A Software Implementation of the Shannon-Fano Coding Algorithm

Student authors: Đorđe K. Manoilov[1] and Daniel S. Dimitrov[1]
Mentors: Radomir Stanković[2] and Dušan Gajić[2]

Abstract – **The Shannon-Fano coding technique is one of the earliest algorithms which produce code words with minimum redundancy and it serves as a basis for some more recent methods. In this paper, we present a C# implementation of the Shannon-Fano encoding method for data compression. We conducted various experiments with different inputs provided to the application and recorded compression rates and algorithm running times. The presented solution features a graphical user interface and has solid real-world performance, but it was developed primarily as an education tool that can help students to better understand this encoding technique.**

Keywords – **Shannon-Fano encoding, C# programming solution, text compression.**

I. INTRODUCTION

Data compression is a mathematical method - an algorithm used to decrease the number of the bits in a file that are necessary for storage, sending or transferring of electronic information. In other words, by using compression the size of a file or group of files is decreased and space needed for storing the information becomes smaller.

There are some compression methods that loose data, but we will discuss only compression that occurs without loss. The "good" part is that the compressed data will be decompressed in the same form (recovering the data into its initial state), but an error producing even a bit less would be fatal. Compression with no loss can be realized with different algorithms like: RLE (Run Length Encoding) algorithm, algorithm for removing all zeros, Shannon-Fano algorithm, Huffman algorithm [1].

We will discuss the Shannon-Fano compression. For Shannon-Fano compression there is an algorithm which uses prefix coding [1].

In this paper, we will present its implementation and include test results for different textual files [7]. In Section II we describe the theoretical basis of the Shannon-Fano coding. Next, in Section III we present a software solution for data compression using the Shannon-Fano algorithm realized in C# programming language. This application is mainly developed for educational purposes. In the following Section IV we give experimental results for data compression ratio, time and number of different characters. We close the paper with some conclusions in the final section.

Student authors:
[1]Đorđe Manoilov and Daniel Dimitrov are with the Faculty of Electronic Engineering, Aleksandra Medvedeva 14, 18000 Niš, Serbia, E-mails: djordje.manoilov@elfak.rs, ddaniel@elfak.rs.

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

II. SHANNON - FANO ALGORITHM

A. Theoretical basis and the algorithm

Shannon-Fano coding was developed by Claude Elwood Shannon and Robert Fano [1]. This is a technique which uses prefix encoding. It is based on a set of symbols and their probabilities.

A prefix code is a type of a code system which is characterized by a prefix property. This property states that there is no valid code word in the system that is a prefix (start) of any other valid code word in the set. Message can be transmitted as a sequence of concatenated code words, without any extra markers to frame the words in the message using prefix code. The recipient decodes the message by repeating the process searching for prefixes that form valid code words. This is not possible with codes that lack the prefix property. Shannon–Fano coding starts with the set of symbols, with elements arranged in order from most probable to least probable. After that, the set is divided into two sets whose total probabilities are as close as possible to being equal. All symbols then have the first digits of their codes assigned. Symbols in the first set receive "0" and symbols in the second set receive "1". Shannon–Fano coding uses a binary tree structure. As long as any set with more than one member remains, the same process is repeated. When a set has been reduced to one symbol this means that the symbol's code is complete and will not form the prefix of any other symbol's code.

The algorithm produces codes with variable and fairly efficient length. When the two smaller sets produced by partitioning are of exactly equal probabilities, one bit of information used to distinguish them is used most efficiently. It can be seen from the examples that the Shannon – Fano algorithm does not always produce the optimum length codes. For a set of probabilities {0.35, 0.17, 0.17, 0.16, 0:15} Shannon - Fano coding does not give the optimal length code. The Shannon – Fano compression uses binary tree as data structure where the encoded symbols are placed in the leaves of this tree.

The tree is constructed in the specific way in order to define the effective code table. The actual algorithm is simple:

– For a given list of symbols, develop a corresponding list of probabilities or frequency counts, so that each symbol’s relative frequency of occurrence is known.

- Sort the lists of symbols according to frequency, with the most frequently occurring symbols at the left and the least common at the right.
- Divide the list into two parts, with the total frequency counts of the left half being as close to the total of the right as possible.
- The left half of the list is assigned the binary digit 0, and the right half is assigned the digit 1. This means that the codes for the symbols in the first half will all start with 0, and the codes in the second half will all start with 1.
- Recursively apply the steps 3 and 4 to each of the two halves, subdividing groups and adding bits to the codes until each symbol has become a corresponding code leaf on the tree.

B. The field of use

Shannon–Fano coding is used in the `IMPLODE` [2] compression method, which is part of the `ZIP` file format. Huffman algorithm [1] is an improved version of the Shannon – Fano algorithm used to compress music files in MP3 format and for JPEG picture compression [8].

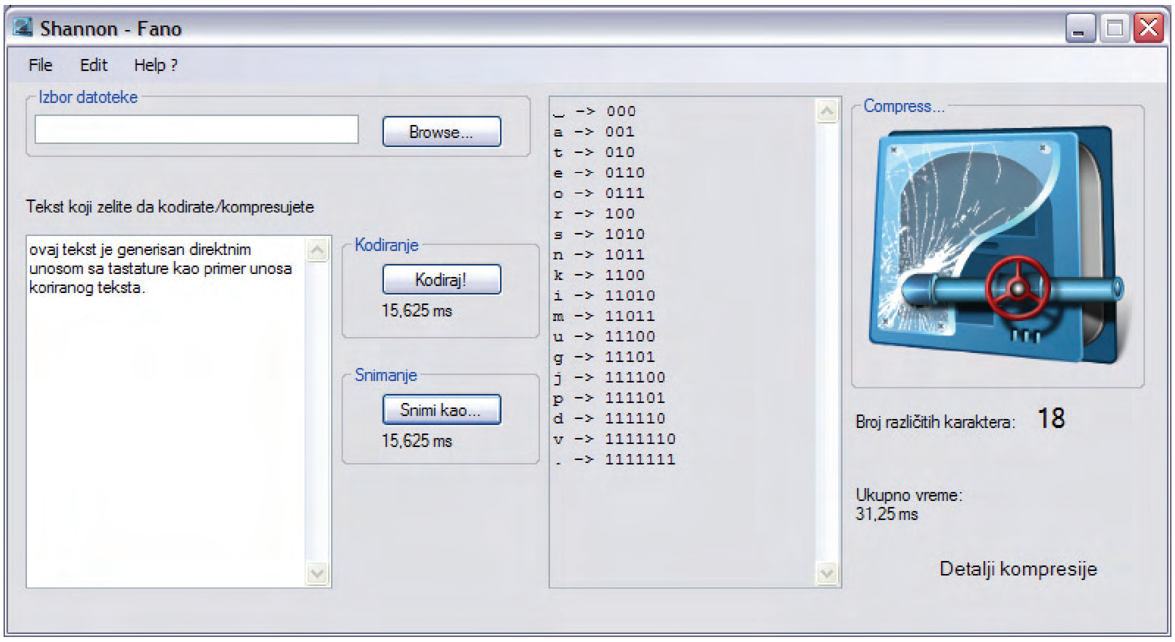
III. Architecture of the Application and the Programming Implementation

The application is developed in Visual C# .NET 3.5 and it can be only used within Microsoft Windows operating system.

The application consists of four forms (Fig. 2.). "Main form" is used for selection of file (for coding) or for manual input of text for coding. Also, on the form "Main form" (Fig. 1.) the symbols and their respectable codes are displayed. It is possible to save coded text on desired location on disk or other medium. "Manual form" offers a brief user manual. In "Statistics form" (Fig. 3.) we can see degree of compression for selected text. "Information form" includes information about authors of the application.

Text that is necessary to compress is placed into a string variable. In the application there is a function for the separation of the different nodes, and also for calculating their probabilities of occurrence. Probability of occurrence of symbols is calculated as the ratio of the number of occurrences of this symbol and the number of symbols in the file. For the purposes of the algorithm, it is necessary to arrange these symbols in ascending or descending order. After sorting, coding of symbols is done by calling the Shannon-Fano algorithm implementation. All symbols in the text change in to their code and all of that put into the new string. Code is replaced with its symbols via library function StringReplace.

C# does not support work on the level of bits. Therefore, before entering into a binary file, the sequence of 8 characters is stored in the buffer, which is the size of a byte. 0 is entered into the buffer by moving the contents of the buffer to the left (shift - left), 1 is entered into the buffer using the shift - left and logical OR operation with 0x01h. This method is possible if the length of the coded text is divisible with 8. It is therefore



Tekst koji zelite da kodirate/kompresujete

ovaj tekst je generisan direktnim unosom sa tastature kao primer unosa koriranog teksta.

15,625 ms

15,625 ms

```
_ -> 000
a -> 001
t -> 010
e -> 0110
o -> 0111
r -> 100
s -> 1010
n -> 1011
k -> 1100
i -> 11010
m -> 11011
u -> 11100
g -> 11101
j -> 111100
p -> 111101
d -> 111110
v -> 1111110
. -> 1111111
```

Broj različitih karaktera: 18

Ukupno vreme: 31,25 ms

Fig. 1. Main form of the application.

necessary to add additional 0 in buffer with last entry in the file. This way leads to an increase in encoded file, for up to 7 bits, but it allows the simulation of work on the level of bits in C#.

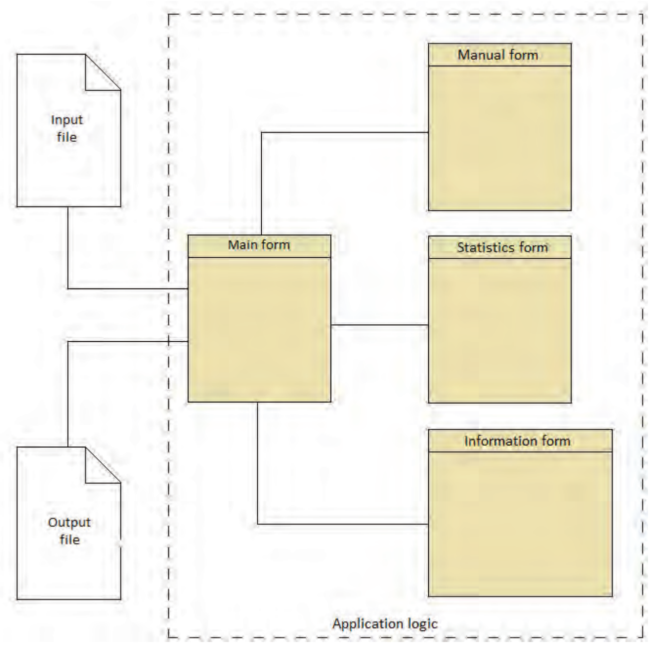
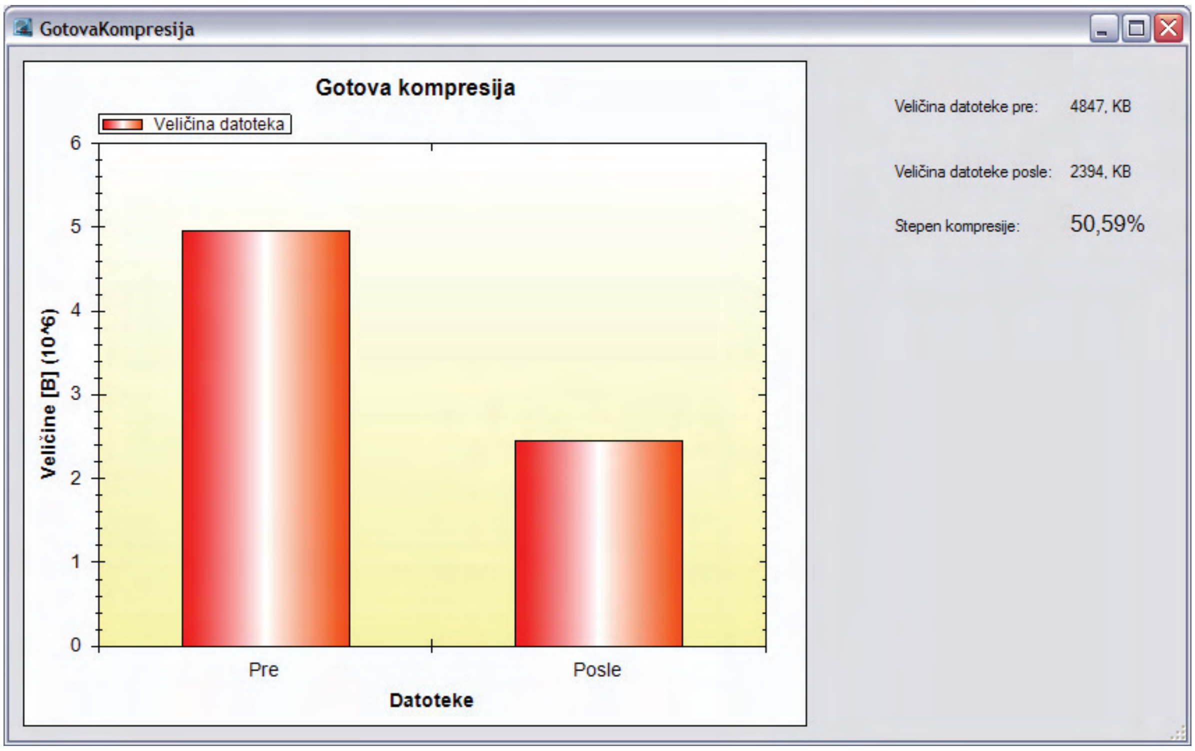


Fig. 2. The architecture of the application.

IV. Experimental Results

The application was tested for various input files in order to get time and percentage of compression. Input file is a text. The all of the experiments were done using a Laptop PC with Intel Core 2 Duo T5450 processor and 3 GBs of RAM, running a Windows XP Service Pack 2 operating. The duration of compression depends on the computer's hardware and current utilization of computer resources. The test results for normal text are shown in Table I, and test results for source code are shown in Table II. Tables I and II also show that the compression speed and compression ratio also depend on the number of different characters and file size. For a small number of different character(s) encoding goes fast regardless of the size of the file. This is because each character encodes a small number of bits and operations with strings quickly completed. For very large files (about 60 MB) the application reports "Memory error". The problem is caused because of the usage of strings and can be solved by using a StringBuilder. Time coding for a normal text file as source code or a book is at most a few seconds. Compression ratio is about 50% but if there are lots of similar character goes up to 80%. From the presented results we can also conclude that the compression ratio of source codes is less than for the plain text.



Veličina datoteke pre: 4847, KB

Veličina datoteke posle: 2394, KB

Stepen kompresije: 50,59%

Fig. 3. Statistics form.

TABLE I

DIFFERENT TEXT FILES, *n* – NUMBER OF DIFFERENT CHARACTERS, t_1 – TIME CODING, t_2 – RECORDING TIME, *fs* – FILES SIZE, *cr* – COMPRESSION RATIO

n	**t_1**	**t_2**	**fs**	**cr**
4	0 ms	15.625	10B -> 3B	70 %
10	0 ms	0 ms	10B -> 5B	50 %
15	0 ms	15.625 ms	21B -> 11B	47.6 %
5	0 ms	15.625 ms	39B -> 12B	69.2 %
47	0 ms	15.625 ms	1,88K -> 1006B	47.7 %
116	656.2 ms	703.12 ms	100,9K -> 49,8K	50.6 %
116	31.9 s	29 s	4847K -> 2394K	50.6 %
3	10.01 s	3.625 s	23523K->4324K	81.8 %
3	27 s	Mem error	61.2M -> ?	?
79	5.112 s	3.718 s	889K -> 501,B	43.26%

TABLE II

SOURCE CODES, *n* – NUMBER OF DIFFERENT CHARACTERS, t_1 – TIME CODING, t_2 – RECORDING TIME, *fs* – FILES SIZE, *cr* – COMPRESSION RATIO

n	**t_1**	**t_2**	**fs**	**cr**
92	807 ms	620ms	126K -> 55,1K	56.2%
95	186 ms	144ms	29,2K -> 14,2K	51.13%
71	44 ms	36ms	8,56K -> 4,55K	46.88%
90	187 ms	118ms	29,7K -> 12,8K	56.99%
90	153 ms	116 ms	21,3K -> 13,2K	38.14%
71	19 ms	17 ms	3,1K -> 1,9K	38,71%
58	5 ms	7 ms	1K -> 647B	41.5%
65	28 ms	22 ms	5,4K -> 3,15K	41.9%
63	13 ms	12 ms	2,3K -> 1,35K	41.61%

V. CONCLUSION

Through performing the experiments with our implementation of the Shannon-Fano algorithm we reached the following conclusions:

– The most common characters have shorter code words and opposite.
– For the same number of different characters, the algorithm has the same compression ratio.
– For two files with the same size, but with different number of unique characters, a file with a smaller number of different characters has a higher compression ratio.
– Time required for recording and encoding increases with the size of the input file.

Application that we have developed cannot actually compete with existing commercial applications that compress data. It was developed primarily as an education tool that can help students to better understand this encoding technique that serves as basis of more recent compression methods.

REFERENCES

[1] David Salomon, *Data Compression: The Complete Reference*, 3rd Edition, Springer, 2004. (ISBN 0-387-40697-2)
[2] *http://en.wikipedia.org/wiki/Shannon%E2%80%93Fano_coding* website last visited on 14/04/2011.
[3] *http://www.ustudy.in/node/6409,* website last visited on 15/12/2010.
[4] *http://www.binaryessence.com/dct/en000041.htm,* website last visited on 14/04/2011.
[5] *http://cppgm.blogspot.com/2008/01/shano-fano-code.html,* website last visited on 14/04/2011.
[6] *http://www.dotnetspark.com/Forum/169-how-to-open-one-chm-help-file-c-sharp-windows.aspx,* website last visited on 14/04/2011.
[7] *http://www.onlinehowto.net/Why-compress-/2,* website last visited on 14/04/2011.
[8] *http://en.wikipedia.org/wiki/Huffman_coding,* website last visited on 14/04/2011.