# Compression (M1) – LZ78 & LZW

C. Nicaud

Dans ce chapitre, on s'intéresse à un autre algorithme de compression adapté au textes, inventé également par Lempel et Ziv, ainsi qu'à une de ses variantes.

## 1 Principe des algorithmes

Ce sont des algorithmes de compression "avec dictionnaire". Comme pour LZ77, quand on reconnaît un mot de notre dictionnaire on l'encode par un numéro (son index dans le dictionnaire) pour gagner de la place. La constitution du dictionnaire est dynamique pour s'adapter automatiquement au texte à compresser.

Dans LZ77, le dictionnaire était constitué de la fenêtre des $W$ caractères précédents. Dans LZ78 et LZW, on va créer et maintenir un dictionnaire en y ajoutant un mot à chaque étape.

## 2 LZ78

Le principe est le suivant:

- On commence avec un dictionnaire qui ne contient que $\epsilon \mapsto 0$ : cela signifie qu'initialement il n'y a que le mot vide dans le dictionnaire, et que son indice est 0.
- A chaque étape on cherche le plus long préfixe $v$ qui est dans le dictionnaire (donc tel que le préfixe suivant $va$ n'est pas dans le dictionnaire) puis:
  - On émet le couple $(Dico[v], \alpha)$, où $Dico[v]$ est l'indice de $v$ dans le dictionnaire.
  - On ajoute $v\alpha$ dans le dictionnaire, d'indice la taille du dictionnaire moins 1.
  - On repart dans le texte après $v\alpha$.

L'algorithme émet donc une séquence de couples $(i, c)$ où $i$ est un indice dans le dictionnaire, et $c$ un caractère.

**Remarque :** à la toute fin, si la totalité du mot est dans le dictionnaire, on prend le mot sans la dernière lettre pour $v$, pour qu'il reste un caractère $\alpha$ derrière (et ainsi ne pas avoir à ajouter un symbole "pas de caractère").

**Exemple :** considérons le mot $u = abaaaabaab$, la barre `|` indique où on en est dans le texte.

| texte | v | alpha | ajout au dico |
|---|---|---|---|
| (init) | | | eps->0 |
| \|abaaaabaaa | eps | a | a->1 |
| a\|baaaabaaa | eps | b | b->2 |
| ab\|aaaabaaa | a | a | aa->3 |
| abaa\|aabaab | aa | b | aab->4 |
| abaaaab\|aab | aa | b | |

Donc la sortie sera: $(0, a), (0, b), (1, a), (3, b), (3, b)$.

Pour la décompression, il s'agit de reconstituer le dictionnaire exactement de la même façon que l'algorithme de compression. Cela ne présente pas de difficulté particulière.

**Exemple :** considérons la séquence $(0, a), (0, b), (1, a), (3, b), (3, b)$:

```
paire    mot produit   ajout au dico
init                   0->eps
(0,a)    a             1->a
(0,b)    b             2->b
(1,a)    aa            3->aa
(3,b)    aab           4->aab
(3,b)    aab
```

On retrouve bien le mot $u = abaaaabaab$.

# 3 LZW

C'est une variante de l'algorithme de Lempel-Ziv 78, inventée en 1984 par Welch. Il est notamment utilisé dans le format d'image `gif`.

L'idée est la même que pour LZ78, sauf que :

- On connaît l'alphabet et on commence avec tous les mots d'une lettre dans le dictionnaire (et non juste le mot vide comme pour LZ78).
- A chaque étape, on cherche $v$ et $\alpha$ comme pour LZ78, mais on n'encode que $v$, par son numéro. On ajoute toujours $v\alpha$ dans le dictionnaire ensuite, et on repart de $\alpha$ puisqu'il n'a pas été encodé.

**Exemple :** considérons le mot $u = abababaab$ sur l'alphabet $A = \{a, b\}$. La barre `|` indique où on en est dans le texte. Au début, le dictionnaire contient $a \mapsto 0$ et $b \mapsto 1$.

```
texte         v      alpha  sortie  ajout au dico
(init)                                a->0, b->1
|abababaab    a      b      0       ab->2
a|bababaab    b      a      1       ba->3
ab|ababaab    ab     a      2       aba->4
abab|abaab    aba    a      4       abaa->5
abababa|ab    ab     .      2
```

Donc la sortie sera: $(0, 1, 2, 4, 2)$.

**Attention (seule subtilité du chapitre) :** à la décompression, on a un temps de retard dans la création du dictionnaire. Cela est du au fait que dans l'algorithme de compression, on ajoute $v\alpha$ dans le dictionnaire, mais on n'encode que $v$. Donc il faut attendre l'étape d'après pour identifier $\alpha$, la première lettre du mot suivant, et retrouver quel mot avait été ajouté dans le dictionnaire à l'étape précédente.

Essayons de décompresser $(0, 1, 2, 4, 2)$ sur l'alphabet $A = \{a, b\}$:

```
Num    v    ajout au dico
init        0->a, 1->b
0      a    (il faut attendre le mot suivant pour identifier sa 1ere lettre)
1      b    2->ab (le v de la ligne au-dessus, suivi de la 1ère lettre du v actuel)
2      ab   3->ba (le v de la ligne au-dessus, suivi de la 1ère lettre du v actuel)
4      ???
```

Et là, on est coincé : comme on a un temps de retard dans la constitution du dictionnaire, le mot d'indice 4 n'est pas encore connu. Heureusement, il y a une solution à ce problème. Si on appelle $v$ le mot de cette étape, alors on met ensuite dans le dictionnaire `4->ab`$\alpha$, où $\alpha$ est la première lettre de $v$. Cela veut dire que $v$ commence par $a$, et donc on peut trouver $v = aba$!
**La règle est générale :** si a un moment dans la décompression on a besoin d'un mot qui n'est pas dans le dictionnaire, alors le mot dont on a besoin est le mot $v\alpha$, où $v$ est le mot produit à l'étape précédente et $\alpha$ est la première lettre de $v$.

Si on reprend l'exemple :

```
Num    v    ajout au dico
init        0->a, 1->b
0      a    (il faut attendre le mot suivant pour identifier sa 1ere lettre)
1      b    2->ab (le v d'au-dessus, suivi de la 1ère lettre du v actuel)
2      ab   3->ba (le v d'au-dessus, suivi de la 1ère lettre du v actuel)
4      aba  4->aba (exception, pas d'indice 4, on utilise la règle ci-dessus)
2      ab
```

Et on retrouve bien le mot *abababaab*.

# 4 Remarques diverses

D'un point de vue algorithmique, le plus efficace pour encoder le dictionnaire à la compression est d'utiliser un arbre prefixiel, (voir `https://fr.wikipedia.org/wiki/Trie_(informatique)`). Cependant, une table associative (ou hashtable en Java) est en général suffisamment rapide en pratique.

Si on veut utiliser un codage de longueur fixe pour représenter les couples de la sortie de LZ78 ou les indices de la sortie de LZW, il faut fixer une taille maximale pour le dictionnaire. Il faut également une stratégie pour décider de quoi faire quand le dictionnaire est plein. Il existe trois stratégies principales :

(i) ne plus modifier le dictionnaire une fois qu'il est plein ;

(ii) ré-initialiser le dictionnaire s'il est plein : avec juste le mot vide pour LZ78 et juste l'alphabet pour LZW ;

(iii) enlever les feuilles de l'arbre préfixiel, c'est-à-dire les mots du dictionnaire qui ne sont préfixe d'aucun autre mot du dictionnaire.

Toutes ces méthodes ne posent aucun problème à la décompression, à condition que le décompresseur connaisse la taille maximale et la stratégie choisie en cas de saturation du dictionnaire.

# 5 Exercices (vérifiez que vous arrivez à les faire)

1. Compressez avec LZ78 et avec LZW le mot $u = bbbabbaabbbb$.
2. Décompressez `(0, 'b'), (0, 'a'), (2, 'a'), (3, 'b'), (4, 'a')` obtenu après application de LZ78.
3. Décompressez `[1,2,0,4,1]` obtenu après application de LZW pour l'alphabet $\{a, b\}$.

## 6 Correction des exercices

1. On doit trouver :
   - pour LZ78 : `(0, 'b'), (1, 'b'), (0, 'a'), (2, 'a'), (3, 'b'), (2, 'b')`
   - pour LZW : `[1, 2, 0, 3, 4, 2, 1]`
2. On doit trouver *baaaaabaaba*.
3. On doit trouver *bbbaaab*.