LZ77 & LZ78

Mașala Doru-Andrei
Profesor Coordonator Cristian Rusu

01

Introducere

02

State of art

03

LZ77

04

LZ78

05

LZ77 vs LZ78

06

Concluzii

Introducere

Compresie = reprezentarea în cât mai puțini biți a informației, fără a pierde conținutul esențial al acesteia

Entropia lui Shannon ne dă o limită superioară a nivelului de compresie care poate fi atins teoretic

$$H = - \sum p(x) \log p(x)$$

Există 2 tipuri de compresie: lossy și lossless
Scopul? Găsirea unui trade-off convenabil între nivelul de compresie și timpul de procesare

State of the art

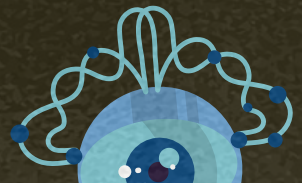
Avem mulți algoritmi de compresie moderni (GZIP, PNG, DEFLATE - ZIP, GIF)

Majoritatea au la bază o versiunea a algoritmilor LZ77 sau LZ78, algoritmi din categoria encriptorilor pe bază de dicționar.

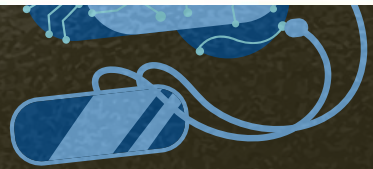
Folosesc o idee simplă, dar revoluționară: encodează string-uri care se repetă cu pointeri către apariții recente ale acestora.

LZ77

- Introdus în anul 1977 de către Abraham Lempel and Jacob Ziv.
- Funcționează prin înlocuirea string-urilor cu referințe spre apariții trecute ale acestora cu tupluri length-distance.
- Se bazează pe principiul de sliding window și lookahead buffer.



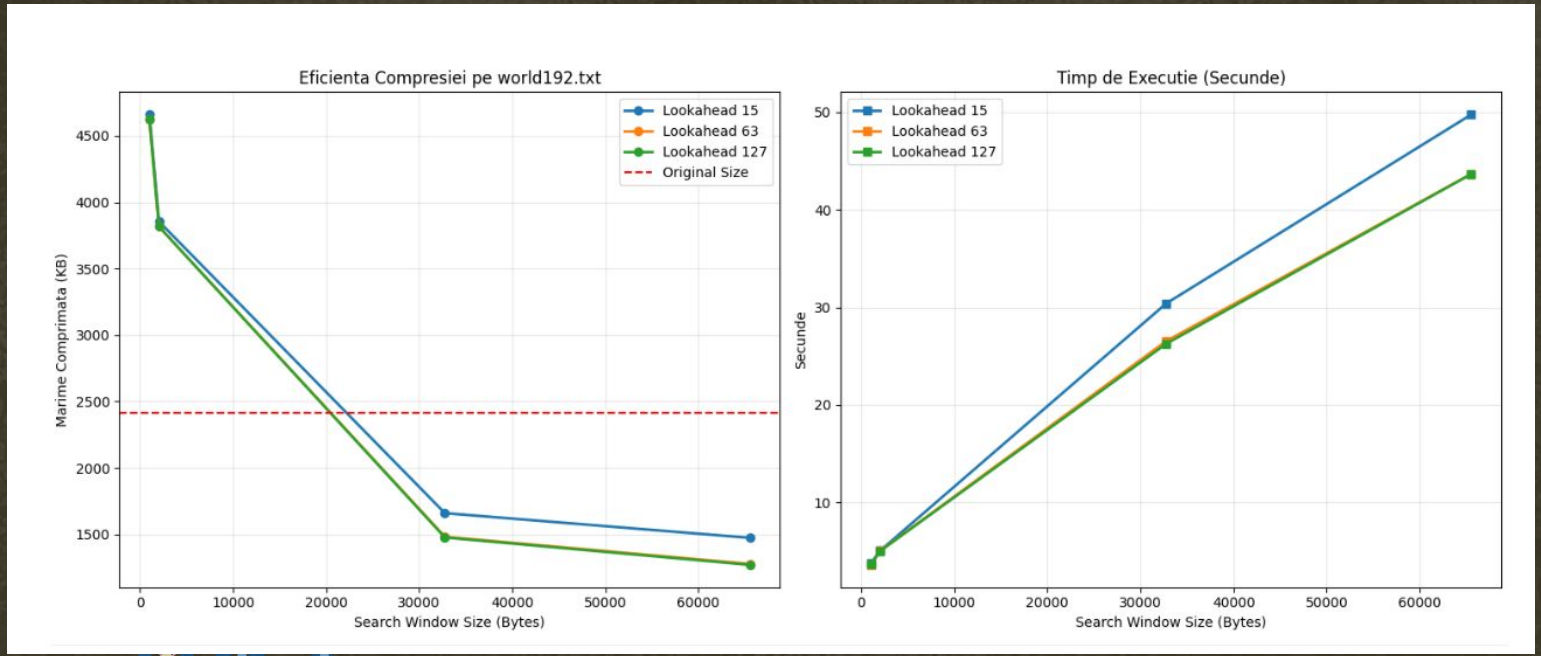
1	2	3	4	5	6	7	8	1	2	3	4	5	6	Output
								a	b	r	a	c	a	<0,0,a>
							a	b	r	a	c	a	d	<0,0,b>
						a	b	r	a	c	a	d	a	<0,0,r>
					a	b	r	a	c	a	d	a	b	<3,1,c>
			a	b	r	a	c	a	d	a	b	r	a	<2,1,d>
	a	b	r	a	c	a	d	a	b	r	a	r	r	<7,4,r>
c	a	d	a	b	r	a	r	r	a	y				<3,2,y>



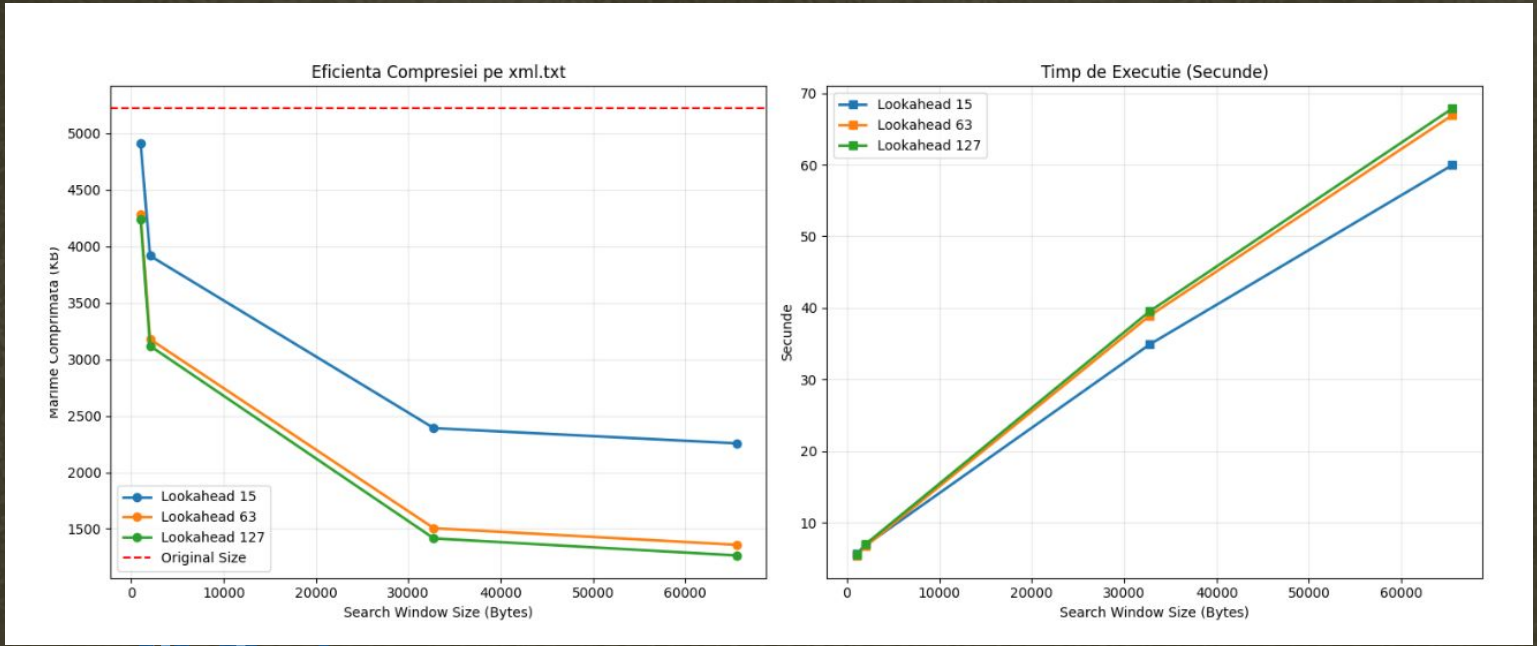
Pseudocod

```
cursor <- 0
cat timp cursor < lungime input
    // findLongestMatch cauta subsirul cel mai lung pe care il poate
       regasi in fereastra glisanta, care coincide cu subsirul care
       porneste de la pozitia actuala a cursorului (pentru eficienta,
        am ales sa caut incepand cu primele 3 caractere din subsir)
    (distanta, lungime_match) <- findLongestMatch(input, cursor)
    index_urmator <- cursor + lungime_match
    daca index_urmator < lungime_totala
        caracter_urmator <- input[index_urmator]
    altfel caracter_urmator <- 0
    output(distanta, lungime_match, caracter_urmator)
    cursor <- cursor + lungime_match + 1
scrie lungime_totala, output in fisier
```

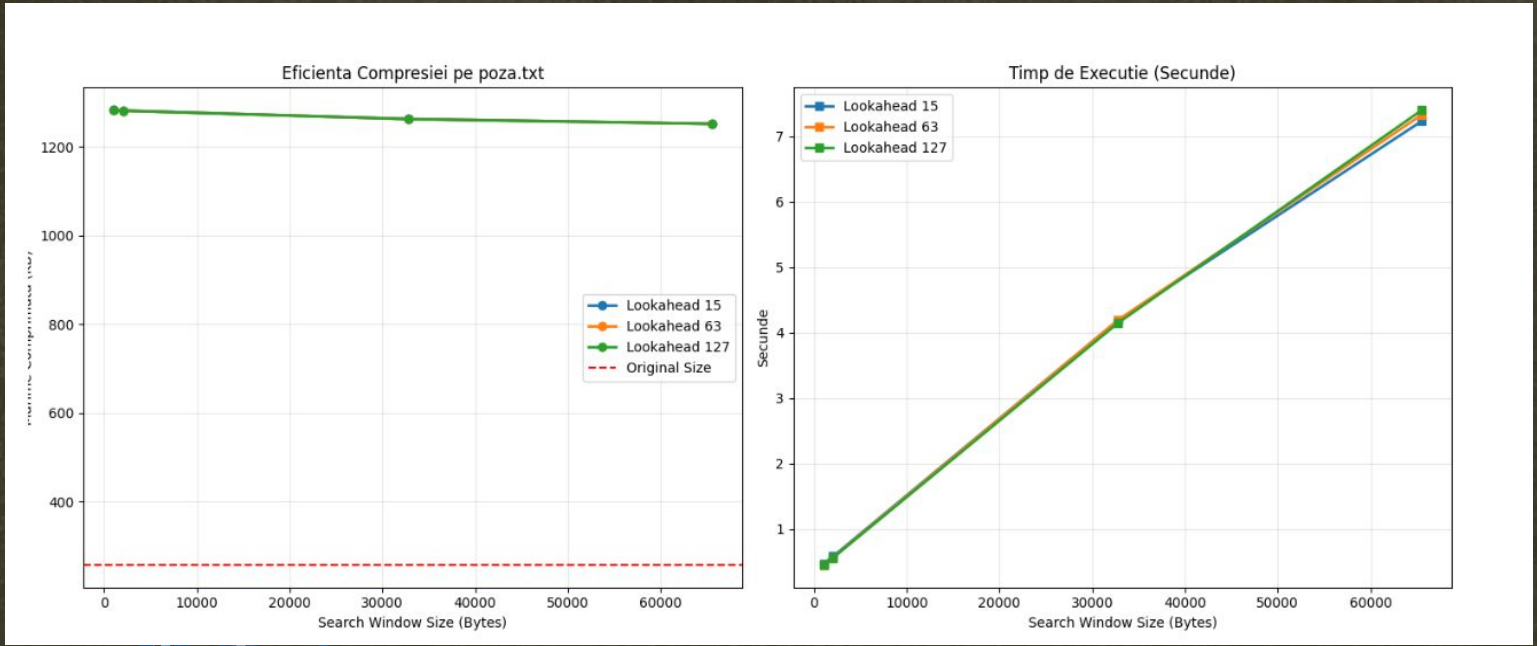
Performanta algoritmului



Performanta algoritmului

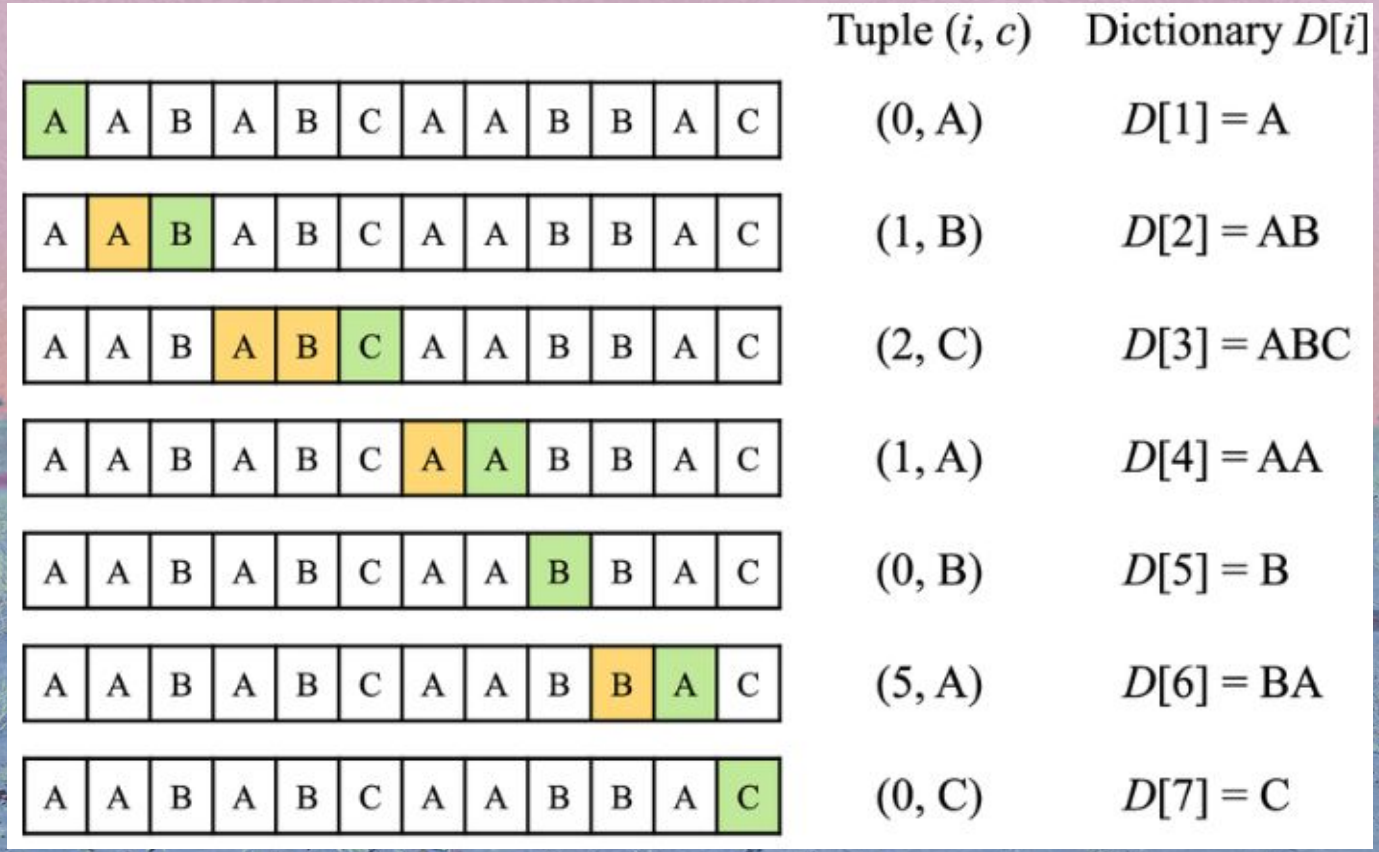


Performanta algoritmului



LZ78

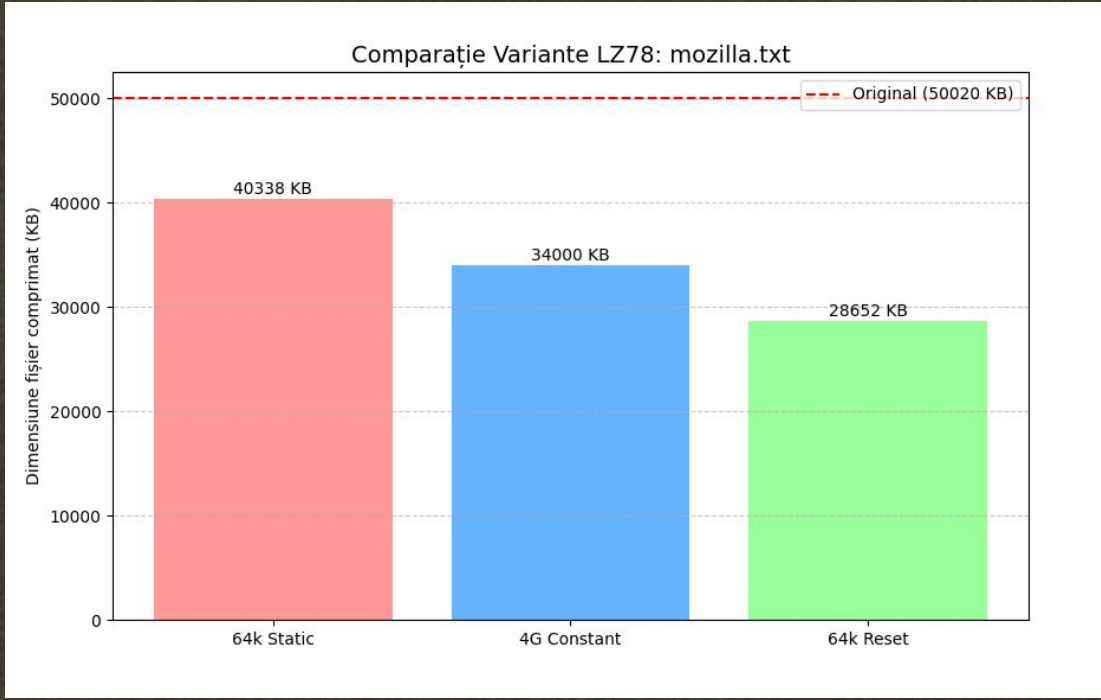
- Introdus în anul 1978 de către Abraham Lempel and Jacob Ziv.
- Spre deosebire de “fratele” său, Lempel-Ziv 2 folosește un dicționar propriu-zis
- Token-urile emise sunt de tipul (index prefix, următorul caracter)


Tuple (i, c)
Dictionary D[i]
A A B A B C A A B B A C
(0, A)
D[1] = A
A A B A B C A A B B A C
(1, B)
D[2] = AB
A A B A B C A A B B A C
(2, C)
D[3] = ABC
A A B A B C A A B B A C
(1, A)
D[4] = AA
A A B A B C A A B B A C
(0, B)
D[5] = B
A A B A B C A A B B A C
(5, A)
D[6] = BA
A A B A B C A A B B A C
(0, C)
D[7] = C

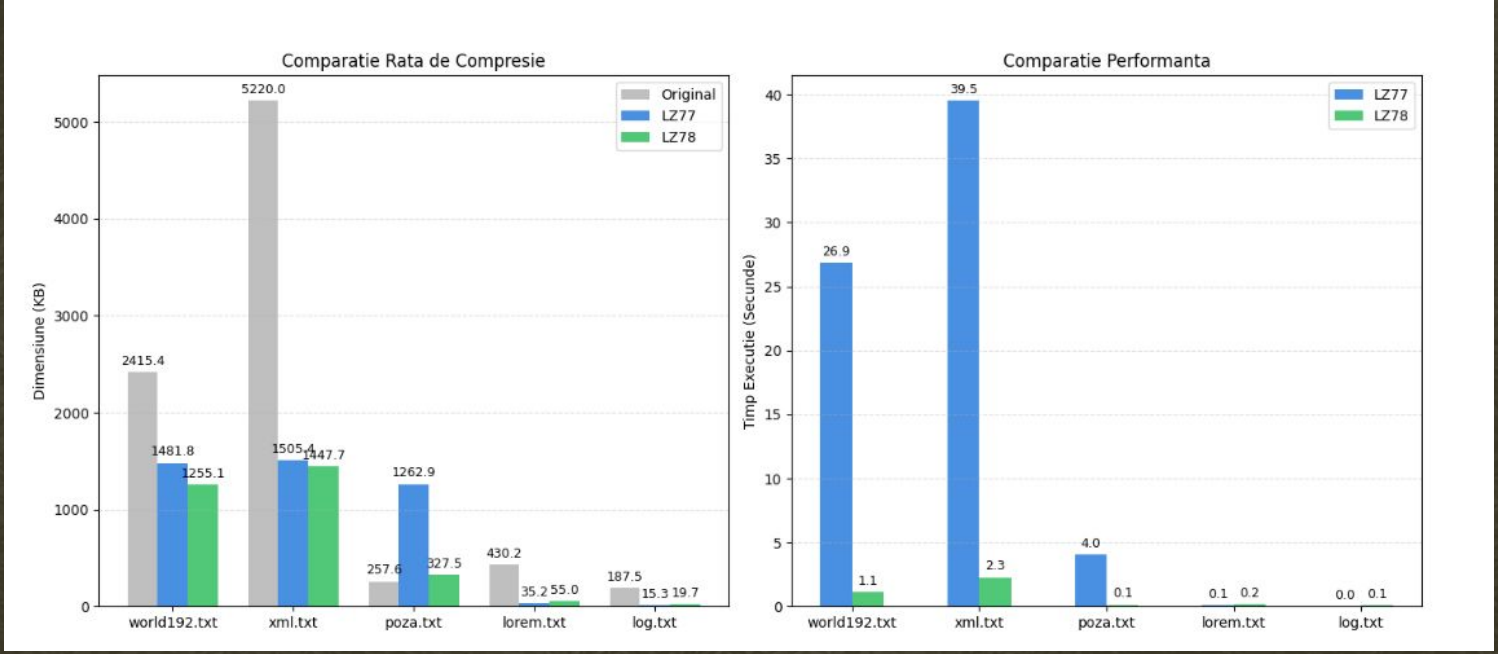
Pseudocod

```
Initializez dictionarul cu sirul vid pe pozitia 0, cursor <- 0 si
   size_dict <- 1
cat timp cursor < lungime_input
    string_curent <- ""
    ultimul_match <- 0
    // Aici caut cel mai lung prefix din dictionar
    cat timp (string_curent + caracter[cursor]) exista in dictionar
        string_curent <- string_curent + caracter[cursor]
        ultimul_match <- index(string_curent)
        cursor += 1

    // Verific daca nu am terminat fisierul
    daca cursor < lungime_input
        next_char <- input[cursor]
        cursor += 1
    altfel
        next_char <- 0

    scrie(ultimul_match, next_char)

    daca size_dict < 65535
        adauga (string_curent + next_char) in dictionar la pozitia
           size_dict
        size_dict += 1
    altfel
        goleste dict
        size_dict = 1
```

Diferite implementari ale dictionarului



LZ77 vs LZ78


Comparatie Rata de Compresie
Dimensiune (KB)
Original
LZ77
LZ78
2415.4
1481.8
1255.1
5220.0
1505.4
1447.7
257.6
1262.9
327.5
430.2
35.2
55.0
187.5
15.3
19.7
world192.txt
xml.txt
poza.txt
lorem.txt
log.txt
Comparatie Performanta
Timp Executie (Secunde)
LZ77
LZ78
26.9
1.1
39.5
2.3
4.0
0.1
0.1
0.2
0.0
0.1
world192.txt
xml.txt
poza.txt
lorem.txt
log.txt

Concluzii

- Nu există un algoritm “mai bun” decât celălalt
- LZ77 se descurcă mult mai bine pe texte cu limbaj natural și texte repetitive
- LZ78 triumfă când vine vorba de fișiere mari cu redundanță globală
- De asemenea, pe fișiere mari, LZ78 este ceva mai rapid
- Niciun algoritm nu se descurcă cu informație random

Bibliografie

- „Adaptive data compression system with systolic string matching logic", 5532693, US Patent, 1996.
- Canterbury Corpus, url: https://corpus.canterbury.ac.nz/descriptions/.
- IEEE Global History Network, Milestones: Lempel–Ziv Data Compression Algorithm, 1977, Institute of Electrical și Electronics Engineers, 22 Iul. 2014, (accesat în 9.11.2014).
- Lossless Data Compression: LZ78, url: cs.stanford.edu.
- Silesia Corpus, url: https://sun.aei.polsl.pl/~sdeor/index.php?page= silesia.
- Jacob Ziv și Abraham Lempel, „A Universal Algorithm for Sequential Data Compression", în IEEE Transactions on Information Theory 23.3 (1977), pp. 337–343, doi: 10.1109/TIT.1977.1055714.
- Jacob Ziv și Abraham Lempel, „Compression of Individual Sequences via VariableRate Coding", în IEEE Transactions on Information Theory 24.5 (1978), pp. 530– 536, doi: 10.1109/TIT.1978.1055934.