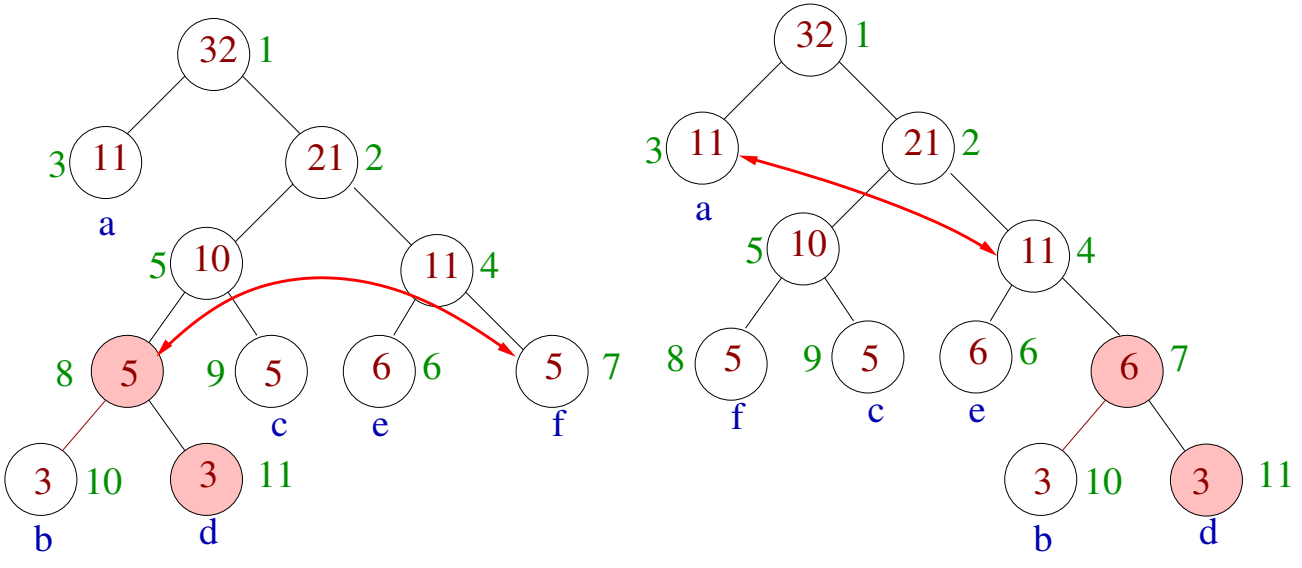
Exemple



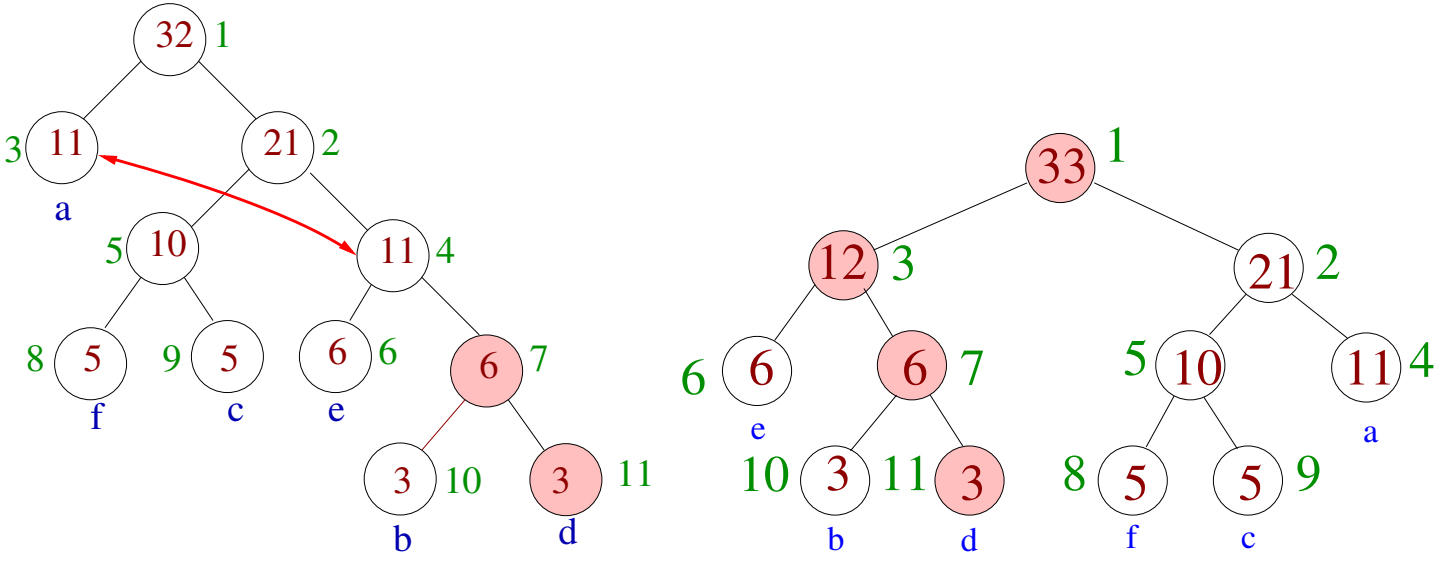
Exemple



Exemple

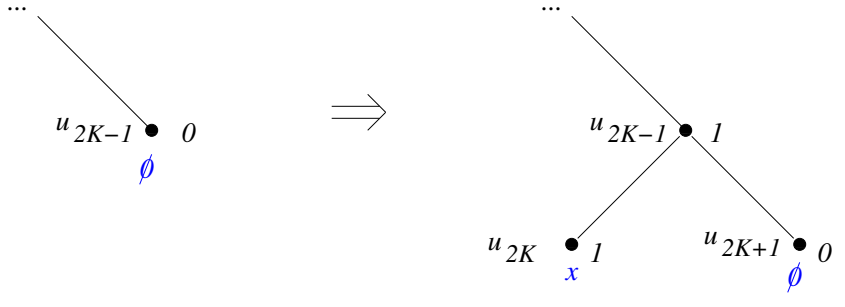

32 1
3 11
a
21 2
5 10
11 4
8 5
9 5
c
6 6
e
5 7
f
3 10
b
3 11
d
32 1
3 11
a
21 2
5 10
11 4
8 5
f
9 5
c
6 6
e
6 7
3 10
b
3 11
d

Exemple


32 1
3 11
21 2
a
5 10
11 4
8 5
9 5
6 6
6 7
f
c
e
3 10
3 11
b
d
1 33
12 3
21 2
6 6
6 7
5 10
11 4
e
a
10 3
11 3
8 5
5 9
b
d
f
c

Huffman adaptatif – Ajouter une feuille

Lorsque la lettre courante x n'est pas présente dans l'arbre, on effectue la mise à jour à partir de la feuille vide. On remplace ensuite celle-ci par un arbre à deux feuilles : l'une pour x, l'autre pour $\emptyset$.



Les deux nouveaux nœuds sont ajoutés à la fin de la liste (u_i).

Méthodes à base de dictionnaire

Principe : on maintient un dictionnaire (clé,chaîne de caractères).

On écrit les clés (indices) plutôt que les chaînes dans la sortie.

On espère que les clés (indices) seront plus courtes à coder que les chaînes.

Lempel-Ziv 78 – Principe

L'algorithme de Lempel-Ziv (1978) lit un texte constitué de symboles d'un alphabet $\mathcal{A}$. Supposons que nous ayons déja lu N symboles et que nous ayons formé un dictionnaire de mots déjà lus.

— à partir du $(N+1)$-ème symbole on lit les symboles un par un jusqu'à obtenir un mot (de longueur n) absent du dictionnaire, on imprime l'indice du dernier mot reconnu dans le dictionnaire (de longueur $n-1$) ainsi que le dernier symbole lu.

— On ajoute ce nouveau mot (de longueur n) au dictionnaire et on recommence à partir du $(N+n+1)$-ème symbole.

Lempel-Ziv 78 – Structure de données

Il nous faut une façon de représenter efficacement le dictionnaire. Celui-ci possède une propriété intéressante : si un mot est dans le dictionnaire, c'est aussi le cas de tous ses préfixes.

On en déduit que les dictionnaires que nous voulons représenter sont exactement les arbres q-aires. Une telle représentation nous donne une implémentation simple et efficace des fonctionnalités nécessaires, à savoir :

- tester la présence d'un mot,
- ajouter un mot

Lempel-Ziv 78 – Codage

Le dictionnaire (i.e. l'arbre) contient initialement le seul mot vide et sa taille est $K = 1$. On répète à partir de la racine jusqu'à épuisement :

— on parcourt l'arbre en lisant les lettres du texte (la i-ème lettre de $\mathcal{A}$ correspondant à la i-ème branche) tant que c'est possible.

Soient $b_1, \ldots, b_n, b_{n+1}$ les symboles lus, et soit i, $0 \leq i < K$ (K la taille du dictionnaire), l'indice du mot $(b_1, \ldots, b_n)$ dans le dictionnaire,

— $(b_1, \ldots, b_n, b_{n+1})$ est ajouté au dictionnaire avec l'indice K,

— on imprime i en base 2 sur $\lceil \log_2 K \rceil$ bits suivi du symbole b_{n+1}.

Lempel-Ziv 78 – Exemple

1	**0**	**0 1**	**0 1 1**	**1 0**	**0 0**	**1 1**	**1 0 0**

Dictionnaire		paire	
indices	mots lu	(indice,symbole)	mot de code
0			
1	**1**	(0,**1**)	**1**
2	**0**	(0,**0**)	**0 0**
3	**01**	(2,**1**)	**10 1**
4	**011**	(3,**1**)	**11 1**
5	**10**	(1,**0**)	**001 0**
6	**00**	(2,**0**)	**010 0**
7	**11**	(1,**1**)	**001 1**
8	**100**	(5,**0**)	**101 0**

Lempel-Ziv 78 – Décodage

Le dictionnaire est cette fois un tableau initialement de taille $K = 1$, sa seule entrée M_0 est le mot vide. On répète jusqu'à épuisement :

- on lit les $\lceil \log_2 K \rceil$ premiers bits du texte codé pour obtenir l'indice i. Soit M_i le mot d'indice i dans le dictionnaire,
- on lit le symbole suivant b,
- on ajoute une K-ème entrée au tableau/dictionnaire $M_K \leftarrow M_i \parallel b$,
- on imprime M_K.

Variante de Welsh

— Au démarrage tous les mots de une lettre sont dans le dictionnaire.
— Au lieu d'afficher la paire (i, b) on n'affiche que i.
— On ajoute le mot (i, b) dans le dictionnaire.
— On reprend la lecture à partir de b inclus.

$\Rightarrow$ variante légèrement plus efficace.

Elle est utilisée dans la commande unix `compress`, dans le format GIF87.

En pratique, elle compresse d'un facteur deux les textes en anglais de taille usuelle.

Lempel-Ziv-Welsh – Exemple

1 0 0 1 0 1 1 1 0 0 0 1 1 1 0 0 1 0 1 . . .

Dictionnaire		Mot reconnu		
indices	mots	mot	indice	mot de code
0	**0**			
1	**1**			
2	**10**	**1**	1	**1**
3	**00**	**0**	0	**00**
4	**01**	**0**	0	**00**
5	**101**	**10**	2	**010**
6	**11**	**1**	1	**001**
7	**110**	**11**	6	**110**
8	**000**	**00**	3	**011**
9	**011**	**01**	4	**0100**
10	**1100**	**110**	7	**0111**
11	**010**	**01**	5	**0101**

Lempel-Ziv-Welsh, décodage

Algorithme de codage :

1. on lit le texte jusqu'à trouver un mot m suivi d'une lettre a tels que m soit dans le dictionnaire, mais pas $m||a$,
2. on imprime l'indice de m dans le dictionnaire,
3. on ajoute $m||a$ dans le dictionnaire,
4. on continue à partir de la lettre a incluse.

Algorithme de décodage :

1. on lit un indice i,
2. on imprime le mot m d'indice i
3. on ajoute à la fin du dictionnaire le mot $m||a$ où a est la première lettre du mot suivant (on ne connaîtra a que lorsqu'on aura lu un indice de plus).

Lempel-Ziv 77

Cette variante est apparue avant celle présentée précédemment (qui date de 78). Elle est plus difficile à mettre en œuvre.

On suppose que N bits ont été lus.

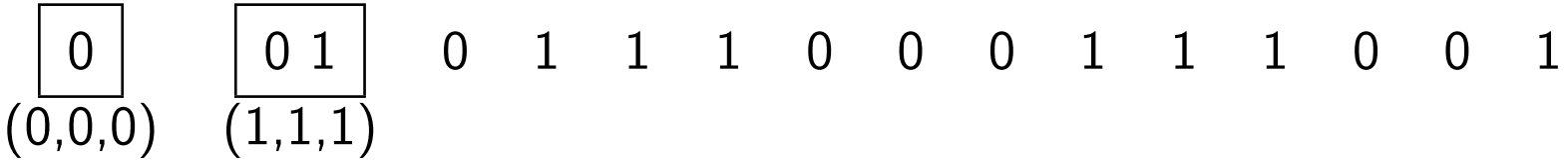
On lit à partir du $N+1$-ème bit le plus long mot (de longueur n) qui soit un sous-mot commençant à un certain i-ème bit avec $i \leq N$. Ce mot est codé par le triplet (i, n, b), où b est le bit suivant le mot.

En pratique, on l'implémente à l'aide d'une fenêtre glissante de taille fixe : le dictionnaire est l'ensemble des sous-mots de cette fenêtre.

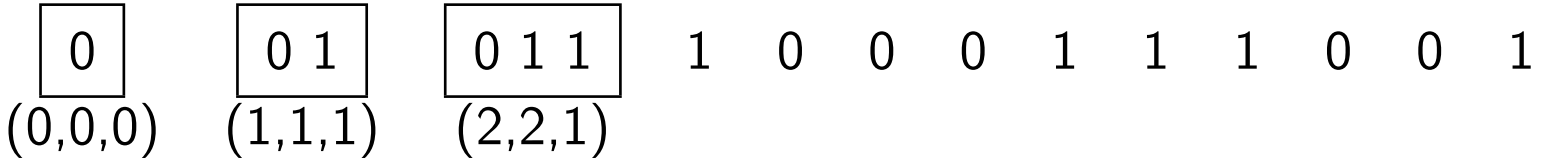
Elle est utilisée dans `gzip, zip`.

Exemple

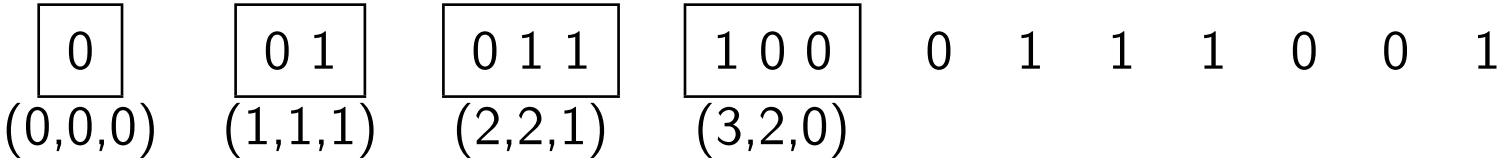
Exemple



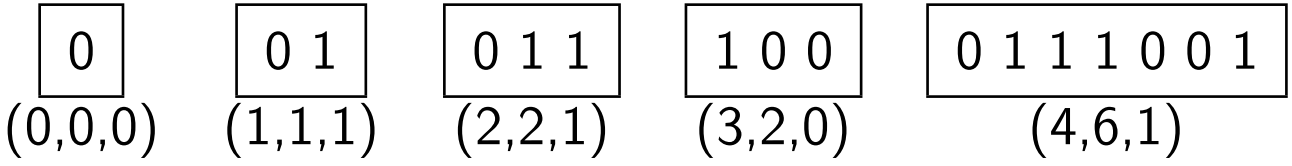
Exemple



Exemple



Exemple



Source stationnaire

Définition Une source est dite *stationnaire* si son comportement ne varie pas lorsque l'on décale l'observation dans le temps. Pour tous entiers positifs n et j, et tout $(x_1, \ldots, x_n) \in \mathcal{X}^n$

$$p_{X_1 \ldots X_n}(x_1, \ldots, x_n) = p_{X_{1+j} \ldots X_{n+j}}(x_1, \ldots, x_n)$$

Théorème 2. *Pour toute source stationnaire, les limites ci-dessous existent et sont égales*

$$H(\mathcal{X}) = \lim_{n \to \infty} \frac{1}{n} H(X_1, \ldots, X_n) = \lim_{n \to \infty} H(X_n \mid X_1, \ldots, X_{n-1}).$$

La quantité $H(\mathcal{X})$ est appelée entropie par lettre.

La propriété fondamentale des algorithmes de Lempel-Ziv

Théorème 3. *Pour toute source $\mathcal{X}$ stationnaire et ergodique le taux de compression tend vers $H(\mathcal{X})$ avec probabilité 1 quand la taille de la séquence à compresser tend vers l'infini.*