Otras técnicas

Fernando Martínez
fernando.martinez@upc.edu

Departament de Matemàtiques • Universitat Politècnica de Catalunya

15 de marzo de 2022

Capítulo 6 de Introduction to Data Compression **Sayood, Khalid** Morgan Kaufmann, 2012, 4th ed.

Códigos de Rice

Se usan para codificar enteros de longitud variable.

Dependen de un parámetro k. Sea $m = 2^k$, para codificar un entero x:

1. Sea $q = x//m$, (cociente), escribir q 1's.
2. Escribir un 0.
3. Escribir $x \mod m$ (resto) con exactamente k bits

$k = 2,\ m = 2^2$ y $k = 4,\ m = 2^4$

x	$k = 2,\ m = 2^2$	$k = 4,\ m = 2^4$
0	0 00	0 0000
1	0 01	0 0001
2	0 10	0 0010
3	0 11	0 0011
4	10 00	0 0011
5	10 01	0 0101
6	10 10	0 0110
7	10 11	0 0111
8	110 00	0 1000
...	...	...
14	1110 10	0 1110
15	1110 11	0 1111
16	11110 00	10 0000
17	11110 01	10 0001

Run Length Encoding (RLE)

Se codifica: `[Símbolo, # de símbolos]`

En el caso de imágenes binarias, no se envía en símbolo y se empieza por enviar blanco (si el primer píxel es negro tenemos un 0 inicial).

Se acostumbra a codificar las longitudes con un Huffman o con una codificación aritmética.

Move to Front (MtF)

Produce repeticiones consecutivas de posiciones y aunque tome valores enteros en un rango `[1, len(alfabeto)]`, estás están sesgadas hacia las más pequeñas.

Ejemplo: mi mama me mima mucho

|mi mama me mima mucho [_, a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, ...] 13
m|i mama me mima mucho [m, _, a, b, c, d, e, f, g, h, i, j, k, l, n, o, p, ...] 10
mi| mama me mima mucho [i, m, _, a, b, c, d, e, f, g, h, j, k, l, n, o, p, ...] 2
mi |mama me mima mucho [_, i, m, a, b, c, d, e, f, g, h, j, k, l, n, o, p, ...] 2
mi m|ama me mima mucho [m, _, i, a, b, c, d, e, f, g, h, j, k, l, n, o, p, ...] 3
mi ma|ma me mima mucho [a, m, _, i, b, c, d, e, f, g, h, j, k, l, n, o, p, ...] 1
mi mam|a me mima mucho [m, a, _, i, b, c, d, e, f, g, h, j, k, l, n, o, p, ...] 1
mi mama| me mima mucho [a, m, _, i, b, c, d, e, f, g, h, j, k, l, n, o, p, ...] 2
...
mi mama me mima much|o [h, c, u, m, _, a, i, e, b, d, f, g, j, k, l, n, o, p, q, r, s, t, v, w, x, y, z] 16

Código: [13, 10, 2, 2, 3, 1, 1, 2, 2, 7, 2, 2, 4, 1, 4, 3, 2, 21, 7, 11, 16]

Burrows-Wheeler (BW)

Aprovecha el contexto para agrupar caracteres iguales.

Ejemplo: cadacaacapasta

Permutaciones cíclicas:		Ordenación lexicográfica:
cadacaacapasta	0	aacapastacadac
acadacaacapast	1	acaacapastacad
tacadacaacapas	2	acadacaacapast
stacadacaacapa	3	acapastacadaca
astacadacaacap	4	adacaacapastac
pastacadacaaca	5	apastacadacaac
apastacadacaac	6	astacadacaacap
capastacadacaa	7	caacapastacada
acapastacadaca	8	cadacaacapasta
aacapastacadac	9	capastacadacaa
caacapastacada	10	dacaacapastaca
acaacapastacad	11	pastacadacaaca
dacaacapastaca	12	stacadacaacapa
adacaacapastac	13	tacadacaacapas

El resultado de la transformación es la última columna de la ordenación lexicográfica y la posición que ocupa el mensaje original en dicha ordenación lexicográfica:
`cdtaccpaaaaaas, 8`

Burrows-Wheeler (BW)

Código: `cdtaccpaaaaaas, 8`

Ordenando alfabéticamente, sabemos las primeras letras, además de las últimas de todas las permutaciones.

La última letra del mensaje la sabemos porque nos dicen la posición del mensaje original en la lista de permutaciones ordenadas, en este caso `a`.

Para saber la penúltima letra nos fijamos en que si eliminamos la última letra y la colocamos al inicio ahora la penúltima letra pasará a ocupar la última posición y será una de las que hay en la última posición de las palabras que empiezan por `a`. ¿Cuál? pues como están ordenadas alfabéticamente si la `a` ocupa la $3^{\underline{a}}$ posición entre las `a` al hacer el desplazamiento cíclico seguirá ocupando la $3^{\underline{a}}$ posición entre las `a`, por lo tanto la penúltima letra será `t`.

Para saber la previa a la `t`, hacemos lo mismo, si eliminamos la `t` y la colocamos al inicio ahora la antepenúltima letra pasará a ocupar la última posición y será `s`.

La anterior a la `s` será la séptima `a`.

La anterior a la (séptima) `a` será la `p`.

La anterior a la `p` será la sexta `a`...

Código: cdtaccpaaaaaas, 8

```
a...........c  0
a...........d  1
a...........t  2
a...........a  3
a...........c  4
a...........c  5
a...........p  6
c...........a  7
c...........a <--------
c...........a  9
d...........a 10
p...........a 11
s...........a 12
t...........s 13
Mensaje reconstruido (aunque también sabemos la primera letra): .....a

Iter:  1  Iter:  2  Iter:  3  Iter:  4   ...     Iter: 13
a......c  a......c  a......c  a......c           a......c
a......d  a......d  a......d  a......d           a......d
a------>t  a......t  a......t  a......t           a......t
a......a  a......a  a......a  a......a           a......a
a......c  a......c  a......c  a......c           a------>c
a......c  a......c  a......c  a......c           a......c
a......p  a......p  a......p  a------>p           a......p
c......a  c......a  c......a  c......a           c......a
c......a  c......a  c......a  c......a           c......a
c......a  c......a  c......a  c......a           c......a
d......a  d......a  d......a  d......a           d......a
p......a  p......a  p......a  p......a           p......a
s......a  s......a  s------>a  s......a           s......a
t......s  t------>s  t......s  t......s           t......s
Mensaje:
   ...ta    ...sta   ...asta  ...pasta     cadacaacapasta
```

Burrows-Wheeler (BW)

Observaciones:

- También podemos recuperar el texto original reconstruyendo desde el inicio hacia el final.
- A la hora de codificar no es necesario tener en memoria todas las permutaciones, ya que estas son cíclicas, se puede ordenar alfabéticamente directamente.
- `iBWT` se puede implementar con coste lineal.

Prediction with Partial Match (PPM)

Aprovecha el contexto para asignar probabilidades.

La idea es que el algoritmo de codificación al leer un nuevo símbolo `s` se fije en el contexto `C` de orden N anterior (N símbolos anteriores) y, basándose en los datos leídos anteriormente, asigne una probabilidad p a que `s` siga a `C`. Con estas probabilidades se invoca a un Huffman o a una codificación aritmética.

Prediction with Partial Match (PPM)

Ejemplo

Supongamos que el contexto de orden 3 es la cadena `QUE` (los últimos 3 caracteres leídos) que ha aparecido con anterioridad 27 veces y que se ha visto seguido:

- 11 veces de la letra ␣ `(espacio)`
- 9 veces de la letra `S`
- 6 veces de la letra `N`
- 1 veces de la letra `M`

Si la siguiente letra fuera una ␣ se envía a codificar la letra ␣ (Huffman o aritmética) con la distribución de probabilidades ␣ `11/27, S 9/27, N 6/27, M 1/27` y se incrementan los contextos de:

- `QUE`+␣ a 12
- `UE`+␣ en +1
- `E`+␣ en +1
- ␣ en +1

Prediction with Partial Match (PPM)

Ejemplo (cont.)

Si la siguiente letra hubiera sido `A`, como no parece en el contexto `QUE` se cambia el contexto a `UE`. Si en este contexto tenemos `UE`

- +`A` 26
- +␣ 11 (como mínimo)
- ...
-

con un total de 54 apariciones, se envía a codificar la letra `A` (Huffman o aritmética) con la distribución de probabilidades `A 26/54,` ␣ `11/54,...` y se incrementan los contextos

- `QUE`+`A` a 1
- `UE`+`A` a 27
- `E`+`A` en +1
- `A` en +1