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 LZ

Q 1) Effectuer la compression de la chaîne `abcababcabbc.` par les algorithmes LZ78 et LZ77 (on prendra pour ce dernier une fenêtre de taille 4).

Vous trouverez en annexe l'exécution de l'algorithme LZ78 sur un paragraphe pris dans un texte de Jules Verne. On le représente par un tableau T qui contient dans la case $T[i]$ l'indice et le caractère obtenu lors de la $i$ème étape de l'algorithme LZ78. (Autrement dit, le résultat de l'algorithme LZ78 est $T[1]T[2]T[3]\dots$)

Q 2) Trouvez les trois premiers mots et les deux derniers mots du texte. Expliquer votre démarche

Q 3) Donner deux chaînes de caractères qui se décompressent en le même résultat par LZ78. Même question avec LZ77.

Q 4) Compresser avec LZ78 la chaîne suivante (elle a 17 lettres) :

`aaababcabcdabcdeabcdef`

Q 5) Montrer qu'on peut la compresser (en gardant le même décompresseur) en beaucoup moins de symboles.

Q 6) Montrer que la même chaîne avec un `a` en plus au début se compresse mieux.

2 MNP5

Il existe plusieurs protocoles de compression utilisés par les modems. Nous nous intéressons ici au protocole MNP5. On suppose maintenant que l'alphabet est l'alphabet à 32 symboles que voici :

␣	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

Bien entendu, le vrai protocole utilise l'alphabet usuel à 256 symboles.

L'algorithme est en deux phases.

La première phase fait partie de la famille appelée compressions RLE (Run-Length Encoding). Le principe est très simple : On recopie chaque caractère. Si on constate une répétition de au moins 3 fois le même caractère, on insère 3 fois ce caractère suivi du nombre de fois où il apparaît dans la suite.

Ainsi, par exemple la chaîne suivante :

`bbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaacccda`

se transformera en `bbaaa,ccc␣da`

Le caractère `,` signifie donc qu'il faut encore insérer 28 lettres `a`, le caractère ␣ qu'il faut encore insérer 0 fois la lettre `c`..

Q 1) Expliquer comment décompresser (Il n'est pas demandé d'écrire un algorithme).

Q 2) Quel est le taux de compression maximum et minimum qu'on peut atteindre en utilisant cette méthode ? Donner un exemple où on atteint ce maximum, et un exemple où on atteint ce minimum.

Une fois passée cette première phase, la seconde partie de l'algorithme est une "variante" d'un algorithme de type Huffman.

On utilise pour cela un tableau, rempli initialement ainsi.

0	␣	000	0	h	11000	0	p	11110000	0	x	11111000
0	a	001	0	i	11001	0	q	11110001	0	y	11111001
0	b	010	0	j	11010	0	r	11110010	0	z	11111010
0	c	011	0	k	11011	0	s	11110011	0	.	11111011
0	d	1000	0	l	111000	0	t	11110100	0	,	11111100
0	e	1001	0	m	111001	0	u	11110101	0	’	11111101
0	f	1010	0	n	111010	0	v	11110110	0	!	11111110
0	g	1011	0	o	111011	0	w	11110111	0	?	11111111

Le tableau contient dans sa première colonne le nombre de fois où on a rencontré un caractère (son nombre d'occurence), et dans sa troisième colonne comment on va coder un caractère.

L'algorithme est alors le suivant : A chaque fois qu'on lit un caractère, on écrit à la place son code, et on incrémente son nombre d'occurence. On permute ensuite les caractères (sans toucher aux codes) de sorte à trier le tableau par nombre d'occurence. Par exemple, examinons ce qui se passe sur `pqqpprrrrr`. A la lecture du premier `p` on insère son code `11110000`, et on incrémente son nombre d'occurence, qui passe donc à 1. On trie alors le tableau et p passe en tête.

1	p	000	0	g	11000	0	o	11110000	0	x	11111000
0	␣	001	0	h	11001	0	q	11110001	0	y	11111001
0	a	010	0	i	11010	0	r	11110010	0	z	11111010
0	b	011	0	j	11011	0	s	11110011	0	.	11111011
0	c	1000	0	k	111000	0	t	11110100	0	,	11111100
0	d	1001	0	l	111001	0	u	11110101	0	’	11111101
0	e	1010	0	m	111010	0	v	11110110	0	!	11111110
0	f	1011	0	n	111011	0	w	11110111	0	?	11111111

On lit alors `q` et on écrit son code `11110001`. le nombre d'occurence de `q` passe de 0 à 1, ce qui donne

1	p	000	0	f	11000	0	n	11110000	0	x	11111000
1	q	001	0	g	11001	0	o	11110001	0	y	11111001
0	␣	010	0	h	11010	0	r	11110010	0	z	11111010
0	a	011	0	i	11011	0	s	11110011	0	.	11111011
0	b	1000	0	j	111000	0	t	11110100	0	,	11111100
0	c	1001	0	k	111001	0	u	11110101	0	’	11111101
0	d	1010	0	l	111010	0	v	11110110	0	!	11111110
0	e	1011	0	m	111011	0	w	11110111	0	?	11111111

A la lecture du `q` suivant, on écrit donc `001` et celui-ci passe en tête :

2	q	000	0	f	11000	0	n	11110000	0	x	11111000
1	p	001	0	g	11001	0	o	11110001	0	y	11111001
0	␣	010	0	h	11010	0	r	11110010	0	z	11111010
0	a	011	0	i	11011	0	s	11110011	0	.	11111011
0	b	1000	0	j	111000	0	t	11110100	0	,	11111100
0	c	1001	0	k	111001	0	u	11110101	0	’	11111101
0	d	1010	0	l	111010	0	v	11110110	0	!	11111110
0	e	1011	0	m	111011	0	w	11110111	0	?	11111111

Q 3) Finir l'exécution de l'algorithme. Il n'est pas demandé d'écrire toutes les étapes.

Q 4) Quel est le taux de compression sur le texte `yyyy...yyyzzzz...zzzz` où il y a n fois le caractère `y` et n fois le caractère `z` ? On donnera la réponse en fonction de n.

Q 5) Expliquer comment (et pourquoi) on peut décompresser.

Q 6) Expliquer en quoi cet algorithme peut être considéré comme un algorithme de type "Huffman adaptatif".

Q 7) Donner un exemple sur lequel l'algorithme n'est pas du tout efficace. On se contentera de donner une idée et on ne cherchera pas à calculer le taux de compression.

3 TP

Q 1) Ecrire l'algorithme de compression LZ78. Votre programme attendra un texte sur l'entrée standard, et renverra le résultat sur la sortie standard. Par exemple, sur l'entrée

```
aabaaabc
```

votre programme pourra répondre

```
(0,a) (1,b) (1,a) (2,c)
```

Le programme C que vous allez écrire prendra le texte sur l'entrée standard. Pour cela, vous pouvez utiliser la commande `getchar` qui lit caractère après caractère de l'entrée standard.

Ainsi l'exemple suivant lit toute l'entrée standard puis la met dans le tableau A.

```
#include <stdio.h>
int main(){
  unsigned char A[50000];
  int c,i;
  i = 0;
    while ((c=getchar()) != EOF)
    {
    A[i] = c;
    i++;
    }
  }
```

On rappelle ici l'algorithme vu en cours, qui utilise un tableau $T[][]$. On rappelle qu'initialement $N = 0$ et que $T[0][c] = -1$ pour tout c.

- 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
  - mettre i à 0.


Attention, il faut éviter de garder en mémoire le texte d'entrée.

Q 2) Ecrire, au choix, soit l'algorithme de décompression LZ78, soit l'algorithme de compression LZ77

A Algorithme LZ78

1	2	3	4	5	6	7	8	9	10
(0, Q)	(0, u)	(0, e)	(0, l)	(0, q)	(2, e)	(0, s)	(0, ␣)	(0, j)	(0, o)
11	12	13	14	15	16	17	18	19	20
(2, r)	(7, ␣)	(0, a)	(0, v)	(13, n)	(0, t)	(0, ,)	(8, a)	(0, i)	(0, n)
21	22	23	24	25	26	27	28	29	30
(7, i)	(8, q)	(6, ␣)	(0, c)	(3, l)	(13, ␣)	(13, v)	(13, i)	(16, ␣)	(0, é)
31	32	33	34	35	36	37	38	39	40
(16, é)	(8, c)	(10, n)	(14, e)	(20, u)	(17, ␣)	(4, e)	(8, j)	(3, u)	(20, e)
41	42	43	44	45	46	47	48	49	50
(10, f)	(0, f)	(19, c)	(19, e)	(0, r)	(8, v)	(19, n)	(29, t)	(45, o)	(2, v)
51	52	53	54	55	56	57	58	59	60
(3, r)	(8, E)	(4, i)	(0, z)	(2, n)	(0, d)	(10, ␣)	(3, t)	(8, l)	(2, i)
61	62	63	64	65	66	67	68	69	70
(8, d)	(3, m)	(15, d)	(26, l)	(26, m)	(28, n)	(61, e)	(8, s)	(26, f)	(19, l)
71	72	73	74	75	76	77	78	79	80
(37, .)	(0, I)	(4, ␣)	(40, ␣)	(4, u)	(19, ␣)	(24, a)	(24, h)	(26, p)	(13, s)
81	82	83	84	85	86	87	88	89	90
(22, u)	(0, ')	(0, H)	(13, d)	(9, i)	(74, s)	(51, a)	(19, t)	(8, h)	(39, r)
91	92	93	94	95	96	97	98	99	100
(39, s)	(3, ␣)	(5, u)	(82, i)	(73, v)	(10, u)	(4, û)	(29, b)	(44, n)	(13, p)
101	102	103	104	105	106	107	108	109	110
(0, p)	(49, u)	(34, r)	(68, a)	(61, é)	(0, m)	(13, r)	(78, e)	(0, .)	(8, D)
111	112	113	114	115	116	117	118	119	120
(82, a)	(70, l)	(90, s)	(36, i)	(73, n)	(92, s)	(111, g)	(19, s)	(7, a)	(88, ␣)
121	122	123	124	125	126	127	128	129	130
(93, e)	(61, ')	(10, b)	(16, e)	(20, i)	(45, s)	(33, ␣)	(80, s)	(3, n)	(16, i)
131	132	133	134	135	136	137	138	139	140
(106, e)	(20, t)	(109, ␣)	(0, L)	(92, m)	(107, i)	(13, g)	(92, n)	(116, e)	(45, a)
141	142	143	144	145	146	147	148	149	150
(120, c)	(30, l)	(30, b)	(45, é)	(81, ')	(13, u)	(8, r)	(58, o)	(11, ␣)	(56, ')
151	152	153	154	155	156	157	158	159	160
(83, e)	(20, r)	(0, y)	(150, A)	(4, b)	(107, e)	(16, .)	(8, S)	(127, a)	(0, b)
161	162	163	164	165	166	167	168	169	170
(7, e)	(20, c)	(3, ,)	(8, i)	(73, l)	(82, e)	(7, p)	(30, r)	(28, t)	(61, u)
171	172	173	174	175	176	177	178	179	180
(8, m)	(10, i)	(20, s)	(36, n)	(92, p)	(96, v)	(169, ␣)	(101, l)	(2, s)	(0, ê)
181	182	183	184	185	186	187	188	189	190
(16, r)	(92, d)	(92, l)	(33, g)	(23, d)	(11, é)	(3, .)	(134, e)	(8, b)	(15, q)
191	192	193	194	195	196	197	198	199	200
(60, e)	(45, ␣)	(24, o)	(20, n)	(28, s)	(119, i)	(29, l)	(26, s)	(88, u)	(13, t)
201	202	203	204	205	206	207	208	209	210
(19, o)	(20, ␣)	(56, u)	(38, e)	(55, e)	(8, o)	(42, f)	(43, i)	(51, ,)	(59, ')
211	212	213	214	215	216	217	218	219	220
(30, t)	(200, ␣)	(56, e)	(119, ␣)	(42, o)	(45, t)	(205, ,)	(59, a)	(32, o)	(173, i)
221	222	223	224	225	226	227	228	229	230
(56, é)	(140, t)	(201, n)	(61, o)	(132, ␣)	(9, o)	(60, s)	(196, t)	(104, ␣)	(42, a)
231	232	233	234	235	236	237	238	239	240
(106, i)	(4, l)	(92, e)	(202, F)	(140, n)	(24, e)				