ISSN:0975-9646

CAPSULE
A Programming Language Code Compression Technique

Govind Prasad Arya, Prince Kashyap, Nilika Kumari, Mitali Hembrom

Department of Computer Science,
Shivalik College of Engineering, Dehradun, Uttarakhand, INDIA.

***Abstract*—In present era internet & large scale scientific research are the most prominent user of large amount of data transmission. Since there is a trade of between efficiency of internet and size of the data being transmitted, so less size data can reach fast and efficiently. Therefore, compressing the data provide a significant change in efficiency of the data transmission. Also it helps in transmission of data efficiently over large scale research in various fields. And on the other side data which is stored on storage disk has some statistical redundancy, which consume unnecessary space and it can be solved by compressing the data.**

***Keywords*—Compression, Decompression, Dictionary, Symbol, Programming Language Code Compression & Decompression, Negative ASCII Characters**

I. INTRODUCTION

COMPRESSION is the reduction in size of data in order to save space and transmission time [Arnavut, 2000]. Compression is possible since there always is some amount of data redundancy or there may be a predictable flow to the data. The size of data is reduced by removing the excessive information. There are two main types of data compression.1- lossless compression [Burrows & Wheeler, 1994; Awan & Mukherjee, 2001], and 2- lossy compression. In lossy compression data once being compressed can't get back to its original form e.g. video compression. If we to reduce the size of a video file by reducing its pixel then it can't be get back to its original form while in lossless compression the compressed data can be get back to its original form e.g. text compression. On the other side when we compress (lossless) a text file it does not have any use if it can't get back to its original form. There are many lossless compression algorithm which we studied e.g. LZ77, LZW, Huffman compression [Balkenhol et al., 1999].

1.1. Shortcomings of other Algorithms

- No special compression technique for programming languages.
- In LZ77, search buffer is thousands of bytes long, while the lookahead buffer is tens of byte long.
- The encryption can be much time consuming due to the larger number of comparison need to be made to find matched pattern.
- Compression techniques have their external dictionary which cause problem while decompressing on another machine.

Other algorithms have thousands of bytes long search buffer, when we check for any character in a string it will go through all the thousands of bytes in the search buffer which is very time consuming, so we advanced our algorithm so that it will just search for some limited no of bytes so that it doesn't have to go for thousands of bytes. It simply matches the character in a search dictionary & replace it with any symbol if no match found then it simply replaces it by itself.

II. METHODOLOGY

2.1. Our Contribution

Since there is no specific algorithm for compressing a programs written in different programming languages. The proposed algorithm compress thousands of lines of codes written in several programming languages, which occupy a large amount of space in memory. It does not generate new output tuple like other algorithms when no match found, rather it replaces the string by itself. And uses an internal dictionary (generated internally while compiling the code). Which removes problem of transmission of codes & programs over the internet.

In this algorithm we have a array in which certain keywords/strings are inserted to find a match for their corresponding symbols(non keyboard) But unlike LZ-77 in this array size is very limited i.e.128 which takes much less time to compare to match .

2.2. Compression and Decompression Process

In this section we show how the compression and decompression is performed using the proposed algorithm.

2.2.1. Compression Process

Input Data: **#include<stdio.h>**

Here in this example input stream is the data which has to be compressed and it has a size of 13 bytes and compression need to be done to make its size less than 11 bytes. Here first characters are checked for alphabet and the string in the text is searched in the internal array for a match. If a match is found in the array then the will be replaced with the corresponding symbol. If not then write the string itself in temporary file. All other characters are written without change in the target file. For example initially when character is ' #', it is written into the target file. Same is the case with '<'. When 'i' is found the characters are inserted into the string and then array is searched for a

match .Since include has a match in the array defined , it is replaced by the corresponding character in the target file. But if it does not find any match, the same string is written into the target file. This process is repeated for all character till the end of file (EOF) is found. The size of target file containing this string is now 9 bytes.

Table 1 – array of keywords/strings

Keywords/Strings	ASCII value of Symbol
include	-2
stdio	-3
printf	-4
scanf	-5
malloc	-6
return	-7
.	.
unsigned	-128

2.2.3. Decompression Process

In this process the compressed code is converted back to its original form.

Here is the compressed data: # ■<ŷ.h>

Now we have to decompress [Cleary, 1995 & Effros 2000] this compressed data into original data. For every symbol we have its corresponding string in the array defined as an internal dictionary. The corresponding string is written back into the target file(decompressed file). The output now is generated by matching each symbol against the corresponding string in the array. All other characters not present in the array get replaced by itself in the decompressed file. Similar process is repeated till EOF. In this way the original data is obtained from the compressed data.

Table 2 – Decompression Process

Symbols	Corresponding String
A	include
B	Stdio
~	Printf
■	Scanf
.	.
Ŷ	unsigned

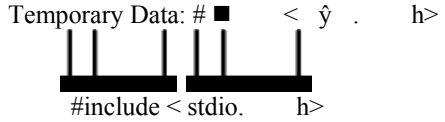

Figure 1 – Generation of Original Data

Now we can see that we have got the data which we had before compression.

2.3 Compression Algorithm

```
COMPRESS (source file)
Open source file & target file
Read one character from source file
While ( EOF)
{
If(character  is an alphabet)
 Declare an empty string;
While (character is an alphabet)
{
   Append the character to string;
   Read next character;
   If(character  is EOF )
    Break;
}
 code=Dictionary(string);
 If(code b/w -2 to -128)
 {
    Write code to target file;
 }
else
   write string to target file;
 }
}
Close source file & target file.

byte DICTIONARY(string)
{
code=1;
Match string from array element;
 Set code=value of array element matched;
Return code; }
```

We can get a compressed file after applying above algorithm.

2.4 Decompression Algorithm

```
DECOMPRESS (source file)
 Open source file & target file
 Read one character from source file
While (EOF)
{
If (character b/w -2 to -128)
{
 Match character from array element;
Write the matched string to target file;
}
else
  Write the character to target file
Read next character ;
  }
  }
Close source file & target file
}
```

We can get a decompressed file after applying above algorithm.

2.5 Advantages of Algorithm

- This algorithm does not generate new tuple when no match found rather it replaces the character by itself.
- Search buffer size in this algorithm is relatively very small so it is not time consuming and is faster than LZ77.
- It has its external dictionary which make it easy to decompress while comparing tuple with external dictionary.

III. CONCLUSION & RESULT

The main aim of compression is to reduce the size of the data so that it will take less space and easily transfer over the internet which reduces the cost. LZ-77 is a good algorithm for compression but still it has some major drawbacks as it take much time to compress while comparing for pattern match, while in our algorithm by adding external dictionary and limited search buffer size we overcome this problem and which make it faster than previous algorithm. As for example taken to describe the process the compression ratio is 45.45%. and the future scope of algorithm can be improved more are by providing threading in program to make program run concurrently [Chapin, 2001] and taking appropriate data structure so dictionary can be accessed fast.

REFERENCES

1. M. Burrows & D.J. Wheeler (1994), "A Block – Sorting Lossless Data compression Algorithm", *SRC Research Report 124,* Digital Research Systems Research Centre.
2. J.G. Cleary, W.J. Teahan & Ian H. Witten (1995), "Unbounded Length Contexts for PPM", *Proceedings of Data Compression Conference, IEEE Computer Society,* Snowbird Utah, Pp. 52– 61.
3. R. Franceschini & A. Mukherjee (1996), "Data Compression using Encrypted Text", *IEEE Proceedings of ADL,* Pp. 130– 138.
4. B. Chapin & S. Tate (1998), "Preprocessing Text to Improve Compression Ratios", *Proceedings of the IEEE Data Compression Conference,* Snowbird, Pp. 532.
5. B. Balkenhol, S. Kurtz & Y.M. Shtarkov (1999), "Modifications of the Burrows Wheeler Data Compression Algorithm", *Proceedings of Data Compression Conference, IEEE Computer Society,* Snowbird Utah, Pp. 188–197.
6. B. Balkenhol & Y. Shtarkov (1999), "One Attempt of a Compression Algorithm using the BWT", *SFB343: Discrete Structures in Mathematics, Faculty of Mathematics, University of Bielefeld,* Germany.
7. Z. Arnavut (2000), "Move-to-Front and Inversion Coding", *Proceedings of Data Compression Conference, IEEE Computer Society,* Snowbird, Utah, Pp. 193–202.
8. M. Effros (2000), "PPM Performance with BWT Complexity: A New Method for Lossless Data Compression", *Proceedings of Data Compression Conference, IEEE Computer Society,* Snowbird Utah, Pp. 203–21 2.

AUTHORS

Govind Prasad Arya has completed Master of Technology (M.Tech) in Computer Science & Engineering from Uttarakhand Technical U niversity, Dehradun (Uttarakhand). He has 8 years of teaching experience from different institutions (Govt. Polytechnic Dwarahat, Kumaon Engineering College Dwarahat). Presently He is working at Shivalik College of Engineering, Dehradun as an Assistant Professor in Computer Science & Engineering Department. Data Structures, Computer Organization, Operating System & Compiler Design are his area of interest. His four research papers on Compression have been published by reputed International Journals. He has attended several Conferences and Seminars.

Prince Kashyap, pursuing his B. Tech, Final year in Computer Science & Engineering from Shivalik College of Engineering. His area of interest is coding and Networking. He has attended several seminar and workshop on cloud computing, Networking, Ethical Hacking conducted by Kyrion ,Ankit Fadia respectively

Nilika Kumari, pursuing her B. Tech, Final year in Computer Science & Engineering from Shivalik College of Engineering. Her area of interest is Software Designing, Data Structure and Networking. She has attended several seminar and workshop on cloud computing, Networking, Ethical Hacking conducted by Microsoft, Kyrion, Ankit Fadia respectively.

Mitali Hembrom, pursuing her B. Tech, Final year in Computer Science & Engineering from Shivalik College of Engineering. Her area of interest is Software Designing, web development Ethical Hacking. She has attended several seminar and workshop on cloud computing and Ethical Hacking conducted by Kyrion, Ankit Fadia respective