A C# Software Implementation of the Golomb Encoding Method for Text Compression

Student authors: Danijel M. Pavlović[1] and Marko B. Mitić[1]

Mentors: Radomir S. Stanković[2] and Dušan B. Gajić[2]

Abstract – **Lossless data compression is a very important class of data compression algorithms. Golomb encoding is a type of prefix encoding which is often used as part of complex lossless data compression algorithms. In this paper, we present a C# software implementation of Golomb encoding with an intuitive graphical user interface. Experimental results, showing compression values and algorithm running times, are also presented and discussed. The main motivation for developing this software solution was to create an educational tool that can help in better understanding and presentation of algorithms for data compression.**

Keywords – **C# programming solution, data compression, Golomb code, text compression.**

I. INTRODUCTION

The large amount of data which people use in every day work is forcing the data compression methods to develop fast and give better and better results in lossless data compression. Golomb encoding [1] is a type of a prefix code often used for such purposes. A prefix code is a variable size code that satisfies the prefix property. Golomb encoding is based on run-length encoding (RLE). For example, if there is a binary string where zero appears with probability p and a one appears with probability 1-p, and with the growth of p it is more likely that longer sequences of one and the same values will appear. But, whether this is true for a particular data set to be coded highly depends on the volume of the data itself and the preliminary chosen code type.

The probability of a run of n zeros is p^n and the probability of a run of n zeros followed by a 1 is $p^n(1-p)$, indicating that run lengths are distributed geometrically [2].

Golomb code for nonnegative integers n depends on the choice of parameter m, which is median and its value is such that about half the run lengths are shorter than m and about half are equal to or greater than m, which make this code parameterized prefix code. In computing the Golomb code [4] we need three other quantities q (quotient), r (remainder), and c. These three quantities are based on the following equations:

$$q = \left\lfloor \frac{n}{m} \right\rfloor, \quad (1)$$

$$r = n - qm, \quad (2)$$

$$c = \lceil \log_2 m \rceil. \quad (3)$$

The code is constructed in two parts; the first is the value of q, coded in unary, and the second is the binary value of r coded in a special way. The first $2^c - m$ values of r are coded, as unsigned integers, in c-1 bits each and the rest are coded in c each (ending with the biggest c-bit number, which consists of c 1`s). Example of calculating necessary quantities and creating family of Golomb codes for m = 2 through 7 are shown in Table I.

TABLE I
FAMILY OF GOLOMB CODES FOR M = 2 THROUGH 7.

m	2	3	4	5	6	7
c	1	2	2	3	3	3
-m	0	1	0	3	2	1

m/n	0	1	2	3	4	5	6
2	0|0	0|1	10|0	10|1	110|0	110|1	1110|0
3	0|0	0|10	0|11	10|0	10|10	10|11	110|0
4	0|00	0|01	0|10	0|11	10|00	10|01	10|10
5	0|00	0|01	0|10	0|110	0|111	10|00	10|01
6	0|00	0|01	0|100	0|101	0|110	0|111	10|00
7	0|00	0|010	0|011	0|100	0|101	0|110	0|111

Software implementation, which we will discuss in this paper, implements the algorithm for lossless data compression [5] using the Golomb code. We will also say a few words about experimental results which we collected from the application. In the end, we will also offer some conclusions.

II. SOFTWARE IMPLEMENTATION DETAILS

A. Implementation details

Application which is presented in this work is developed in C# programming language and .NET 3.5 framework. Designed graphical user interface is very simple because focus is not on interface, it is on implemented algorithm. Also application is very easy for usage. Simply choose an input text

Student authors:
[1]Danijel Pavlović and Marko Mitić are with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia, E-mails: daki88@elfak.rs, zmicezmaj@elfak.rs

Mentors:
[2]Radomir Stanković and Dušan Gajić are with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Nis, Serbia, E-mails: radomir.stankovic@elfak.ni.ac.rs, dusan.gajic@elfak.ni.ac.rs.

file for compressing, and choose output text file if it exist, if not just type name for that output file and application will create it for you, then start compression on one single button.

Architecture of application composed from two forms, first of them is main form which contains all functionality for making compression complete, and the second form which is just user manual. Input in application is text file for compression, and the output is compressed text file. Graphical view of application architecture is shown in Figure 1.

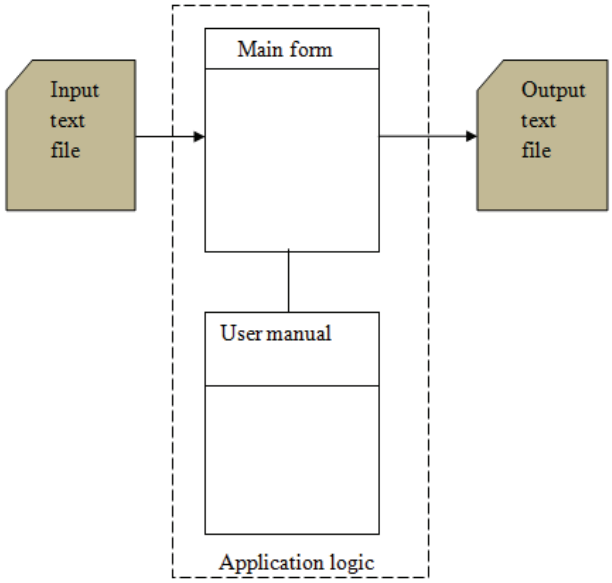

Figure 1. Architecture of the application.

Main problem during developing this application was that bit-level operations are not a natural way of thinking in C# [3]. As it is explained before Golomb code uses run lengths of zeros and ones for creation families of codes for coding those values. Therefore in the application work with bits is simulated on special way.

As we implemented algorithm for text compression, and as we knew that characters inside the text are coded with values from ASCII table which consist of zeros and ones, we read one character from text, convert it to its ASCII value, and we saved it in one string variable. Every other character is also converted to ASCII value and concatenated to the string variable that is used in the beginning. This process practically converts whole text document into string variable that represents bit representation of text. Example of this conversion is shown in Example 1.

Text = “abc”

Characters	a	b	c
ASCII values	1100001	1100010	1100011

Result string = “a”+”b”+”c”=
=”1100001”+”1100010”+”1100011”=
=“110000111001011000011”

Example 1. Conversion from text to a binary string value.

Based on that string value, we calculated all necessary parameters that are explained in introduction. Using that parameters we created family of Golomb code. Family of code is used for coding run lengths of zeros in primary string value. On that way whole text from document is encoded with Golomb code. Example of encoding is shown in Example 2.

StringForCoding =“110000111001011000011”

Run length of zeros	Code
0	0
1	10
2	110
3	1110
4	11110

Encoded String = “10111110101011101101011110101”

Example 2. Encoding of a binary string.

When we have encoded binary string we convert that string into ASCII values, practically we do reverse process compared to the previous. We read from this encoded binary string values of zeros and ones for one ASCII character, and convert that zeros and ones into ASCII character. That character is written into output text file. This process is repeating while there are still not processed zeros and ones in encoded binary string. When this process is finished also is finished whole process of text compression.

Because of using the C# development environment and .NET 3.5 framework during implementation of this solution, it is necessary for proper functioning of application to use application on Microsoft Windows platforms with installed .NET framework 3.5. In any other case application will not work properly.

B. Graphical user interface details

Graphical user interface is shown in Figure 2. The user interface is designed to be as simple as possible. There are text fields where is shown path to the input and output file respective, when they are chosen by the user of application by clicking on the browse buttons. Below browse section of the main form, is statistical section where user can see details from compression, like size of input and output file, output/input ratio, which is calculated using equation 4, and run time for compression.

$$\frac{Output}{InputRatio} = \frac{sizeOfOutputFile}{sizeOfInputFile} \times 100\% \qquad (4)$$

There is also one list box on the form which is used to show how run lengths of zeros are encoded with created family of Golomb code. In the top right corner of main form is a button which starts user manual form. In the top left corner there is a button which starts info form about developers.

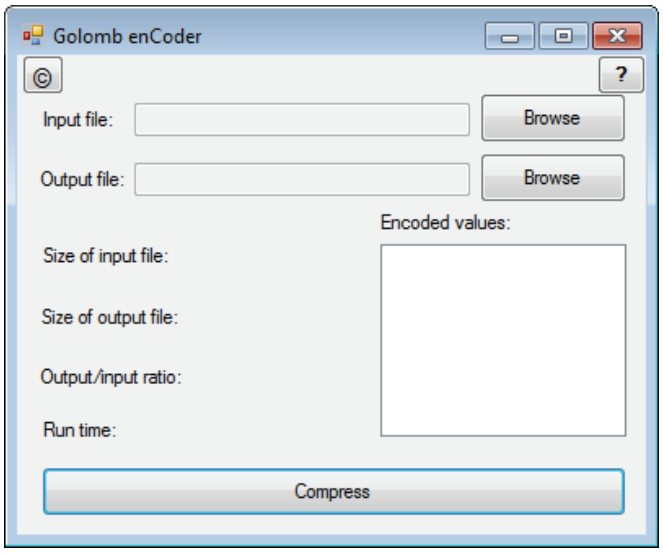

Figure 2. Graphical user interface of the application.

III. Experimental Results and Discussion

During the testing of the application for different sizes of input files we collected different results. The application is tested on a desktop PC with the specification given in Table II. Results and values of input, output files and run times are shown in the Table III.

As we can see from the Table III results are not so good. The main reason why the ratio is not so good is in characteristics of the Golomb code. Golomb code as it said before, is a parameterized prefix code. It depends on median *m* of run lengths of zeros, and *m* depends on probability *p* of appearance of zero lengths. The nature of ASCII code for characters is that there are no long runs of zeros, which are necessary for creating a good family of Golomb codes. A good family of the Golomb code provides us with better output/input ratio.

Because of this fact we tried to improvise long runs of zeros. If we look at ASCII table we can see that long run of zeros is in character "*space*". We tested our application with text file in which we have table of integer values, in that table columns are separated with space character. As we can see from the Table III results for input/output ratio showing us minimal compression of the input files. In this kind of the input files the runs of zeros are more frequently than in input files with text content, therefore and parameters for Golomb code are better as well as family of codes.

Table II
Test PC specification

CPU	Pentium® Dual-Core CPU E5200 @2.5 GHz
RAM	2 GB DDR2
OS	Windows 7 (64-bit)
GPU	NVIDIA GeForce 9400 GT
Motherboard	MSI Intel P31 Neo-F

Table III
Experimental results for input text files with table of integer content

Size of input file (KB)	Size of output file (KB)	Output/Input ratio (%)	Run time (ms)
0.53	0.49	92.45	13
1.05	0.99	94.29	32
2.00	1.88	94.00	56
5.96	5.58	93.62	820
10.03	9.39	93.60	3570
20.04	18.78	93.71	10356
50.26	47.1	93.71	140917
100.3	93.99	93.72	506916

Beside the bad output/input ratio from the Table III, we can also see that run time for compression is unsatisfactorily. The cause for long compression time is in chosen development environment. Like it is said before the main problem in implementation was because C# is not created with focus on bit-level operations. Because of this constraint we had to do conversions between data types into two directions, as we explained in implementation details, and this significantly slows the process of compression. If we had chosen some other development environment in which working with bits is faster and easier, for example C or C++, the run time for compression will have been undoubtedly shorter, but good compression ratio will be probably the same like in this example.

IV. Conclusions

In this paper we presented the software solution which implements the algorithm for text compression using the Golomb code. Considering what has been said above we can conclude that: the Golomb code is not the best choice, for algorithms which are used for text compression. This is because of the fact that the probability of run lengths of zeros in binary text representation is too small. The other reason is when we want to implement this algorithm for compression which uses the Golomb code, it is better to use it in a development environment which has ability and good interface for working with bits.

The Golomb code would show better results in compression for a different type of data. The data where probability, of run lengths of zeros, is big are suitable for encoding with this code, for example monochromatic pictures.

The application which is presented in this paper is developed for educational purposes. It can be a good educational tool for presentation and understanding the compression algorithm which uses the Golomb code for text compression.

REFERENCES

[1] David Salomon, *Data Compression – The Complete Reference*, Springer, 2007.

[2] *Golomb coding*, http://en.wikipedia.org/wiki/Golomb_coding, website last visited on 12/04/2011.

[3] *C# Tutorials*, http://msdn.microsoft.com/en-us/library/aa288436(v=vs.71).aspx, website lat visited on 12/12/2010.

[4] *Compression Algorithms*, http://www.inference.phy.cam.ac.uk/itprnn/code/c/compress/, website last visited 12/04/2011.

[5] *Lossless data compression*, http://en.wikipedia.org/wiki/Lossless_data_compression, website last visited on 05/04/2011.