Sesión 4: Outras técnicas

- Algunhas técnicas básicas: MTF,RLE,…
- Compresión baseada en diccionario: Ziv-Lempel
- RePair/BPE
- Prediction by Partial Matching (PPM)
  - Modelado orden K.
  - Codificación aritmética (estatística)
- BWT

Métodos Básicos: Run-Length Encoding (RLE)

- A idea é que se o elemento ***d*** aparece ***n*** veces consecutivas na cadea de entrada, reemplazar as *n* ocorrencias co par ***nd***.
- Só vale a pena cando unha letra aparece repetida tres ou máis veces.

2. all is too well
2. a@2l is t@2o we@2l

Métodos Básicos: Run-Length Encoding (RLE)

- Problemas do RLE (orixinal):
  - No texto non hai moitas repeticións.
  - O carácter @ pode aparecer no texto.
  - O número de repeticións é un 1 byte, polo que está limitado a 255 carácteres.

- **Aplicacións**:
  - Usado primeiros modems (MNP class 5) combinado con outras técnicas.
  - Bzip2 (combinado con BWT, Move-to-front e Huffman).
  - Compresión de bitmaps (índices). 1-runs, 0-runs

    1000100000111111110000000100001

    1,0@3,1,0@5,1@8,0@7,1,0@4,1
  - Structuras de datos compactas: CSA-psi

    +6 +1 +1 +1 +1 +1 +1 +6 +90 +1 +1 +3 -5 ...

Métodos Básicos: Relative encoding/*gap encoding*

- Útil cando se ten que comprimir unha cadea de números que non difiren en moito.
  - Redúcese a magnitude do número a representar !!
- Tamén cando a cadea son strings de carácteres que non difiren en moito.

- Exemplos:
  - Lecturas de temperaturas.
    - 20, 22, 19, 21…
    - 20, 2, -1, 1 …
  - Listas invertidas:

Lista orixinal	4	10	15	25	29	40	46	54	57	70	79	81
	1	2	3	4	5	6	7	8	9	10	11	12
Diferencias	4	+6	+5	+10	+4	+11	+6	+8	+3	+13	+9	+2

Métodos Básicos: Move-to-front (MTF)

- A idea básica deste método é manter o alfabeto Σ de símbolos como unha lista onde os símbolos máis frecuentes están máis preto do comezo.
- Un símbolo Σ é codificado como o número de símbolos que o preceden da lista.
  - Codifícase a posición do símbolo dentro do Σ (no instante actual).
- Se Σ={t,h,e,s,…} e o próximo símbolo a comprimir é *e*, será codificado como *2*.
- O normal é que logo *e* sexa movido á cabeza da lista. No exemplo Σ={*e*,t,h, s,…}
- O que se espera é que o carácter *e* teña máis probabilidade de ocurrir que os outros (no momento actual)

Métodos Básicos: Move-to-front (compresión)

- En n iteracións (no exemplo desde i=0 até 10) calcula R(secuencia de offsets emitidos)

T = {pssmipissii} Σ = {i,m,p,s} σ=4

Y = Σ = imps
R = ()

i=0	T[0]='p'	→	R[0]=2	Y=pims
i=1	T[1]='s'	→	R[1]=3	Y=spim
i=2	T[2]='s'	→	R[2]=0	Y=spim
i=3	T[3]='m'	→	R[2]=3	Y=mspi

…

R = (2 3 0 3 3 3 1 3 0 1 0)

Métodos Básicos: Move-to-front (descompresión)

- En n iteracións (no exemplo desde i=0 ata 10) calcula T a partires de R

Y = Σ = imps

R = (2 3 0 3 3 3 1 3 0 1 0)

i=0	R[0]=2	→	T[0]='p'	Y=pims
i=1	R[1]=3	→	T[1]='s'	Y=spim
i=2	R[2]=0	→	T[2]='s'	Y=spim
i=3	R[2]=3	→	T[3]='m'	Y=mspi

…

T = {pssmipissii}

Métodos Básicos: Move-to-front (MTF)

- Normalmente combínase con outras técnicas
- Exemplo:
  - no bzip2 con BWT, RLE e Huffman.

Técnicas baseadas en diccionario

- Normalmente:
  - asocian códigos de lonxitude fixa, a símbolos de lonxitude variable (substrings do texto)
  - Canto máis longo sexa o substring substituido → mellor compresión

- Representante máis coñecido: Familia Lempel-Ziv
  - LZ77 (1977): gzip, pkzip, arj …
  - LZ78 (1978)
    - LZW (1984): Compress, imaxes GIF
  - …

LZW

exemplo

input	next character	output	Dictionary
			entry_0 = "a" entry_1 = "b" entry_2 = "c"
a	b	0	entry_3 = "ab"
b	b	1	entry_4 = "bb"
b	a	1	entry_5 = "ba"
ab	c	3	entry_6 = "abc"
c	a	2	entry_7 = "ca"
ab	b	3	entry_8 = "abb"
bb	b	4	entry_9 = "bbb"
b	c	1	entry_{10} = "bc"
c	ε	2	–

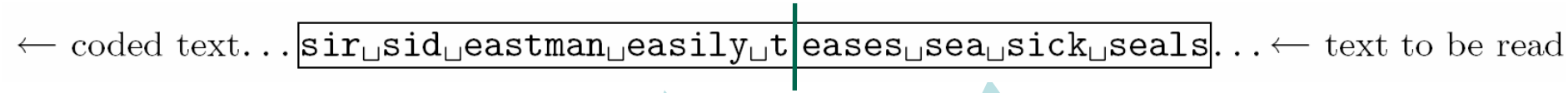
Compression of "abbabcabbbbc", Σ={a, b, c}, using LZW.

- Variante do LZ78
- Partir dun diccionario inicial (que contén os símbolos de Σ)
- Dada unha posición no texto.
  - Leer caracteres $\underline{w=w_0 w_1 w_2}$ … mentres w aparezca no diccionario
  - Cando $\underline{w_0 \ldots w_k w_{k+1}}$ xa non está no diccionario e $\underline{w_0 \ldots w_k}$ si →
    - **output** (i = entryPos($\underline{w_0 \ldots w_k}$)) (NOTA: lonxitude do código dep núm entradas = $\log_2 n$)
    - **insertar entrada** $\underline{w_0 \ldots w_k w_{k+1}}$
    - Continuar a partires de $\underline{w_{k+1}}$ (incluido)

- O diccionario ten tamaño limitado e podería encherse
  - Políticas: LRU, baleirar diccionario e continuar , …

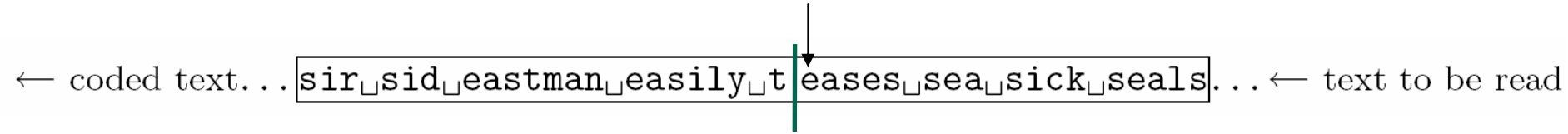
LZ77

- A idea básica é usar parte do texto previamente procesado como dicionario.
- O compresor mantén unha ventá sobre o stream a comprimir, e móvese sobre os símbolos de entrada de dereita a esquerda.
- Polo tanto o método baséase nunha *fiestra deslizante*



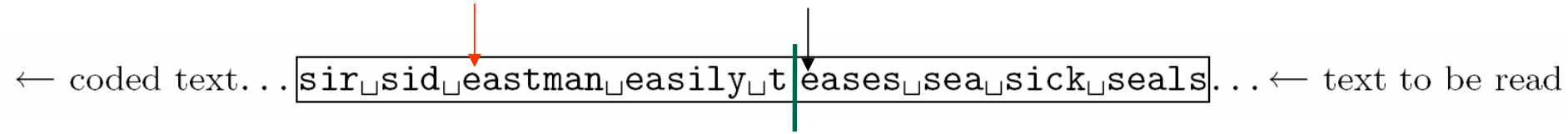
- A fiestra está dividida en dúas partes.
  - A parte esquerda: *buffer de busca qu*e é o dicionario actual.
  - A parte dereita: *buffer de busca cara adiante* e é o texto que se vai a comprimir.

LZ77



- O compresor recorre o *buffer de busca* de dereita a esquerda buscando un emparellamento do primeiro carácter *e* do *buffer busca cara adiante.*
- Atopa o *e* de *easily* que está a 8 bytes do fin do *buffer de busca.*
  - Daquela o compresor trata de ver se hai máis carácteres que concorden despois das dúas *e's.*
  - Atopa que *eas* concorda nos dous lados. Así que a lonxitude do emparellamento é *3.*
- O compresor busca máis atrás no buffer de busca por emparellamentos máis longos.

LZ77



- No noso caso, a palabra *eastman* (offset 16) ten un emparellamento da mesma lonxitude.
- O compresor escolle o emparellamento máis longo, ou se hai varios da mesma lonxitude escolle o último atopado.
- Logo emite un token, no noso caso: ***(16, 3, "e")***
  - Offset (ata o comezo do *match*)
  - Lonxitude (número de caracteres que coincidend)
  - Seguinte carácter (o segundo *e* de *eases)*

LZ77

- Se non se produce ningún emparellamento emítese un token (0,0,"x"), esta é a razón da necesidade do terceiro compoñente.

	sir␣sid␣eastman␣	⇒	(0,0,"s")
s	ir␣sid␣eastman␣e	⇒	(0,0,"i")
si	r␣sid␣eastman␣ea	⇒	(0,0,"r")
sir	␣sid␣eastman␣eas	⇒	(0,0,"␣")
sir␣	sid␣eastman␣easi	⇒	(4,2,"d")

- O seguinte paso emparella o espazo e codifica a cad "␣e"

sir␣sid	␣eastman␣easily␣	⇒	(4,1,"e")
sir␣sid␣e	astman␣easily␣te	⇒	(0,0,"a")

- Típico: 12bits offset + 4bits lonxitude (2bytes total)

LZ77

- O descompresor é incluso máis sinxelo.
- Ten que manter un buffer de igual tamaño que a fiestra do compresor.
- O descompresor obtén un token, atopa o emparellamento no seu buffer, e escribe na saída o emparellamento e o terceiro elemento (caracter) do token.
- Isto fai o LZ77 **moi rápido** en descompresión.

Deflate (zip e gzip)

- Zip, Http, PNG, PDF.
- Deseñado por Philip Katz para o seu PKZIP.
- O formato Zip e Deflate son públicos.
- Zlib é unha librería libre que implementa Deflate e o formato Zip. Débese a Jean-Loup Gailly e Mark Adler.

  (http://zlib.net/)
- É o corazón da maioría das aplicacións Deflate, incluído o Gzip.
- Deflate é unha variación de LZ77 combinado con Huffman

Deflate (zip e gzip)

- LZ77 emitía tokens de tres compoñentes. Necesítase o terceiro elemento para os casos nos que non existe ningún emparellamento.
- Como no caso do LZW con respecto ao LZ78, Deflate elimina a necesidade de engadir ao token o seguinte carácter.
- Na saída escribe:
  - **Se hai emparellamento**: Escribe polo tanto un par:

    <lonxitude, offset>
  - **Se non hai emparellamento**: escribe o carácter (en lugar de un token).

- Polo tanto hai tres tipos de entidades:
  - Literais (carácteres non emparellados).
  - Offsets (distancias)
  - Lonxitudes

Deflate (zip e gzip)

- Eses tres tipos de compoñentes son comprimidos con Huffman.
- Usa dous dicionarios distintos (**dous códigos Huffman distintos**) :
  - Un para os literais e as lonxitudes.
  - Outro para as distancias/offsets.
- Isto é así porque os literais son carácteres (0-255) e as lonxitudes están limitadas a 258 (mínimo *match* de 3 bytes).
- Mentres que as distancias poden ser enteiros moito maiores ao ser o *buffer* de busca normalmente 32Kbytes.

Deflate (zip e gzip)

- Huffman de lonxitudes e literais

  – Créase unha árbore Huffman con espacio para 288 símbolos:

  – 0–255: literais.

  – 256: fin de bloque – parar de procesar se é o derradeiro bloque, noutro caso comeza a procesar o próximo bloque.

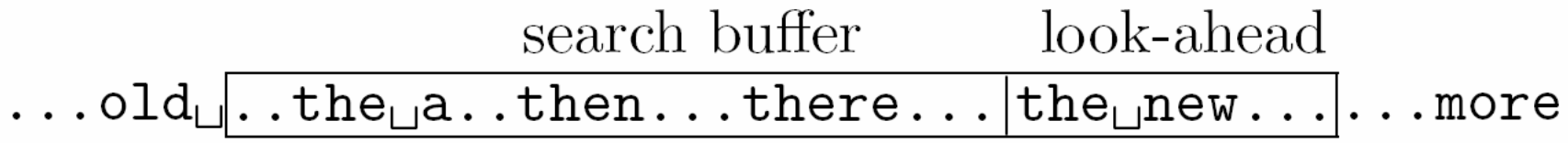
  – 257–285: combinado con bits extra, un emparellamento de lonxitude 3–258 bytes.

  – Ademais... 286-287: non usados, son valores reservados (pero que se manteñen na árbore)

Deflate (zip e gzip)

- Despois dun código de emparellamento sempre ven un código de distancia
- códigos 257+...
  - 0–3: distancias 1–4
  - 4–5: distancias 5–8, 1 extra bit
  - 6–7: distancias 9–16, 2 extra bits
  - 8–9: distancias 17–32, 3 extra bits
  - ...
  - 26–27: distancias 8193–16384, 12 extra bits
  - 28–29: distancias 16385–32768, 13 extra bits

Deflate (zip e gzip) exemplo



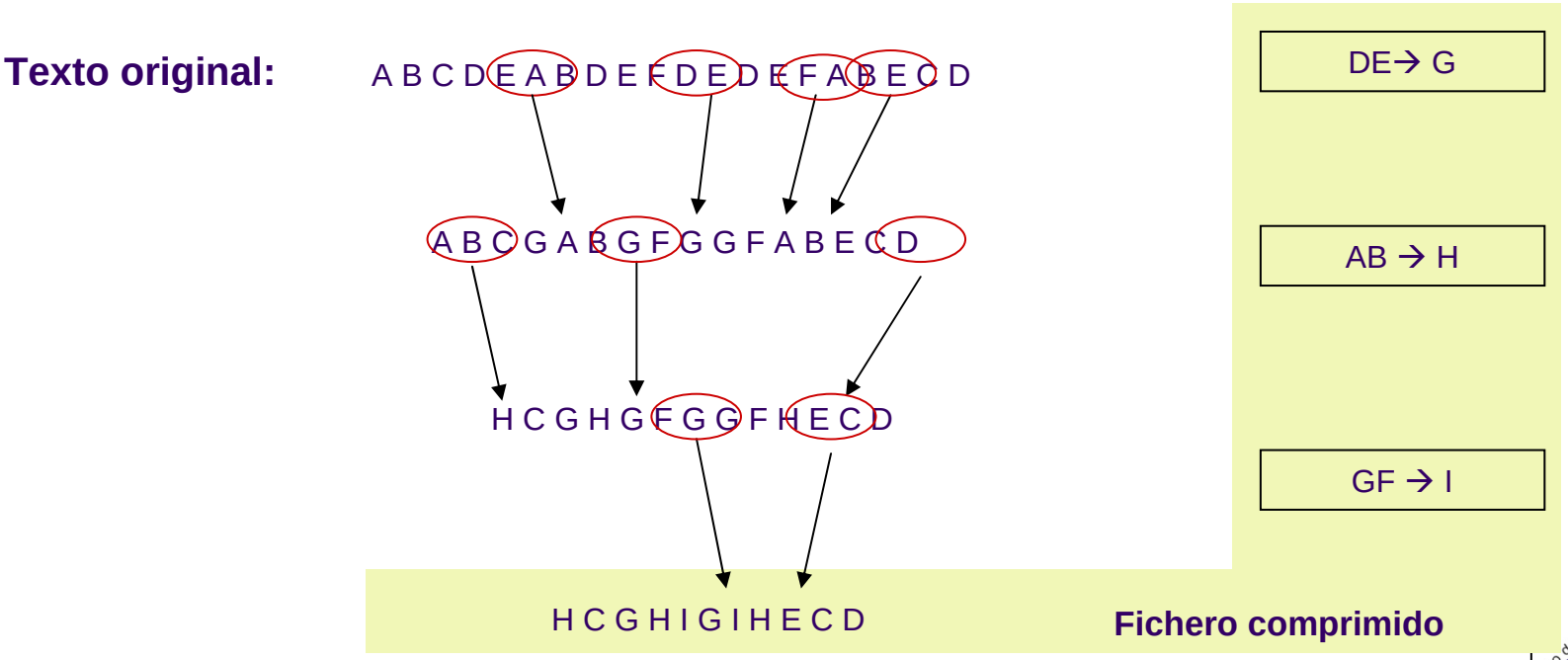
- O emparellamento máis largo é de lonxitude 3. Pero antes de seleccionar este emparellamento, o compresor salva o *t* e comeza unha nova busca onde quere emparellar `he␣new..`
- Se atopa un emparellamento máis longo que 3, emite o *t* como un literal, seguido do emparellamento atopado.
- Este segundo emparellamento é controlado por un parámetro que ten tres valores:
  - Normal (non fai o segundo emparellamento se o primeiro emparellamento inicial supera un parámetro dado).
  - Compresión alta: sempre o fai (o segundo emparellamento)
  - Modo rápido: nunca o fai

Deflate (zip e gzip)

- Deflate comprime o arquivo de entrada por bloques.
- Cada bloque é comprimido segundo un destes tres modos:
  - **Non compression**: utilízase para porcións do arquivo que non son compresibles o xa están comprimidas, ou cando se quere dividir un arquivo en porcións. Un bloque deste tipo comeza cunha cabeceira indicando *modo 1* seguido la lonxitude do bloque (Max. 64 Kbytes).
  - **Compresión con Huffman estático** (máis rápido). Utiliza un código especial para indicar o fin do bloque.
  - **Compresión con Huffman semi-estático** (mellor compresión). Utiliza un código especial para indicar o fin do bloque.

Byte-Pair Encoding (BPE → RePair)

- BPE:: Philip Gage. A new algorithm for data compression. C Users Journal,12(2):23, 1994
- Re-Pair:: Raymond Wan. Browsing and searching compressed documents, PhD thesis, 2003.
- Algoritmo multipasada: Sustituir pares de símbolos por uno nuevo, hasta que no haya un par que se repita o no haya más símbolos sin usar.



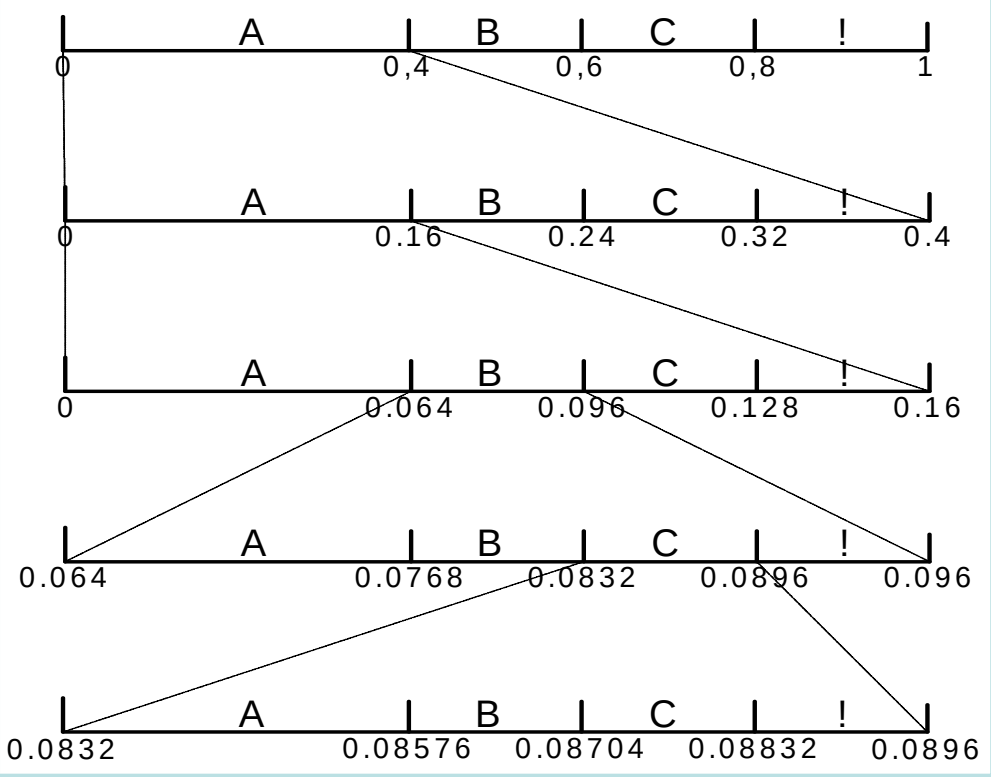
Codificación Aritmética

- A idea orixinal débese a **Peter Elias**.
- É un esquema de compresión estatístico.
- A idea é representar a secuencia orixinal de símbolos como un *ÚNICO* número real no intervalo [0,1).
- Cando a mensaxe é máis grande, o intervalo necesita ser máis pequeno, e a mensaxe necesita máis bits para representalo.

Codificación Aritmética

- A compresión iniciase co intervalo [0,1)
- Cando se procesa un novo símbolo, o **intervalo estreitase** en concordancia coa probabilidade do símbolo procesado.
- O novo intervalo representa o texto procesado ata o momento.

Codificación Aritmética

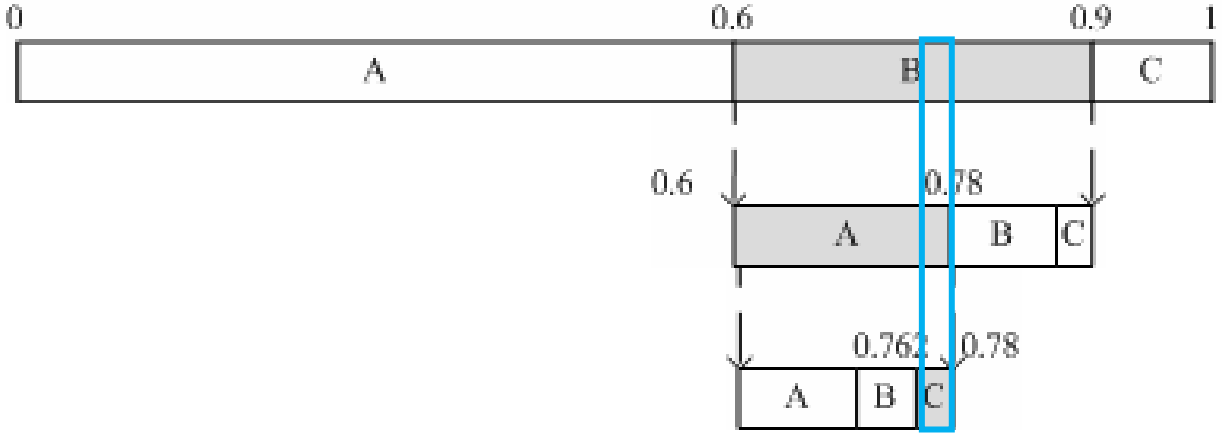


- Sexa o texto AABC!, con frec
  - 'A': 0.4, 'B': 0.2, 'C': 0.2, '!': 0.2
    - [0,0.4) representa A
    - [0.4,0.6) representa B
    - [0.6,0.8) representa C
    - [0.8,1) representa !
  - O primeiro símbolo (A) reduce o intervalo a [0,0.4)
  - O segundo símbolo (A) reduce o intervalo a [0,0.16)
  - O terceiro símbolo (B) reduce o intervalo a [0.064,0.096)
  - C reduce o intervalo a [0.0832,0.0896)
  - ! reduce o intervalo a [0.08832,0.0896)
- Cualquera número nese intervalo representa o texto comprimido.

Codificación Aritmética

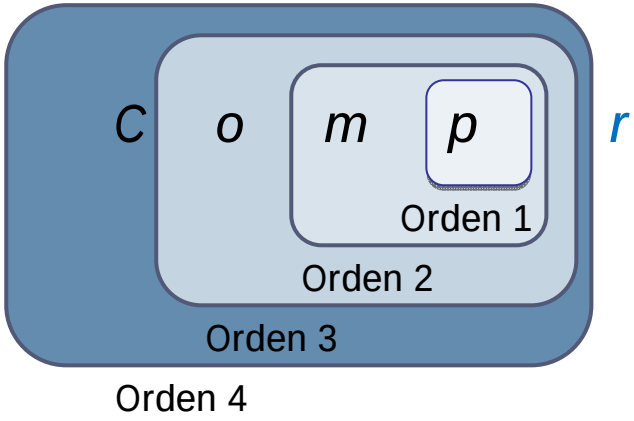
- Codificación.
  - División do rango [0,1) en sub-intervalos proporcionais á probabilidade de cada símbolo.
- Exemplo: codificar *BAC* *(alfabeto = {A,B,C})*

Símb	$Prob_i$
A	0,6
B	0,3
C	0,1



Prediction by Partial Matching (PPM)

- Cleary e Witten 1984.
- Técnica estatística, e dinámica:
  - Símbolos + frecuentes ➔ códigos máis curtos
- Modelador de orden K + codificación aritmética



- Obtén a frecuencia do símbolo actual en función do contexto dos últimos k símbolos que o preceden.
  - ➔ frecuencia máis alta que no texto "completo"
  - ➔dinámico: necesidade de códigos de escape para introducir novos contextos

Prediction by Partial Matching (PPM)

- Exemplo: codificar nun contexto de orden 2

- Texto: …compa**r**ar
  - Contexto de orden 2: mp

contexto “mp”

Símbolos	Frec
a	5
o	3

Codificar *a*

- Texto: …comp**r**esión
  - Contexto de orden 2: mp

Símbolos	Frec
a	5
o	3

Símbolo de Escape (r non precedido por mp previamente)

  - Contexto de orden 1: p

Símbolos	Frec
a	6
o	4
l	2
r	3
u	5

Codificar *r*

Burrows-Wheeler Transform (BWT)

- Algoritmo reversible creado en 1983 (e publicado en 1994) para transformar o texto orixinal (T) noutro máis compresible (L).

- BWT (T) → (L,I)
  Algoritmo para calcular L e I.

- $BWT^{-1}(L,I) \rightarrow T$

  Algoritmo inverso para calcular T a partir de L e I.

Cálculo de L e I

T = {mississippi} de lonxitude n.

M = matriz de permutacións do texto ordenadas lexicograficamente …

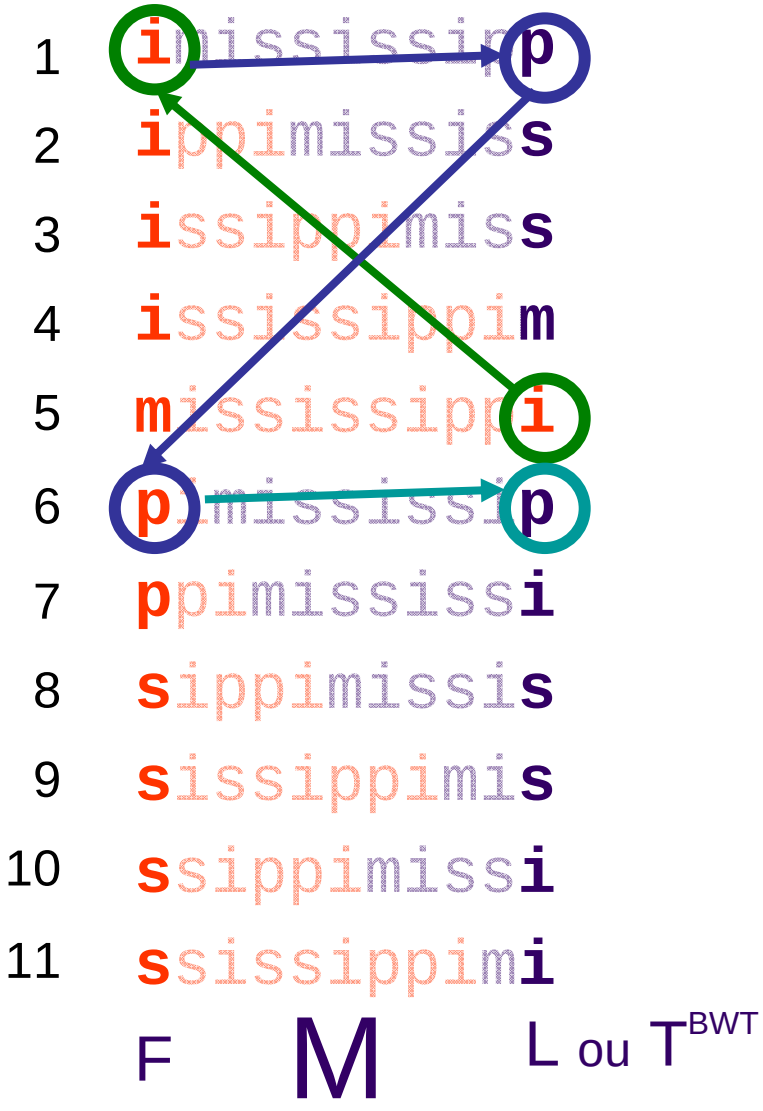
mississippi		imississipp
ississippim		ippimississ
ssissippimi		issippimiss
sissippimis		ississippim
issippimiss	ordenados →	mississippi
ssippimissi		pimississip
sippimissis		ppimississi
ippimississ		sippimissis
ppimississi		sissippimis
pimississip		ssippimissi
imississipp		ssissippimi

- L= {pssmipissii} l=5

	M
1	**i**mississip**p**
2	**i**ppimissis**s**
3	**i**ssippimis**s**
4	**i**ssissippi**m**
5	**m**ississippi
6	**p**imississi**p**
7	**p**pimississi
8	**s**ippimissi**s**
9	**s**sippimissi
10	**s**sippimissi
11	**s**sissippimi

F M L ou T^{BWT}

- L= {pssmipissii} l=5



Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})

	F										L
1	**i**	m	i	s	s	i	s	s	i	p	**p**
2	**i**	p	p	i	m	i	s	s	i	s	**s**
3	**i**	s	s	i	p	p	i	m	i	s	**s**
4	**i**	s	s	i	s	s	i	p	p	i	**m**
5	**m**	i	s	s	i	s	s	i	p	p	**i**
6	**p**	i	m	i	s	s	i	s	s	i	**p**
7	**p**	p	i	m	i	s	s	i	s	s	**i**
8	**s**	i	p	p	i	m	i	s	s	i	**s**
9	**s**	i	s	s	i	p	p	i	m	i	**s**
10	**s**	s	i	p	p	i	m	i	s	s	**i**
11	**s**	s	i	s	s	i	p	p	i	m	**i**

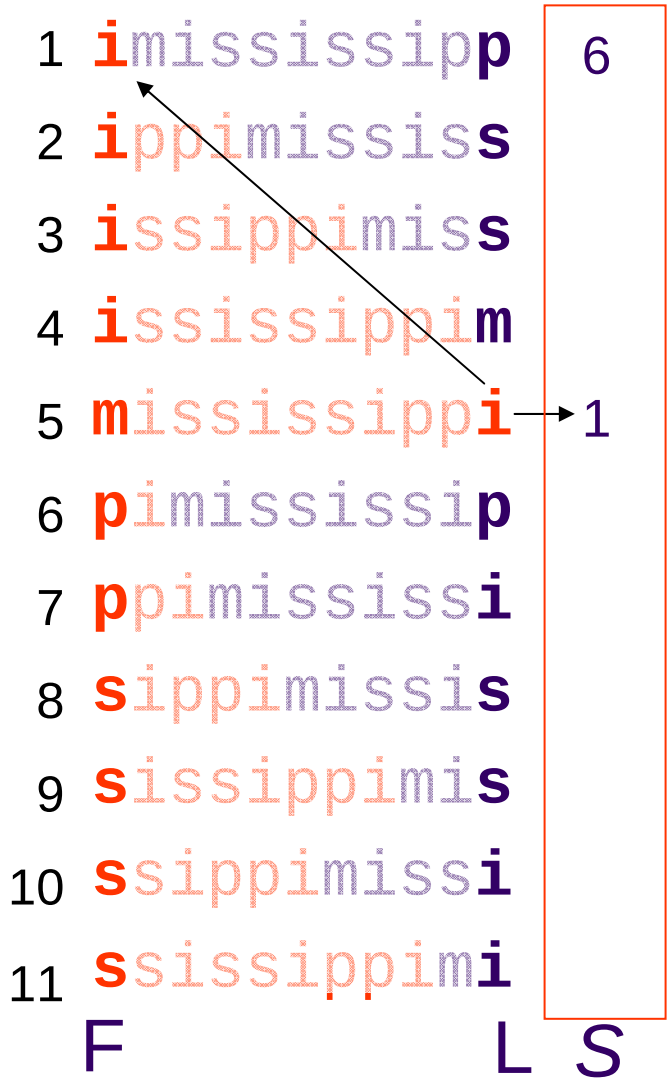
3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º.

3º.

Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})



3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º. Calcúlase *S* : array que contén a posición en F de cada caracter de L.

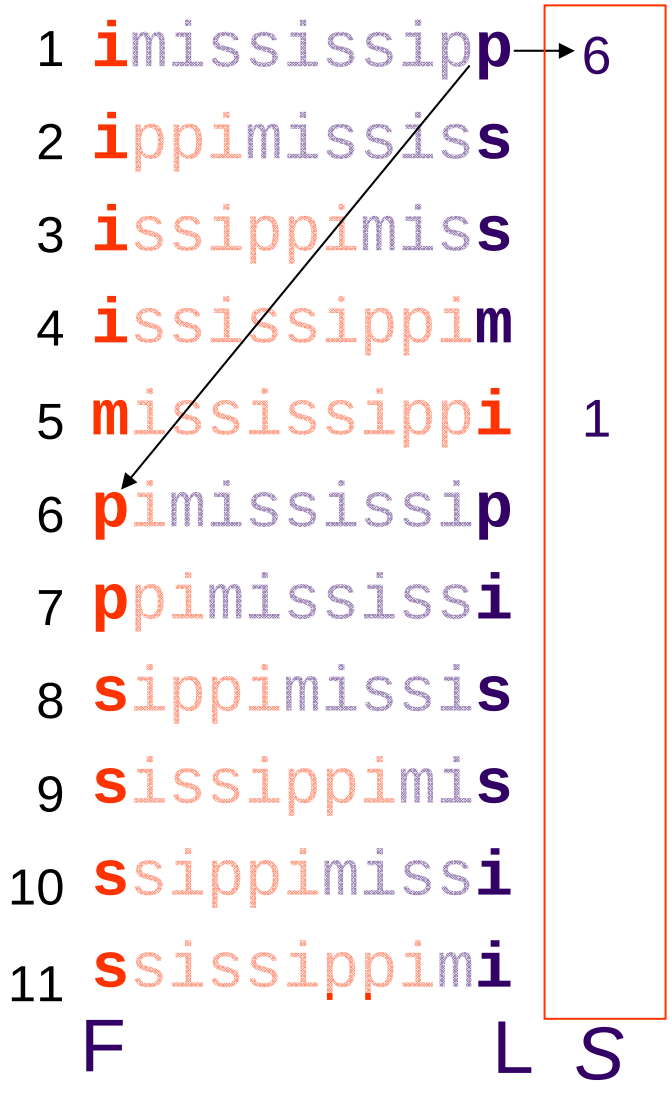
Se L[j] e a ocorrencia número *k* dun caracter *c* en L, entón S[j] = i tal que F[i] é a ocorrencia número *k* en F dese caracter *c*.

F[S[j]] = L[j]

Exemplo: S[1] = **6** porque L[1] é a primeira ocorrencia do caracter "p" en L e F[**6**] é a primeira ocorrencia do carácter "p" en F.

3º.

Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})



3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º. Calcúlase *S* : array que contén a posición en F de cada caracter de L.

Se L[j] e a ocorrencia número *k* dun caracter *c* en L, entón S[j] = i tal que F[i] é a ocorrencia número *k* en F dese caracter *c*.
F[S[j]] = L[j]

3º.

Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})

	F	L	S
1	**i**mississip	**p**	6
2	**i**ppimissis	**s**	8
3	**i**ssippimis	**s**	9
4	**i**ssissippi	**m**	5
5	**m**ississipp	**i**	1
6	**p**imississi	**p**	7
7	**p**pimississ	**i**	2
8	**s**ippimissi	**s**	10
9	**s**sippimiss	**i**	11
10	**s**sippimiss	**i**	3
11	**s**sissippim	**i**	4

3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º. Calcúlase *S* : array que contén a posición en F de cada caracter de L.

Se L[j] e a ocorrencia número *k* dun caracter *c* en L, entón S[j] = i tal que F[i] é a ocorrencia número *k* en F dese caracter *c*.

F[S[j]] = L[j]

3º. En *n* pasos (*n* es el número de caracteres del texto) recuperamos el texto original.

Inicialmente p=I=5, i=0

En cada paso: T[n-i] = L[p]

p = *S* [p]

i = i+1

Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})

	F	LS	T
1	**i**mississip**p**	6	-
2	**i**ppimissis**s**	8	-
3	**i**ssippimis**s**	9	-
4	**i**ssissippi**m**	5	-
5	**m**ississippi**i**	1	-
6	**p**imississi**p**	7	-
7	**p**pimississ**i**	2	-
8	**s**ippimissi**s**	10	-
9	**s**sippimisi**s**	11	-
10	**s**sippimissi**i**	3	-
11	**s**sissippimi**i**	4	i

3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º. Calcúlase *S* : array que contén a posición en F de cada caracter de L.

Se L[j] e a ocorrencia número *k* dun caracter *c* en L, entón S[j] = i tal que F[i] é a ocorrencia número *k* en F dese caracter *c*.
F[S[j]] = L[j]

3º. En *n* pasos (*n* es el número de caracteres del texto) recuperamos el texto original.

Inicialmente p=I=5, i=0

En cada paso: **T[n-i] = L[p]** → T[11]='i'
p = *S* [p] → p= 1
i = i+1 → i=1

Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})

	F	L	S	T
1	imississipp	p	6	-
2	ippimississ	s	8	-
3	issippimiss	s	9	-
4	ississippim	m	5	-
5	mississippi	i	1	-
6	pimississip	p	7	-
7	ppimississi	i	2	-
8	sippimissis	s	10	-
9	sissippimis	s	11	-
10	ssippimissi	i	3	p
11	sissippimi	i	4	i

3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º. Calcúlase *S* : array que contén a posición en F de cada caracter de L.

Se L[j] e a ocorrencia número *k* dun caracter *c* en L, entón S[j] = i tal que F[i] é a ocorrencia número *k* en F dese caracter *c*.

F[S[j]] = L[j]

Exemplo: S[0] = **5** porque L[0] é a primeira ocorrencia do caracter "p" en L e F[**5**] é a primeira ocorrencia do carácter "p" en F.

3º. En *n* pasos (*n* es el número de caracteres del texto) recuperamos el texto original.

Inicialmente p=I=5, i=0

En cada paso: **T[n-i] = L[p]** → T[10]='p'

p = *S* [p] → p= 6

i = i+1 → i=2

Cálculo de Texto orix (T) a partir de L e I (BTW^{-1})

	F	L	S	T
1	**i**mississip**p**	6	m	
2	**i**ppimissis**s**	8	i	
3	**i**ssippimis**s**	9	s	
4	**i**ssissippi**m**	5	s	
5	**m**ississippi**i**	1	i	
6	**p**imississi**p**	7	s	
7	**p**pimississ**i**	2	s	
8	**s**ippimissi**s**	10	i	
9	**s**issippimi**s**	11	p	
10	**s**sippimiss**i**	3	p	
11	**s**sissippim**i**	4	i	

3 pasos:

1º. Calcúlase F ordenando os caracteres de L.

2º. Calcúlase *S* : array que contén a posición en F de cada caracter de L.

Se L[j] e a ocorrencia número *k* dun caracter *c* en L, entón S[j] = i tal que F[i] é a ocorrencia número *k* en F dese caracter *c*.

F[S[j]] = L[j]

Exemplo: S[1] = **6** porque L[1] é a primeira ocorrencia do caracter "p" en L e F[**6**] é a primeira ocorrencia do carácter "p" en F.

3º. En *n* pasos (*n* es el número de caracteres del texto) recuperamos el texto original.

Inicialmente p=I=5, i=0

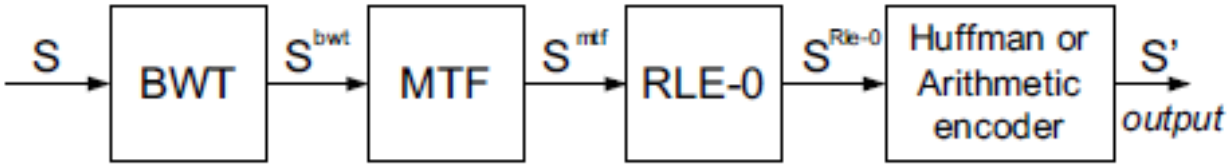
En cada paso: T[n-i] = L[p]

p = *S* [p]

i = i+1

José R. Paramá

Bzip2



1. RLE
2. BWT. Coloca moitos caracteres iguais xuntos.
3. MTF. Cando hai 2 ou máis caracteres iguais xuntos iso se convirte en secuencias de ceros (ou polo menos enteiros pequenos).
4. RLE. Esas secuencias de ceros (ou otros enteiros se comprimen).
5. Huffman.

Algunhas comparacións…

Comparison of compression ratio against adaptive compressors

Corpus	Original size	SCDC	ETDC	*arith*	*gzip -f*	*gzip -b*	*bzip2 -f*	*bzip2 -b*	ppmdi	
									-9	**def**
CALGARY	2,131,045	43.64	44.43	34.68	43.53	36.840	32.83	28.92	25,62%	26,39%
FT91	14,749,355	33.50	34.15	28.33	42.57	36.33	32.31	27.06	23,44%	25,30%
CR	51,085,545	30.80	31.45	26.30	39.51	33.18	29.51	24.14	20,72%	22,42%
FT92	175,449,235	31.68	32.28	29.82	42.59	36.38	32.37	27.09	23,42%	25,34%
ZIFF	185,220,215	32.92	33.62	26.36	39.66	32.98	29.64	25.11	21,77%	23,04%
FT93	197,586,294	31.68	32.36	27.89	40.20	34.12	30.62	25.32	21,68%	23,55%
FT94	203,783,923	31.54	32.23	27.86	40.24	34.12	30.54	25.27	21,68%	23,55%
AP	250,714,271	32.14	32.69	28.00	43.65	37.23	33.26	27.22	23,33%	25,62%
ALL_FT	591,568,807	31.45	32.13	27.85	40.99	34.85	31.15	25.87	22,24%	24,12%
ALL	1,080,719,883	32.72	33.36	27.98	41.31	35.00	31.30	25.98	**22,34%**	24,21%

Algunhas comparacións…

Compression time comparison (in seconds)

Corpus	PHC	SCDC	ETDC	THC	*arith*	*gzip -f*	*gzip -b*	*bzip2 -f*	*bzip2 -b*
CALGARY	0.415	0.405	0.393	0.415	1.030	0.360	1.095	2.180	2.660
FT91	2.500	2.493	2.482	2.503	6.347	2.720	7.065	14.380	18.200
CR	7.990	7.956	7.988	8.040	21.930	8.875	25.155	48.210	65.170
FT92	29.243	29.339	29.230	29.436	80.390	34.465	84.955	166.310	221.460
ZIFF	30.354	30.620	30.368	30.725	82.720	33.550	82.470	174.670	233.250
FT93	32.915	33.031	32.783	33.203	91.057	36.805	93.135	181.720	237.750
FT94	33.874	33.717	33.763	33.700	93.467	37.500	96.115	185.107	255.220
AP	42.641	42.676	42.357	42.663	116.983	50.330	124.775	231.785	310.620
ALL_FT	99.889	100.570	100.469	101.471	274.310	117.255	293.565	558.530	718.250
ALL	191.396	191.809	191.763	192.353	509.710	188.310	532.645	996.530	1,342.430

				ppmdi	
CORPUS	ETDC	gzip	bzip2	-9	def
CALGARY	0,16	0,31	0,77	1,34	1,24
CR	3,11	7,48	17,03	27,89	26,75
ZIFF	11,89	26,29	63,02	105,43	98,81
FT93	12,70	27,48	67,60	112,81	106,69
AP	16,45	39,16	83,83	152,73	145,81
ALL FT	39,13	84,70	202,80	345,81	337,24
ALL	75,03	**157,91**	**336,81**	**636,57**	599,15

Algunhas comparacións…

Decompression time comparison (in seconds)

Corpus	PHC	SCDC	ETDC	THC	*arith*	*gzip -f*	*gzip -b*	*bzip2 -f*	*bzip2 -b*
CALGARY	0.088	0.097	0.085	0.092	0.973	0.090	0.110	0.775	0.830
FT91	0.577	0.603	0.570	0.575	5.527	0.900	0.825	4.655	5.890
CR	1.903	1.971	1.926	1.968	18.053	3.010	2.425	15.910	19.890
FT92	7.773	7.592	7.561	7.801	65.680	8.735	7.390	57.815	71.050
ZIFF	8.263	7.988	7.953	8.081	67.120	9.070	8.020	58.790	72.340
FT93	8.406	8.437	8.694	8.657	71.233	10.040	9.345	62.565	77.860
FT94	8.636	8.690	8.463	8.825	75.925	10.845	10.020	62.795	80.370
AP	11.040	11.404	11.233	11.637	88.823	15.990	13.200	81.875	103.010
ALL_FT	24.798	25.118	24.500	26.280	214.180	36.295	30.430	189.905	235.370
ALL	45.699	46.698	46.352	47.156	394.067	62.485	56.510	328.240	432.390

				ppmdi	
CORPUS	ETDC	gzip	bzip2	-9	def
CALGARY	0,02	0,04	0,33	1,33	1,31
CR	0,62	0,95	7,06	27,78	28,15
ZIFF	2,25	3,44	25,61	104,53	102,85
FT93	2,40	3,72	28,00	112,42	112,38
FT94	2,55	3,78	28,77	115,89	116,47
AP	3,33	5,12	37,66	153,19	153,13
ALL FT	7,59	**11,47**	84,53	**345,05**	344,01
ALL	14,36	20,70	155,88	630,27	631,09

Referencias…

- D. Salomon. *Data Compression.* Springer, 2007.
- Bell, Cleary e Witten. *Text Compression,* Prentice Hall, 1990.

A. Moffat and A. Turpin. *Compression and Coding Algorithms.* Kluwer_Academic Publ., March 2002.

FM-Index

Ferragina & Manzini

FM-Index

Full-text index that occupies Minute space

T : Texto más el carácter terminador $; longitud n

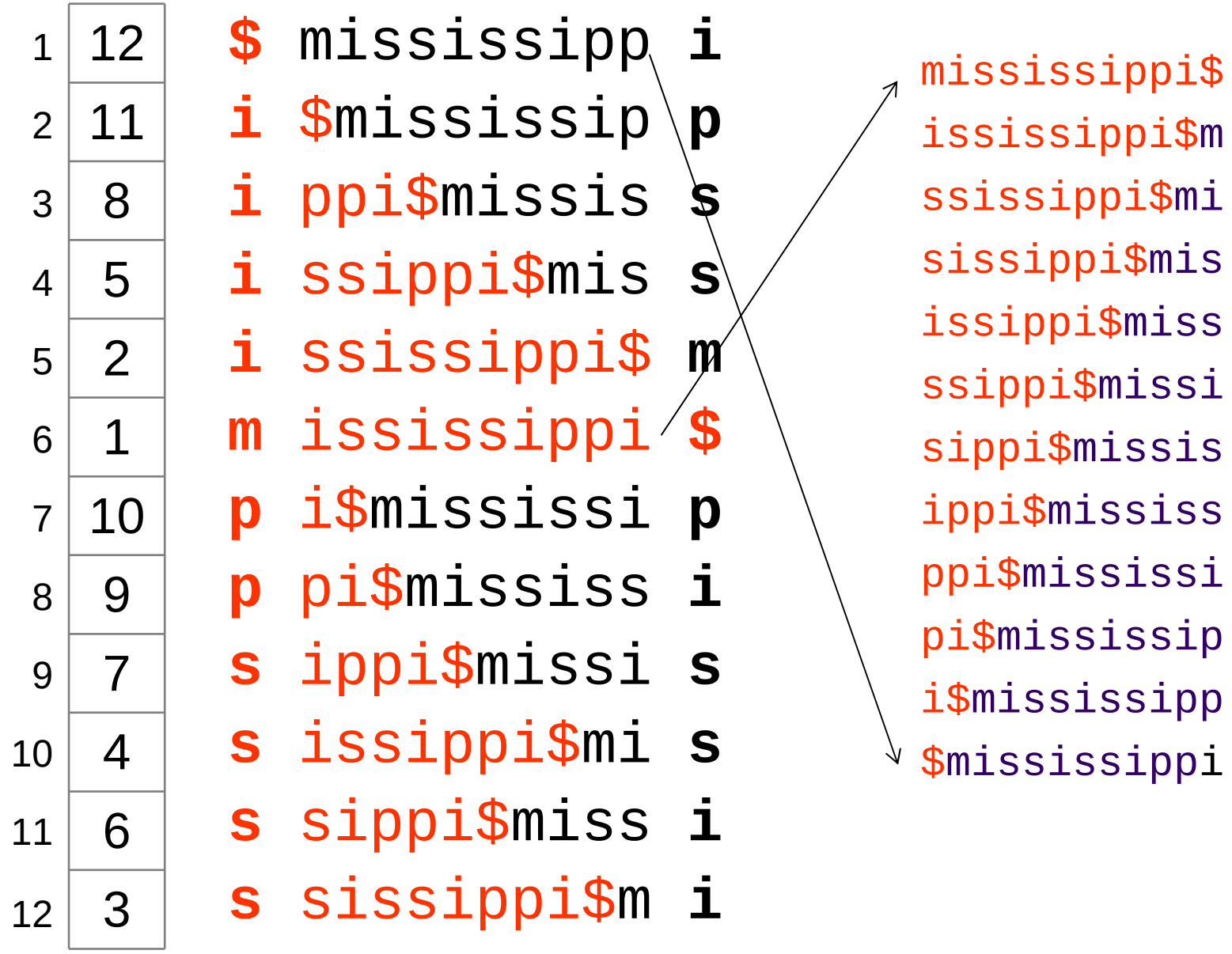
Σ : Alfabeto de longitud σ

M : Matriz con todas las posibles permutaciones de T ordenadas lexicográficamente

L : última columna de M

- Permite recuperar el texto original (extract)
- Permite buscar el número de ocurrencias de un patrón en el texto (count)
- Permite conocer la posición de un sufijo en el texto (locate)

➔ En definitiva lo convierte en lo que comúnmente se conoce como un “autoíndice”.


1 12 $ mississipp i
2 11 i $mississip p
3 8 i ppi$missis s
4 5 i ssippi$mis s
5 2 i ssissippi$ m
6 1 m ississippi $
7 10 p i$mississi p
8 9 p pi$mississ i
9 7 s ippi$missi s
10 4 s issippi$mi s
11 6 s sippi$miss i
12 3 s sissippi$m i
mississippi$
ississippi$m
ssissippi$mi
sissippi$mis
issippi$miss
ssippi$missi
sippi$missis
ippi$mississ
ppi$mississi
pi$mississip
i$mississipp
$mississippi

- C[*c*] : para cada carácter *c* del alfabeto almacena el número total de ocurrencias en T de los caracteres que son lexicográficamente menores que *c*.

  En el ejemplo:

  `C[$]=0  C[i]=1  C[m]=5  C[p]=6  C[s]=8`

- OCC(c, k): número de ocurrencias del carácter c en el prefijo de L (1, k)

  En el ejemplo: para k desde 1 a n=12

  ```
  Occ[$] = 0,0,0,0,0,1,1,1,1,1,1,1
  Occ[i] = 1,1,1,1,1,1,1,2,2,2,3,4
  Occ[m] = 0,0,0,0,1,1,1,1,1,1,1,1
  Occ[p] = 0,1,1,1,1,1,2,2,2,2,2,2
  Occ[s] = 0,0,1,2,2,2,2,2,3,4,4,4
  ```

- El carácter L[i] se localiza en F en la posición LF(i):

  `LF(i) = C[L[i]] + Occ(L[i],i)`

1	12	**$** mississipp **i**
2	11	**i** $mississip **p**
3	8	**i** ppi$missis **s**
4	5	**i** ssippi$mis **s**
5	2	**i** ssissippi$ **m**
6	1	**m** ississippi **$**
7	10	**p** i$mississi **p**
8	9	**p** pi$mississ **i**
9	7	**s** ippi$missi **s**
10	4	**s** issippi$mi **s**
11	6	**s** sippi$miss **i**
12	3	**s** sissippi$m **i**

L[3]=s
LF(3)=
=C[L[3]]+Occ(L[3],3)=
=C[s]+Occ(s,3)=
=8+1=9

RECUPERAR EL TEXTO:

- Si T[k] es el i-ésimo carácter de L entonces
  `T[k-1] = L[LF(i)]`

  `Ejemplo:`
  `T[k] = “p” que es el séptimo carácter de L`
  `T[k-1] = L[LF(7)] =`
  `L[C[“p”]+Occ(“p”,7)] = L[6+2] = “i”`

- Si empezamos en la posición del carácter “$” podemos recuperar el texto T completo.

1	12	**$** mississipp **i**
2	11	**i** $mississip **p**
3	8	**i** ppi$missis **s**
4	5	**i** ssippi$mis **s**
5	2	**i** ssissippi$ **m**
6	1	**m** ississippi **$**
7	10	**p** i$mississi **p**
8	9	**p** pi$mississ **i**
9	7	**s** ippi$missi **s**
10	4	**s** issippi$mi **s**
11	6	**s** sippi$miss **i**
12	3	**s** sissippi$m **i**

```
C[$]=0 C[i]=1 C[m]=5
C[p]=6 C[s]=8

Occ[$] = 0,0,0,0,0,1,1,1,1,1,1,1
Occ[i] = 1,1,1,1,1,1,1,2,2,2,3,4
Occ[m] = 0,0,0,0,1,1,1,1,1,1,1,1
Occ[p] = 0,1,1,1,1,1,2,2,2,2,2,2
Occ[s] = 0,0,1,2,2,2,2,2,3,4,4,4

LF(i) = C[L[i]] + Occ(L[i],i)
```

COUNT:

- Devuelve el número de ocurrencias de un patrón en el texto.

  Patrón: P[1,p]

  Texto: T[1,u]

- Algoritmo:

```
i=p, c=P[p], sp=C[c]+1, ep=C[c+1]
while ((sp<=ep) and (i>=2)) do
      c=P[i-1];
      sp=C[c]+Occ(c,sp-1)+1;
      ep=C[c]+Occ(c,ep);
      i=i-1;
if (ep<sp) then return “not found”
else return “found (ep-sp+1) ocurrences”
```

1	12	**$** mississipp **i**
2	11	**i** $mississip **p**
3	8	**i** ppi$missis **s**
4	5	**i** ssippi$mis **s**
5	2	**i** ssissippi$ **m**
6	1	**m** ississippi **$**
7	10	**p** i$mississi **p**
8	9	**p** pi$mississ **i**
9	7	**s** ippi$missi **s**
10	4	**s** issippi$mi **s**
11	6	**s** sippi$miss **i**
12	3	**s** sissippi$m **i**

```
C[$]=0 C[i]=1 C[m]=5
C[p]=6 C[s]=8

Occ[$] = 0,0,0,0,0,1,1,1,1,1,1,1
Occ[i] = 1,1,1,1,1,1,1,2,2,2,3,4
Occ[m] = 0,0,0,0,1,1,1,1,1,1,1,1
Occ[p] = 0,1,1,1,1,1,2,2,2,2,2,2
Occ[s] = 0,0,1,2,2,2,2,2,3,4,4,4

LF(i) = C[L[i]] + Occ(L[i],i)
```

LOCATE:

- Devuelve la posición en el texto de una determinada posición de M.
- Se marca un conjunto de filas de M y se almacena la posición en el texto de cada una de ellas.

1	12	**$** mississipp	**i**
2	11	**i** $mississip	**p**
3	8	**i** ppi$missis	**s**
4	5	**i** ssippi$mis	**s**
5	2	**i** ssissippi$	**m**
6	1	**m** ississippi	**$**
7	10	**p** i$mississi	**p**
8	9	**p** pi$mississ	**i**
9	7	**s** ippi$missi	**s**
10	4	**s** issippi$mi	**s**
11	6	**s** sippi$miss	**i**
12	3	**s** sissippi$m	**i**