Algoritmi per IR

Dictionary-based compressors

Lempel-Ziv Algorithms

Keep a “dictionary” of recently-seen strings.

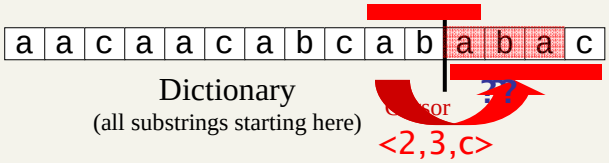
No explicit frequency estimation

The differences are:

- How the dictionary is stored
- How it is extended
- How it is indexed
- How elements are removed

LZ-algos are asymptotically optimal, i.e. their compression ratio goes to H(S) for $n \to \infty$!!

LZ77

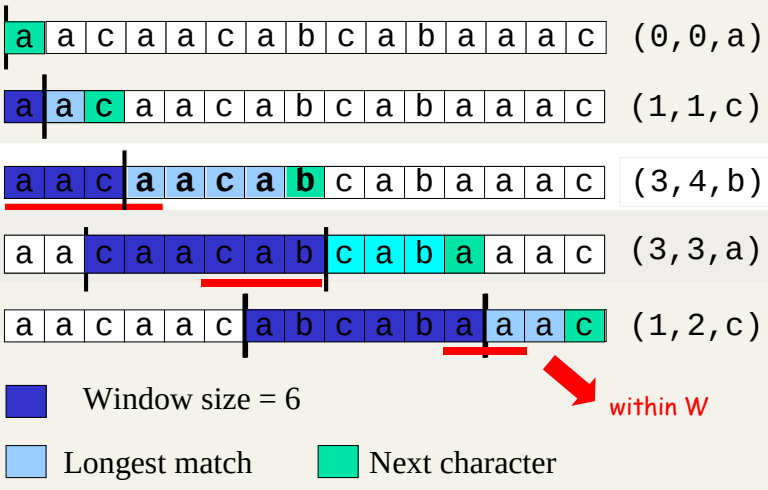


Algorithm's step:

- Output <d, len, c>
  d = distance of copied string wrt current position
  len = length of longest match
  c = next char in text beyond longest match
- Advance by len + 1

A buffer "window" has fixed length and moves

Example: LZ77 with window



LZ77 Decoding

Decoder keeps same dictionary window as encoder.

- Finds substring and inserts a copy of it

What if l > d? (overlap with text to be compressed)

- E.g. seen = abcd, next codeword is `(2,9,e)`
- Simply copy starting at the cursor

```
for (i = 0; i < len; i++)
  out[cursor+i] = out[cursor-d+i]
```

- Output is correct: abcdcdcdcdce

LZ77 Optimizations used by gzip

LZSS: Output one of the following formats

`(0, position, length)` or `(1,char)`

Typically uses the second format if length < 3.

Special greedy: possibly use shorter match so that next match is better

Hash Table for speed-up searches on triplets

Triples are coded with Huffman's code

LZ78

Dictionary:

- substrings stored in a trie (each has an *id*).

Coding loop:

- find the longest match ***S*** in the dictionary
- Output its ***id*** and the next character ***c*** after the match in the input string
- Add the substring ***Sc*** to the dictionary

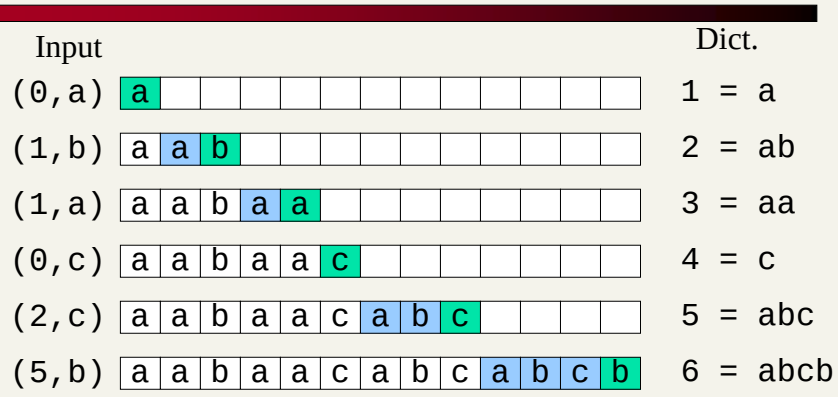
Decoding:

- builds the same dictionary and looks at *id*s

LZ78: Coding Example

	Output	Dict.
a a b a a c a b c a b c b	(0,a)	1 = a
a **a b** a a c a b c a b c b	(1,b)	2 = ab
a a b **a a** c a b c a b c b	(1,a)	3 = aa
a a b a a **c** a b c a b c b	(0,c)	4 = c
a a b a a c **a b c** a b c b	(2,c)	5 = abc
a a b a a c a b c **a b c b**	(5,b)	6 = abcb

LZ78: Decoding Example



LZW (Lempel-Ziv-Welch)

Don't send extra character ***c***, but still add ***Sc*** to the dictionary.

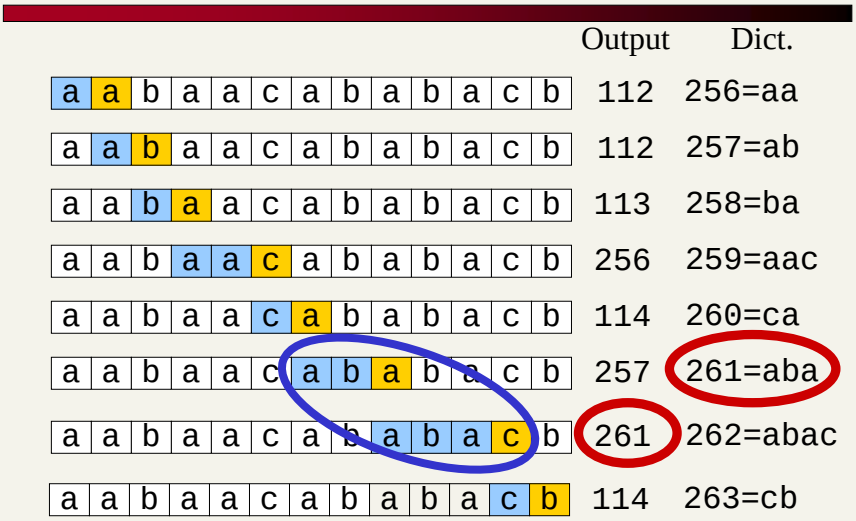
Dictionary:

- initialized with 256 *ascii* entries (e.g. a = 112)

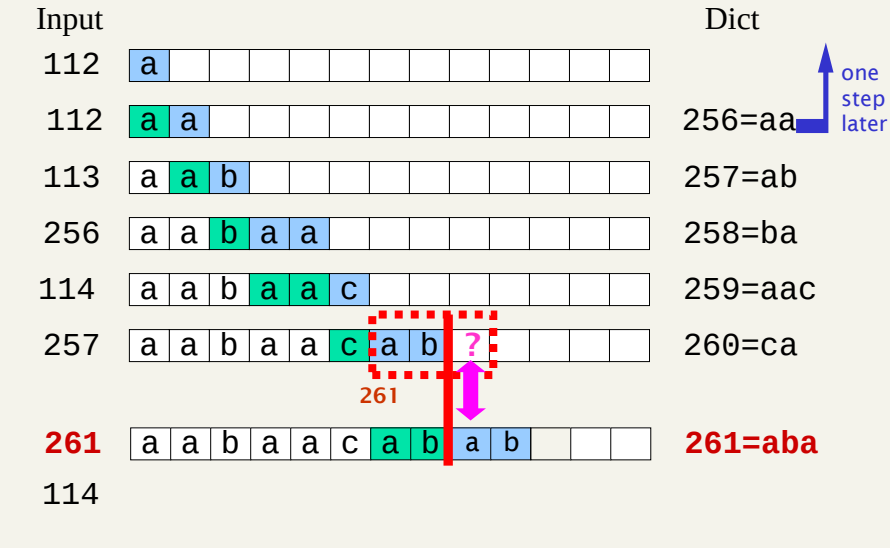
Decoder is **one step behind** the coder since it does not know c

- There is an issue for strings of the form
  SSc where *S[0] = c*, and these are handled specially!!!

LZW: Encoding Example



LZW: Decoding Example

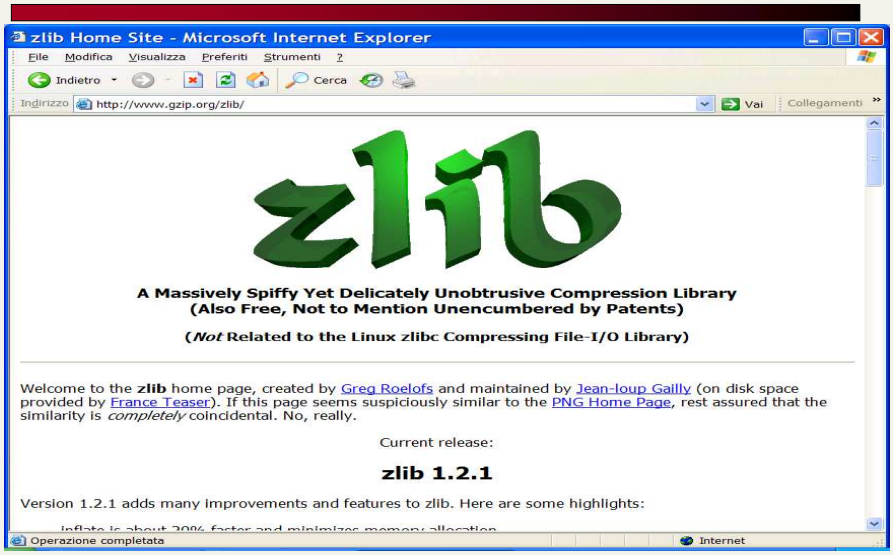


LZ78 and LZW issues

How do we keep the dictionary small?

- Throw the dictionary away when it reaches a certain size (used in GIF)
- Throw the dictionary away when it is no longer effective at compressing (e.g. `compress`)
- Throw the least-recently-used (LRU) entry away when it reaches a certain size (used in BTLZ, the British Telecom standard)

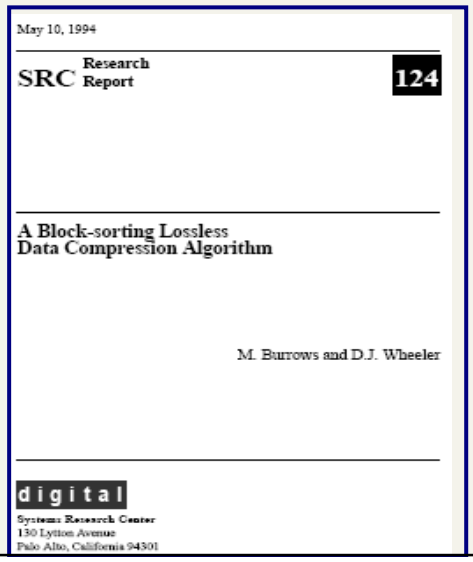
You find this at: www.gzip.org/zlib/



Algoritmi per IR

Burrows-Wheeler Transform

The big (unconscious) step...



May 10, 1994

SRC Research Report 124

A Block-sorting Lossless
Data Compression Algorithm

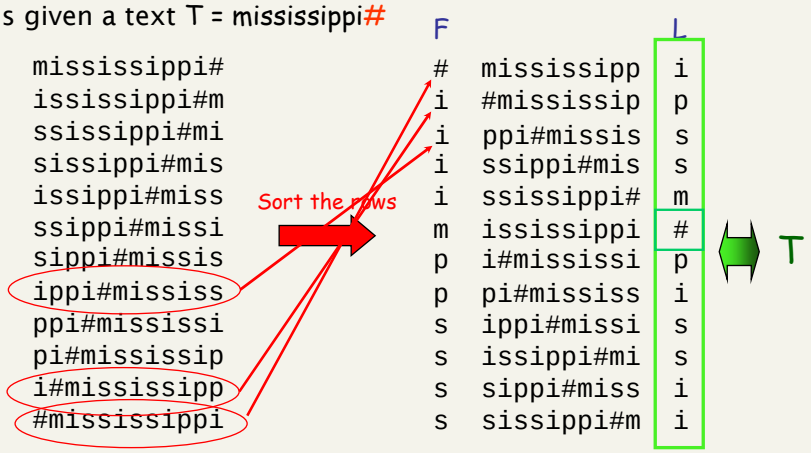
M. Burrows and D.J. Wheeler

digital

Systems Research Center
130 Lytton Avenue
Palo Alto, California 94301

The Burrows-Wheeler Transform (1994)

Let us given a text T = mississippi#

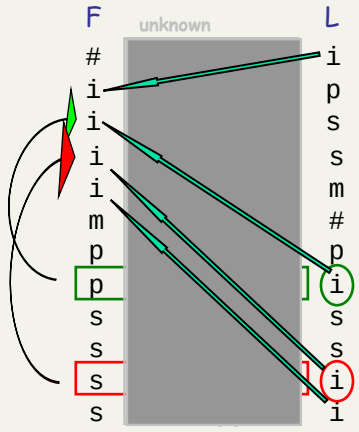


A famous example

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Much longer...

A useful tool: L → F mapping

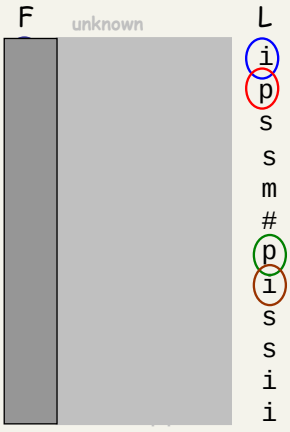


How do we map L's onto F's chars ?

... Need to distinguish equal chars in F...

- Take two equal L's chars
- Rotate rightward their rows
- Same relative order !!

The BWT is invertible



Two key properties:

1. LF-array maps L's to F's chars
2. L[i] precedes F[i] in T

Reconstruct T backward:

T = ... ippi #

InvertBWT(L)

```
Compute LF[0,n-1];
r = 0; i = n;
while (i>0) {
   T[i] = L[r];
   r = LF[r]; i--;
   }
```

How to compute the BWT ?

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

We said that: L[i] precedes F[i] in T

L[3] = T[7]

Given SA and T, we have L[i] = T[SA[i]-1]

How to construct SA from T ?

SA	
12	#
11	i#
8	ippi#
5	issippi#
2	ississippi#
1	mississippi
10	pi#
9	ppi#
7	sippi#
4	sissippi#
6	ssippi#
3	ssissippi#

Elegant but inefficient

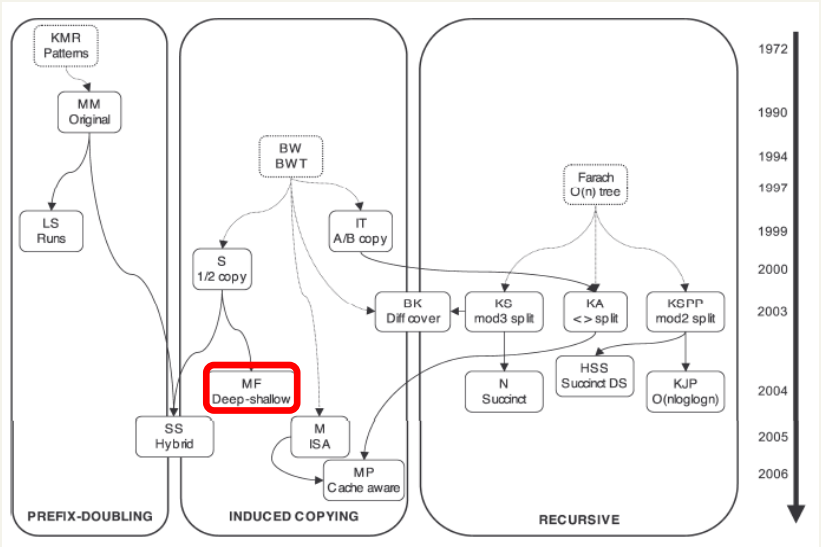
```
COMPARISON_BASED_CONSTRUCTION(char *T, int n, char **SA)
  { for(i = 0; i < n; i++)   SA[i] = T + i;
    QSORT(SA, n, sizeof(char *), Suffix_cmp); }

SUFFIX_CMP(char **p, char **q){ return strcmp(*p,*q); }
```

Obvious inefficiencies:
- $\Theta(n^2 \log n)$ time in the worst-case
- $\Theta(n \log n)$ cache misses or I/O faults

Input: T = mississippi#

Many algorithms, now...



Compressing L seems promising...

final char (L)	sorted rotations
a	n to decompress. It achieves compression
o	n to perform only comparisons to a depth
o	n transformation} This section describes
o	n transformation} We use the example and
o	n treats the right-hand side as the most
a	n tree for each 16 kbyte input block, enc
a	n tree in the output stream, then encodes
i	n turn, set $L[i]$ to be the
i	n turn, set $R[i]$ to the
o	n unusual data. Like the algorithm of Man
a	n use a single set of probabilities table
e	n using the positions of the suffixes in
i	n value at a given point in the vector $R
e	n we present modifications that improve t
e	n when the block size is quite large. Ho
i	n which codes that have not been seen in
i	n with ch appear in the {\em same order
i	n with ch. In our exam
o	n with Huffman or arithmetic coding. Bri
o	n with figures given by Bell~\cite{bell}.

Key observation:

- L is locally homogeneous

➡ L is highly compressible

Algorithm Bzip :

1. Move-to-Front coding of L
2. Run-Length coding
3. Statistical coder

☑ Bzip vs. Gzip: 20% vs. 33%, but it is slower in (de)compression !

An encoding example

T = mississippimississippimississippi

L = ipppssssssmmmii#pppiiissssssiiiiii

at 16

Mtf = [i,m,p,s]

Mtf = 02003000003003020030030000010000 0

Mtf = 03004000004004030040040000020000 0

Bin(6)=110, Wheeler's code

RLE0 = 03141041403141410210

Alphabet |Σ|+1

Bzip2-output = Arithmetic/Huffman on **|Σ|+1** symbols...

... plus γ(16), plus the original Mtf-list (i,m,p,s)

You find this in your Linux distribution