Codage et Traitement des Données Numériques

-

Compression

Emmanuel Jeandel (emmanuel.jeandel@lif.univ-mrs.fr)
`http://www.lif.univ-mrs.fr/~ejeandel/enseignement.html`

8 février 2011

1 Un algorithme efficace

On considère ici l'algorithme de compression suivant, sur une chaine de départ x :
- L'algorithme compte le nombre d'occurrences de chaque caractère c dans x et écrit le résultat
- L'algorithme écrit ensuite la position de x dans l'ordre alphabétique parmi toutes les chaînes ayant exactement le même nombre d'occurences de chaque caractère que x.

Q 1) Exécuter l'algorithme sur l'entrée `abaaaba`.

On suppose pour simplifier que notre texte ne contient que les lettres `a,b,c` (mais le raisonnement se généralise). On note N_a le nombre d'occurrences de la lettre a. On a donc $N_a+N_b+N_c = N$, où N est la longueur du mot initial. On note p_a la fréquence d'apparition de la lettre a, c'est à dire $p_a = N_a/N$.

Q 2) Exprimer en fonction de tous ces paramètres le nombre de mots ayant exactement N_a occurrences de la lettre a, N_b occurrences de la lettre b et N_c occurrences de la lettre c. On appelle G ce nombre.

(On rappelle que $C_n^k = \frac{n!}{(n-k)!k!}$ représente le nombre de façons de choisir k objets parmi n)

Q 3) Le nombre de bits nécessaire pour stocker la position de x est donc de $\log_2 G$. Calculer $\log_2 G$ et l'exprimer (approximativement) en fonction de l'entropie.

On rappelle que si n est assez grand $\log_2 n! \approx n \log_2 n - n \log_2 e$.

Q 4) En déduire que cet algorithme est meilleur que l'algorithme de Huffman.

2 Burrows-Wheeler

L'algorithme de Burrows-Wheeler est un algorithme de compression au fonctionnement très simple et très efficace. C'est celui qui est utilisé, entre autres, dans le logiciel `bzip2`.

L'algorithme utilisé dans bzip2 fonctionne en trois temps

- Il applique tout d'abord la transformation de Burrows-Wheeler ;
- Il applique ensuite une transformation par liste ;
- Il code le résultat obtenu avec un codeur d'entropie (Huffman).

On s'intéresse ici aux deux premiers points. Il est à noter que ces deux points ne diminuent pas la taille du texte, mais en quelque sorte "préparent" le texte afin que la phase de codage de l'entropie soit efficace.

2.1 Burrows-Wheeler

L'algorithme de Burrows-Wheeler est d'une simplicité extrême. On va expliquer son fonctionnement sur un exemple facile, le "mot" rantanplan.

- On calcule toutes les permutations circulaires du mot rantanplan :
  ```
  rantanplan
  antanplanr
  ntanplanra
  tanplanran
  anplanrant
  nplanranta
  planrantan
  lanrantanp
  anrantanpl
  nrantanpla
  ```

- On les classe par ordre alphabétique :
  ```
  anplanrant
  anrantanpl
  antanplanr
  lanrantanp
  nplanranta
  nrantanpla
  ntanplanra
  planrantan
  rantanplan
  tanplanran
  ```
- Le résultat de l'algorithme est alors constitué :
  - De la dernière colonne du tableau ci-dessus (c'est à dire : des dernières lettres des mots, lorsqu'ils sont classés par ordre alphabétique). Dans notre cas, il s'agit de `tlrpaaannn`
  - De l'index du mot original (rantanplan) dans le tableau (ici le mot est en neuvième position, donc l'index est 9)

Q 1) Effectuer l'algorithme sur le mot `repetitif`

Q 2) Expliquer intuitivement pourquoi le résultat de cet algorithme semble plus facile à compresser que le mot initial.

Q 3) Expliquer comment on peut retrouver, uniquement à partir du resultat de l'algorithme, le mot de départ. On commencera par remarquer qu'on obtient, à partir du résultat, toutes les lettres du mot, et donc qu'on peut en déduire la première colonne du tableau.

Soit D la dernière colonne du tableau obtenu et soit P la première.

Q 4) Montrer qu'on peut déduire P de D.

Soit T un tableau tel que $P[T[i]] = D[i]$. T indique où se trouve dans P le $i$ème caractère de D.

Q 5) Montrer que dans la chaîne de départ, $D[i]$ précède $P[i]$.

Q 6) En déduire comment reconstruire la chaîne initiale à partir de D et T.

Q 7) Trouver le mot de départ, sachant que le résultat est `mpehsoohmmroi`, et l'index 3. On expliquera la démarche, et pas uniquement quel est le résultat.

Q 8) Expliquer pourquoi appliquer uniquement cet algorithme puis un codeur d'entropie type Huffmann n'est pas intéressant.

2.2 Move-to-Front - Une transformation de liste

On rappelle qu'on travaille avec un alphabet comportant seulement 32 lettres.

␣	00000	0	h	01000	8	p	10000	16	x	11000	24
a	00001	1	i	01001	9	q	10001	17	y	11001	25
b	00010	2	j	01010	10	r	10010	18	z	11010	26
c	00011	3	k	01011	11	s	10011	19	.	11011	27
d	00100	4	l	01100	12	t	10100	20	,	11100	28
e	00101	5	m	01101	13	u	10101	21	’	11101	29
f	00110	6	n	01110	14	v	10110	22	!	11110	30
g	00111	7	o	01111	15	w	10111	23	?	11111	31

Le principe de la transformation par liste est la suivante :
- On part avec un tableau T contenant tous les caractères de l'alphabet
- On lit les lettres du mot d'entrée une par une. A chaque étape, on écrit la position de cette lettre dans le tableau T, et on la déplace en tête du tableau T.

Q 9) Exécuter l'algorithme sur l'entrée `tlrpaaannn`.

Q 10) Expliquer pourquoi cet algorithme produit un texte facile à compresser par une méthode de compression statistique lorsque le texte de départ a déjà été transformé par la transformation Burrows-Wheeler.

Q 11) Décompresser (transformation de liste puis Burrows-Wheeler) le mot $20, 15, 0, 5, 2, 16, 0, 2, 12, 9, 19, 8$ sachant que l'index est 3.

3 TP

Q 1) Ecrire un programme C (ou python) qui effectue la transformation de Burrows-Wheeler, puis appliquez-le sur un exemple.

- Si vous écrivez toutes les permutations circulaires du mot de départ, vous risquez de prendre trop de mémoire, essayez de réfléchir à une autre façon de faire.
- Il ne vous est pas demandé d'utiliser un tri compliqué, vous pouvez vous contenter d'un algorithme simple. Pour les plus expérimentés d'entre vous, vous pouvez essayer d'utiliser la fonction `qsort` (en C) ou `sort` (plus simple d'utilisation, en python)
- Pour pouvoir comparer des chaînes de caractères (en C), utilisez les fonctions `strcmp` et/ou `strncmp` ou `memcmp`.

Q 2) Ecrire un programme C (ou python) qui effectue la transformation de liste, puis appliquez-le sur un exemple.