DESIGN AND IMPLEMENTATION OF A DICTIONARY-BASED ARCHIVER

Radu RĂDESCU[1]

Acest articol îşi propune să prezinte un arhivator de uz general construit folosind tehnica de translaţie cu ajutorul dicţionarului. Contribuţia originală constă în metoda de arhivare a indecşilor astfel obţinuţi Acest arhivator se poate folosi pentru realizarea compresiei fără pierderi aplicate oricăror tipuri de fişiere. Aplicaţia creată poate oferi concluzii utile privind performanţele de compresie şi influenţa dicţionarului ales asupra parametrilor.

This paper intends to present a common use archiver, made up following the dictionary technique. The original contribution of the paper resides in using the index archiving method. This archiver is useful in order to accomplish the lossless compression for any file types. The application can offer useful conclusions regarding the compression performances (compression ratio and packing time) and the influence of the chosen dictionary over the parameters.

Key words: data lossless compression, dictionary codes, archive characters

1. Introduction

The archivers using dictionary techniques [1], [2], [3] can be very efficient, especially when using some files that have different words very often repeated. This happens because the archivers generate their dictionary during the archiving process, the program „learning" new words this way. Because the application can make an archive that contains more files, this has to be very well configured, so that during the unpacking of the files it can be separated with lossless information.

2. Structure of the archived file

For the beginning, the header file structure is presented:

- 3 bytes to store 3 letters (CBA). These letters are used as the identification of the archive. It is very important to verify these characters in order not to let the archiver to try unpacking a file that is not a CBA archive.
- 2 bytes to store the maximum length of the dictionary.

[1] Reader, Dept. of Applied Electronics and Information Engineering, University "Politehnica" of Bucharest, Romania

- 2 bytes to store the minimum length of the dictionary.
- 1 byte to store the settings. This byte is used to store 3 binary validation variables:
  - 1 bit for the path of the file;
  - 1 bit for the unpacked size of the file;
  - 1 bit for the password.
- 1 byte to store the length of the password (optionally).
- *n* bytes to store the password, *n* being the length of the password (optionally).
- 2 bytes to store the number of files.

After describing the header of the files, it repeats the following sequence for every new added archive file:

- 2 bytes to store the length of the string that contains the path of the file (optionally).
- *s* bytes for the *s* characters of the string that contains the path of the file (optionally).
- 1 byte for the length of the file name.
- *f* bytes for the *f* characters of the file name.
- 4 bytes to store the unpacked size of the file (optionally).
- 1 byte for the archiver type. Some files can only be copied in the archive, because their archiving would cause the increasing of the file size.
- 4 bytes for the size of the packed file.
- *nr* bytes for the *nr* characters of the packed file.

3. Packing and unpacking of the files

The packing and the unpacking of the files contain two big stages.

3.1. Transformation of the initial characters into dictionary codes

This stage is accomplished using the Lempel-Ziv-Welch (LZW) dictionary compression [1], [2], [3], [4], [7], [8]. Initially, it begins with a 257-word dictionary, i.e., the 256 ASCII characters and a special word that indicates the end of the file. Opposed to the classical version of this method, a dictionary limitation appears in the described application. Therefore, when getting to a maximum size of the dictionary, this one will be deleted to a minimum value. The user can set both the maximum and minimum values. According to the chosen values, the number of the packed files and the compression time change. The choice of a too large maximum value of the dictionary results in a very long waiting time, getting a too small dimension improvement. The optimal values for the two limitation dictionary variables are different from one file to another.

3.2. Transformation of the dictionary codes into archive characters

This method relies on tackling from two different perspectives of two strings of numbers, having the same basic table. Dictionary codes greater than 256 elements cannot be written in the archive using only one byte. Therefore, it is necessary to have a 2 bytes space. This space is too large comparing it to the necessary one, especially in the initial phases, where the dictionary has not a large size.

From the first steps, the dictionary has a maximum of 512 elements, and the dictionary code can be written on 9 bits from the 16 available bits. Hence, 7 out of the 16 available bits remain unused, meaning almost half of the overall space. Grouping 8 codewords, 8 × 9 bits = 72 bits are needed. It can be written on 72 bits / 8 bits = 9 bytes, comparing to the 16 bytes usually needed. This is available for dictionary codes stored on more than 9 bits too, reducing the gained from the classical alternative.

Table 1 refers to the transformation of codewords (the dictionary indexes) into archive characters.

Table 1

Transformation of dictionary codes into archive characters

Codewords / Archive characters	161	231	44	182	14	93	152	137	241
423	1	1	0	1	0	0	1	1	1
137	0	1	0	0	0	1	0	0	1
481	1	1	1	1	0	0	0	0	1
45	0	0	0	1	0	1	1	0	1
94	0	0	1	0	1	1	1	1	0
248	0	1	1	1	1	1	0	0	0
176	0	1	0	1	1	0	0	0	0
395	1	1	0	0	0	1	0	1	1

Initially, it works with a 257-word dictionary (256 characters + 1 end of file control character). Table 1 is, in fact, an example in which the dictionary has a number of words ≤ 512. On the first column (423, 137, 481, 45, ...) there are the codewords (dictionary indexes), which have values up to 512, so that can be written on 9 bits. An archive character has 8 bits and because of this reason cannot be written directly, because it would not be efficiently. The codewords (423, 137, 481, 45, ...) are binary written on the rows.

In order to have the certainty to obtain archive characters (8 bits), 8 codewords are used:

$$(8 \text{ codewords}) \times (9 \text{ bits}) = (9 \text{ archive characters}) \times (8 \text{ bits}). \quad (1)$$

The first row (161, 231, 44, 182, ...) contains the character words obtained by transforming every column from binary to the 10^{th} base.

For example,

$$161 = 1 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0. \quad (2)$$

It works similarly for the other values: 231, 44, 182, ...

In the case in which the dictionary has between 512 and 1024 words, the procedure is similar, Table 1 having 8 rows, but one extra column, because every dictionary word needs 10 bits (10 columns):

$$(8 \text{ codewords}) \times (10 \text{ bits}) = (10 \text{ archive characters}) \times (8 \text{ bits}). \quad (3)$$

4. Experimental results

In order to test the application, different file types are used, in order to remark the behavior of the archive [5], [6], [9], [10].

In Table 2, the characteristics of the files used in compression testing are presented.

Table 2

Experimental files

File type	File no.	Min. size KB	Max. size KB	Total size B	Average size B
XLS	6	1.08	84.5	224420	37403
DOC	3	44	77.5	199680	66560
PPS	2	111	179	296960	148480
PAS	6	0.53	1.81	6247	1041
EXE	6	11.4	83.8	195050	32508
RAR	3	16.7	100	163185	54395
BMP	5	1.24	47.5	120148	24030
WAV	4	1.16	78.9	97286	24322
DLL	6	7	69	147968	24661
MID	3	21.5	39.1	86425	28808

Next, the results of the compression are shown according to the maximum size of the dictionary.

The compression ratio and the compression time are calculated for the next pair of values representing the minimum size and the maximum size of the dictionary, expressed in number of words: (256, 512), (256, 640), (256, 768), and (256, 1024).

The corresponding values are represented in Table 3÷6.

Table 3

Compression ratio, compression time, and compression speed for parameters (256, 512)

File type	Dimension B	Compression ratio %	Compression time ms	Compression speed KB/s
XLS	111924	49.87	42	5.22
DOC	82044	41.09	33	5.91
PPS	257445	86.69	70	4.14
PAS	3717	59.50	2	3.05
EXE	153063	78.47	45	4.23
RAR	163185	100.00	44	3.62
BMP	70209	58.44	24	4.89
WAV	96723	99.42	26	3.65
DLL	108927	73.62	33	4.38
MID	70956	82.10	20	4.22

Table 4

Compression ratio, compression time, and compression speed for parameters (256, 640)

File type	Dimension B	Compression ratio %	Compression time ms	Compression speed KB/s
XLS	103614	46.17	42	5.22
DOC	79726	39.93	35	5.57
PPS	264199	88.97	73	3.97
PAS	3458	55.35	2	3.05
EXE	152520	78.20	45	4.23
RAR	163185	100.00	45	3.54
BMP	68126	56.70	24	4.89
WAV	96506	99.20	27	3.52
DLL	107743	72.82	34	4.25
MID	69431	80.34	20	4.22

Table 5

Compression ratio, compression time, and compression speed for parameters (256, 768)

File type	Dimension B	Compression ratio %	Compression time ms	Compression speed KB/s
XLS	97888	43.62	42	5.22
DOC	77860	38.99	36	5.42
PPS	266541	89.76	78	3.72
PAS	3414	54.65	1	6.10
EXE	151155	77.50	46	4.14
RAR	163185	100.00	48	3.32
BMP	66000	54.93	25	4.69
WAV	96292	98.98	28	3.39
DLL	106453	71.94	36	4.01
MID	67374	77.96	22	3.84

Table 6

Compression ratio, compression time, and compression speed for parameters (256, 1024)

File type	Dimension B	Compression ratio %	Compression time ms	Compression speed KB/s
XLS	90852	40.48	45	4.87
DOC	76123	38.12	39	5.00
PPS	268006	90.25	87	3.33
PAS	3396	54.36	3	2.03
EXE	150764	77.30	48	3.97
RAR	163185	100.00	53	3.01
BMP	64273	53.49	27	4.35
WAV	96056	98.74	30	3.17
DLL	15846	10.71	39	3.71
MID	66594	77.05	23	3.67

From the obtained values, the diagrams shown in Figs. 1÷4 are derived.

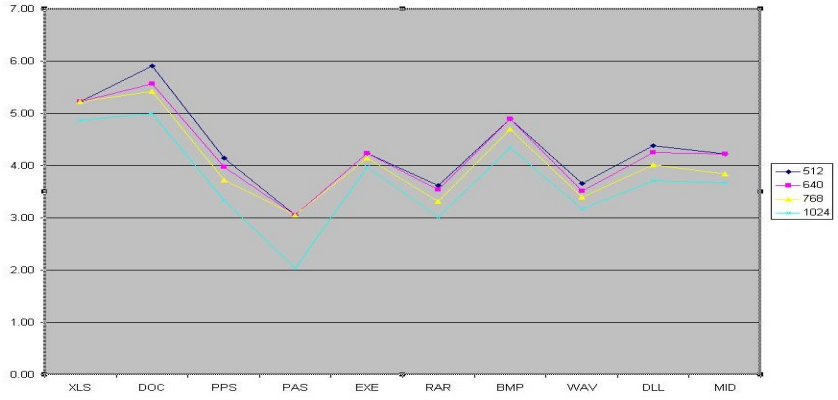

Fig. 1. Compression speed for different file types (KB/s).

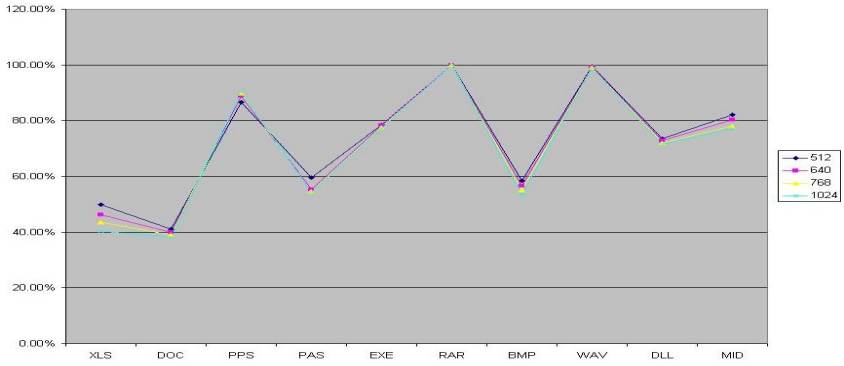

Fig. 2. Compression ratio for different file types (%).

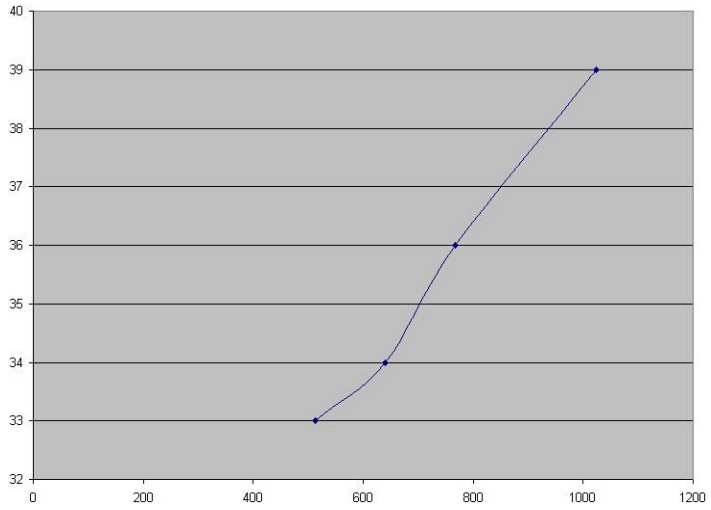

Fig. 3. Compression time according to the maximum size of the dictionary (s).

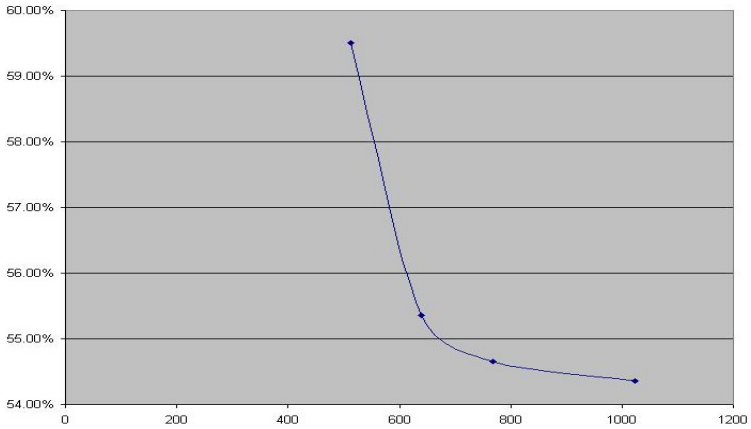

Fig. 4. Compression ratio according to the maximum size of the dictionary (%).

6. Conclusions

The application described in this paper represents a good example of the way the archive performance and the waiting time are determined in the case of the alternation of the dictionary, making thus easier to understand the dictionary-based lossless compression. At the same time, the indexes archiving method can be very efficiently used not only by specialized archivers [11], but also in programs that manage information. This method is also recommended for storing the information for long time, where it is necessary only to check periodically the information, because of the good archiving speed.

REFERENCES

[1] *A. T. Murgan*, Principiile teoriei informației în ingineria informației şi a comunicațiilor, Romanian Academy Press, Bucharest, 1998 (in Romanian).
[2] *R. Rădescu*, Transmisiunea digitală a informației – lucrări practice, Polytechnic Press, Bucharest, 2007 (in Romanian).
[3] *R. Rădescu*, Compresia fără pierderi – metode şi aplicații, Matrix Rom Press, Bucharest, 2003 (in Romanian).
[4] *R. Rădescu*, "Sistem integrat de studiu al compresiei fără pierderi a datelor", Symposium of Educational Technologies on Electronic Platforms in Engineering Higher Education, Technical University of Civil Engineering of Bucharest, 9-10 May 2003, pp. 415-422, Conspress Bucharest, 2003 (in Romanian).
[5] *R. Rădescu*, *I. Bălăşan*, "Recent Results in Lossless Text Compression Using the Burrows-Wheeler Transform (BWT)", Proceedings of IEEE International Conference on Communications 2004 (COMM04), pp. 105-110, Bucharest, Romania, 3-5 June 2004.
[6] *R. Rădescu*, *R. Popa*, "On The Performances of Symbol Ranking Text Compression Method", Scientific Bulletin of the "Politehnica" University of Timişoara, Romania, Transactions on Electronics and Communications, special issue dedicated to the Electronics and Telecommunications Symposium ETC 2004, 22-23 October 2004, vol. 49 (63), no. 2, pp. 25-27, 2004.
[7] *R. Rădescu*, *Şt. Olteanu*, "Compresia de text şi de imagini cu algoritmi LZW derivați", EEA Revue of Electro-technique, Electronics and Automatics, vol. 53, no. 4, pp. 7-10, October-December 2005 (in Romanian).
[8] *R. Rădescu*, *Al. Ene*, "Interactive Learning of Lossless Compression Methods", Proceedings of the Symposium "Educational Technologies on Electronic Platforms in Engineering Higher Education" (TEPE 2005), Technical University of Civil Engineering of Bucharest, 27-28 May 2005, pp. 211-218, CONSPRESS Publishing House, 2005.
[9] *R. Rădescu*, *C. Harbatovschi*, "Compression Methods Using Prediction By Partial Matching", Proceedings of the 6th International Conference Communications 2006 (COMM2006), pp. 65-68, Bucharest, Romania, 8-10 June 2006.
[10] *R. Rădescu*, *C. Bălănescu*, "Lossless Text Compression Using the Star (*) Transform", Proceedings of the 6th International Conference Communications 2006 (COMM2006), pp. 69-71, Bucharest, Romania, 8-10 June 2006.
[11] www.rar.com