Compression
Compression par dictionnaires

E. Jeandel

Compression par dictionnaire

Principe :
- Avoir une liste des mots “fréquents” ;
- Lorsqu’on trouve un mot dans la liste, remplacer ce mot par sa position dans la liste.

Deux types de fonctionnement :
- Dictionnaire calculé une fois pour toute ;
- Dictionnaire qui évolue

Texte français
- Le français contient de l’ordre de 200000 mots.
- Pour coder tous les mots, il suffit de 18 bits ($2^{18} = 262144$)
- Comme un mot français fait de l’ordre de 5 caractères, on peut gagner un facteur de l’ordre de 55% (et seulement 30% pour notre code sur 5 bits)

En pratique, on gagnera beaucoup moins puisqu’on trouve, même dans un texte français, autre chose que ces 200000 mots (ponctuation, noms propres…).

Compression LZ
- Ziv et Lempel ont inventé en 1977 et 1978 deux algorithmes de compression faisant usage de dictionnaire.
- On va les étudier ici, ainsi que certaines de leurs variantes.

1 LZ78

LZ78

Principe :
- On a un dictionnaire qu’on met à jour progressivement
- À chaque étape, on cherche le plus cours mot non présent dans le dictionnaire.

1. t	2. s	3. e
4. th	5. ta	6. ev
7. sa	8. the	9. sat
10. theo	11. evas	12. theor

theoreme de parseval

- On écrit la position du mot trouvé, ainsi que la lettre à ajouter
  (10,r)
- On écrit le nouveau mot dans le dictionnaire.
- Et on continue à partir de la suite

LZ78 - Exemple

Le résultat de l’algorithme est (0, v)(0, e)(0, r)(0, i)(0, d)(4, q)(0, u)(2, ␣)(0, !)(0, ␣)(5, o)(0, m)(4, n)(6, u)(8, p)(14, e)(10, n)(16, ␣)(2, n)(10, t)(7, n)(16, .)

qu’on obtient comme suit

Lu		Ajout dans le dictionnaire	Écrit
v	non trouvé	1 v	(0, v)
e	non trouvé	2 e	(0, e)
r	non trouvé	3 r	(0, r)
i	non trouvé	4 i	(0, i)
d	non trouvé	5 d	(0, d)
i	trouvé en position 4		
iq	non trouvé	6 iq	(4, q)
u	non trouvé	7 u	(0, u)
e	trouvé en position 2		
e␣	non trouvé	8 e␣	(2, ␣)
!	non trouvé	9 !	(0, !)
␣	non trouvé	10 ␣	(0, ␣)
d	trouvé en position 5		
do	non trouvé	11 do	(5, o)
m	non trouvé	12 m	(0, m)
i	trouvé en position 4		
in	non trouvé	13 in	(4, n)
i	trouvé en position 4		
iq	trouvé en position 6		
iqu	non trouvé	14 iqu	(6, u)
e	trouvé en position 2		
e␣	trouvé en position 8		
e␣p	non trouvé	15 e␣p	(8, p)
i	trouvé en position 4		
iq	trouvé en position 6		
iqu	trouvé en position 14		
ique	non trouvé	16 ique	(14, e)
␣	trouvé en position 10		
␣n	non trouvé	17 ␣n	(10, n)
i	trouvé en position 4		
iq	trouvé en position 6		
iqu	trouvé en position 14		
ique	trouvé en position 16		
ique␣	non trouvé	18 ique␣	(16, ␣)
e	trouvé en position 2		
en	non trouvé	19 en	(2, n)
␣	trouvé en position 10		
␣t	non trouvé	20 ␣t	(10, t)
u	trouvé en position 7		
un	non trouvé	21 un	(7, n)
i	trouvé en position 4		
iq	trouvé en position 6		
iqu	trouvé en position 14		
ique	trouvé en position 16		
ique.	non trouvé	22 ique.	(16, .)

LZ78 - Codage
- Il faut maintenant décider comment coder les paires (index, symbole).
- Le symbole sera codé sur 8 bits (ici 5 bits)
- L’indice sera codé sur le plus petit nombre de bits possible : Si le dictionnaire est de taille n à un instant donné, on codera l’indice sur $\lceil \log_2 n \rceil$ bits.

LZ78 - Mise en oeuvre (Python)

```
current = ''
tailledict=0
dict = {'': 0}
for c in texte:
  if (current+c) in dict:
    current+=c
  else:
    print dict[current], c
    tailledict+=1
    dict[current+c] = tailledict
    current = ''
```

LZ78 - Mise en oeuvre

LZ78 nécessite de savoir trouver facilement, dans un dictionnaire si un mot est présent. Soit `dict` le dictionnaire et soit T un tableau à deux entrées. $T[i][j]$ correspond à l’indice du mot `dict`$[i]j$ dans `dict`, et vaut -1 si ce mot n’est pas dans le dictionnaire.

La recherche s’écrit maintenant ainsi, où N désigne la taille du dictionnaire à un instant donné.
- i = 0.
- lire un caractère c
- Si $T[i][c]$ est différent de -1, alors $i = T[i][c]$, et lire un nouveau caractère.
- Sinon
  - écrire (i, c) ;
  - mettre $T[i][c]$ à la valeur $N + 1$;
  - mettre $T[N + 1][j]$ à la valeur -1 pour tout j ;
  - Incrémenter N

On a plus besoin du dictionnaire !

LZW

LZW (W pour Welsh) est une variante de LZ78. On s’aperçoit que dans LZ78 on écrit trop de trucs (en particulier des caractères). Comment faire mieux ?
- LZW part avec un dictionnaire qui contient toutes les lettres de l’alphabet ;
- Si on trouve le mot `theo` dans le dictionnaire, mais pas le mot `theor`, on écrit l’indice du mot `theo` et on reprend la lecture au *r compris*.

LZW - Exemple

Lu		Ajout dans le dictionnaire	Écrit
v	trouvé en position 22		
ve	non trouvé	32 ve	22
er	non trouvé	33 er	5
ri	non trouvé	34 ri	18
id	non trouvé	35 id	9
di	non trouvé	36 di	4
iq	non trouvé	37 iq	9
qu	non trouvé	38 qu	17
ue	non trouvé	39 ue	21
e␣	non trouvé	40 e␣	5
␣!	non trouvé	41 ␣!	0
!␣	non trouvé	42 !␣	30
␣d	non trouvé	43 ␣d	0
do	non trouvé	44 do	4
om	non trouvé	45 om	15
mi	non trouvé	46 mi	13
in	non trouvé	47 in	9
ni	non trouvé	48 ni	14
iq	trouvé en position 37		
iqu	non trouvé	49 iqu	37
ue	trouvé en position 39		
ue␣	non trouvé	50 ue␣	39
␣p	non trouvé	51 ␣p	0
pi	non trouvé	52 pi	16
iq	trouvé en position 37		
iqu	trouvé en position 49		
ique	non trouvé	53 ique	49
e␣	trouvé en position 40		
e␣n	non trouvé	54 e␣n	40
ni	trouvé en position 48		
niq	non trouvé	55 niq	48
qu	trouvé en position 38		
que	non trouvé	56 que	38
e␣	trouvé en position 40		
e␣e	non trouvé	57 e␣e	40
en	non trouvé	58 en	5
n␣	non trouvé	59 n␣	14
␣t	non trouvé	60 ␣t	0
tu	non trouvé	61 tu	20
un	non trouvé	62 un	21
ni	trouvé en position 48		
niq	trouvé en position 55		
niqu	non trouvé	63 niqu	55
ue	trouvé en position 39		
ue.	non trouvé	64 ue.	39
.□	non trouvé	65 .□	27

LZW - Décodage

Comment décoder ?

- Lorsqu'on lit un symbole compressé, on ne sait pas quoi ajouter dans le dictionnaire : cette information n'arrive qu'après avoir lu le symbole suivant ;
- Exemple : 4 15 32 0 12 29
- Exemple qui marche mal : 1 2 3 32 35 4

LZ78 et LZW - Remarques

- Que faire lorsque le dictionnaire (la mémoire) est plein(e) ?
  - Le vider totalement (ce qui revient à couper le texte et à compresser chacune des parties séparemment) ;
  - Ne plus y toucher ;
  - Supprimer des mots. Comment ?

  Unix `compress` ne touche pas au dictionnaire. Cependant, s'il s'aperçoit que la compression devient mauvaise, il supprime totalement le dictionnaire.

2 LZ77

LZ77

- LZ77 n'a pas de dictionnaire proprement dit, mais se sert des k caractères lus précédemment comme dictionnaire

  veridique␣!␣dominique␣pique␣nique␣en␣tunique.
  veridique␣!␣dominique␣pique␣nique␣en␣tunique.
- Quand on a trouvé la plus longue partie commune, on écrit sa position, sa longueur et le caractère qui suit :

  (9,5,n)
- Puis on continue

LZ77 - Exemple

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 v

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 e

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 r

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 i

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 d

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 3 1 q

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 u

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 1 1 ␣

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 !

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 9 1 d

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 o

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 m

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 3 1 n

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 3 5 p

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 9 5 n

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 3 5 e

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 8 1 ␣

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 0 0 t

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 2 1 n

`veridique␣!␣dominique␣pique␣nique␣en␣tunique.` 4 4 .

LZ77 - Codage

- Comme la taille de la fenêtre est fixe (ici, disons 16 caractères), on peut coder longueur et position par un nombre fixe de bits (ici 4).
- Chaque code aura donc une longueur fixe, ici de $4+4+5$ bits (4 pour la longueur, 4 pour la position et 5 pour le caractère)

Signalons aussi que, pour améliorer la vitesse d'exécution du programme, LZ77 n'essaie pas de trouver des parties communes de longueur trop grande (en pratique on cherche des parties de taille 32 pour une fenêtre de taille 2000)

LZ77 - Variantes

- Il existe des tas de variantes de LZ77. LZ77 utilise beaucoup trop de bits dans le cas où on n'a pas réussi à retrouver le caractère : $(0, 0, a)$ fait beaucoup trop de bits par rapport à l'information qu'il contient.
- LZSS utilise un bit pour signaler si on a trouvé une partie commune ou non. Lorsqu'on a trouvé cette partie commune, LZSS recommence au caractère non trouvé (contrairement à LZ77). Sinon, LZSS écrit le caractère non trouvé.
- Deflate (`zip`,`gzip`) opère de façon similaire, mais utilise des codes de Huffman (fixes ou calculés à la volée) pour encoder les différents types d'éléments

3 Conclusion

Comparaisons

- LZ77 a un caractère local : Si la taille de la fenêtre est trop petite, on peut ne pas voir qu'on pourrait compresser ;
- LZ78 a un problème similaire dû à la mémoire limitée.
- La trop grande mémoire de LZ78 peut aussi être un problème : Si le fichier est constitué de deux parties différentes, le dictionnaire sera "encombré" inutilement lorsqu'on lira la deuxième partie

Utilisation

- Ces algorithmes sont suffisament efficaces pour être utilisés directement : c'est le cas des logiciel `zip` ou `gzip` ;
- On les utilise également dans d'autres formats de fichiers, comme les fichiers OpenOffice (deflate) ou les fichiers PDF/PostScript (deflate, LZW)
- Signalons aussi LZEXE qui permet de compresser des fichiers exécutables.

A noter que LZW était breveté jusque fin 2003 ce qui posait des problèmes quant à son utilisation.