RESEARCH ARTICLE OPEN ACCESS

Complexity Analysis of a Lossless Data Compression Algorithm using Fibonacci Sequence

Saptarshi Bhattacharyya
Research Associate
Department Of Computer Science
University Of Calcutta, Kolkata, West Bengal.

ABSTRACT
In spite of the amazing advances in data storage technology, compression techniques are not becoming obsolete, and in fact research in data compression is flourishing. Data compression has been widely applied in many data processing areas. Compression methods use variable-length codes with the shorter codes assigned to symbols or groups of symbols that appear in the data frequently. Fibonacci code, as a representative of these codes, is often utilized for the compression of small numbers. In this work, we concentrate to compressed data using binary representations of integers based on Fibonacci numbers of order $n \geq 2$.

Keywords:- Data, Bit, Nibble, Data Compression, Lossless Compression, Lossy Compression, Fibonacci Sequence, Complexity Analysis, Big-Oh Notation, O(1), O(N), $O(N^2)$, $O(2^n)$,Recursion.

I. INTRODUCTION

Data (data is the plural of *datum*) is distinct pieces of information, usually formatted in a special way. All software is divided into two general categories: *data* and *programs*. Programs are collections of instructions for manipulating data. Data can exist in a variety of forms -- as numbers or text on pieces of paper, as bits and bytes stored in electronic memory.

A bit (short for binary digit) is the smallest unit of data in a computer. A bit has a single binary value, either 0 or 1. Although computers usually provide instructions that can test and manipulate bits, they generally are designed to store data and execute instructions in bit multiples called bytes. There are eight bits in a byte. Half a byte (four bits) is called a nibble. In some systems, the term octet is used for an eight-bit unit instead of byte. In many systems, four eight-bit bytes or octets form a 32-bit word. In such systems, instruction lengths are sometimes expressed as full-word (32 bits in length) or half-word (16 bits in length).

What is Data Compression?

Data compression defined as the representation of data in such a way that, the storage area needed for target data is less than that of, the size of the input data. In compression techniques or compression algorithm, we are actually referring to two algorithms. There is the compression algorithm that takes an input and generates a representation X that requires fewer bits, and there is a reconstruction algorithm that operates on the compressed representation X to generate the reconstruction.

We will follow convention and refer to both the compression and reconstruction algorithms together to mean the compression algorithm. Based on the requirements of reconstruction, data compression schemes can be divided into two broad classes: lossless compression schemes, in which is identical to, and lossy compression schemes, which generally provide much higher compression than lossless compression but allow to be different from previous.

Lossless compression techniques, as their name implies, involve no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless compression is generally used for applications that cannot tolerate any difference between the original and reconstructed data. Text compression is an important area for lossless compression. It is very important that the reconstruction is identical to the text original, as very small differences can result in statements with very different meanings. Consider the sentences “a faint scent of roses” and “a faint cent of roses” A similar argument holds for computer files and for certain types of data such as bank records. Lossy Compression Lossy compression techniques involve some loss of information, and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. In return for accepting this distortion in the reconstruction, we can generally obtain much higher compression ratios than is possible with lossless compression.

The data compression used in plenty of data processing areas. In ASCII code, we have 256 characters represented by the different numbers. Ex. ‘A' represented, numerically as 65. The frequency of each ASCII code differs from each other. If text data used as an input, some characters

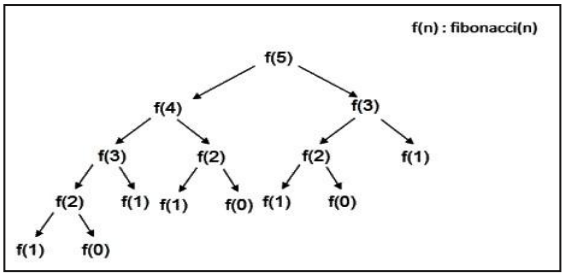
occur most frequently, and many characters never utilized in the input. The alphabet, digits and some special characters used mostly. In the alphabet, the most used characters are vowels. The 256 symbols never used frequently. The variations in this frequency of characters need data compression. The fixed length code, need to be replaced by variable length code.

II. FIBONACCI CODE: A BRIEF IDEA

Fibonacci coding generates variable length codes. It uses traditional methods of replacing input characters by specific code like code words. It uses Fibonacci series to implement the compression.

Fig 1: Recursion Tree Generated for computing 5th number of Fibonacci sequence

In mathematics, the Fibonacci numbers are the



numbers in the following integer sequence, called the Fibonacci sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones: 1;1;2;3;5;8;13;21;34;55;89;144;

Often, especially in modern usage, the sequence is extended by one more initial term: 0;1;1;2;3;5;8;13;21;34;55;89;144; In mathematical terms, the sequence F_n of Fibonacci numbers is defined by the recurrence relation $F_n = F_{n-1} + F_{n-2}$

Where $F_1 = 1$ & $F_2 = 1$ Or $F_0 = 0$ & $F_1 = 1$

The Fibonacci sequence is named after Italian mathematician Leonardo of Pisa, known as Fibonacci.

FIBONACCI SEQUENCE

F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}
0	1	1	2	3	5	8	13	21	34	55	89	144

Table 1: Fibonacci Sequence.

The Fibonacci series used to create Fibonacci code.

III. EFFICIENCY OF AN ALGORITHM

The *complexity* of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process. Usually there are natural units for the domain and range of this function. There are two main complexity measures of the efficiency of an algorithm:

- *Time complexity* is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. "Time" can mean the number of memory accesses performed, the number of comparisons between integers, the number of times some inner loop is executed, or some other natural unit related to the amount of real time the algorithm will take.
- *Space complexity* is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm. We often speak of "extra" memory needed, not counting the memory needed to store the input itself. Again, we use natural (but fixed-length) units to measure this. We can use bytes, but it's easier to use, say, number of integers used, number of fixed-sized structures, etc. In the end, the function we come up with will be independent of the actual number of bytes needed to represent the unit. Space complexity is sometimes ignored because the space used is minimal and/or obvious, but sometimes it becomes as important an issue as time.

What is big-O notation?

A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items.

Informally, saying some equation f(n) = O(g(n)) means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n".

f(n) = O(g(n)) means there are positive constants c and k, such that $0 \le f(n) \le cg(n)$ for all $n \ge k$.

The values of c and k must be fixed for the function f and must not depend on n.

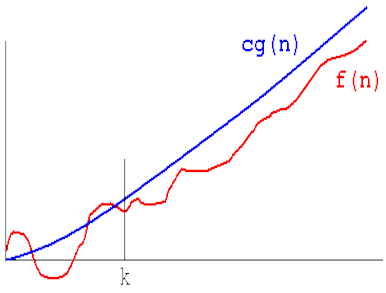


Fig 2: Big-O notation

- **O(1)**

O(1) describes an algorithm that will always execute in the same time (or space) regardless of the size of the input data set.

- **O(N)**

O(N) describes an algorithm whose performance will grow linearly and in direct proportion to the size of the input data set. A matching string could be found during any iteration of the for loop and the function would return early, but Big O notation will always assume the upper limit where the algorithm will perform the maximum number of iterations.

➢ **$O(N^2)$**

$O(N^2)$ represents an algorithm whose performance is directly proportional to the square of the size of the input data set. This is common with algorithms that involve nested iterations over the data set. Deeper nested iterations will result in $O(N^3)$, $O(N^4)$ etc.

➢ **$O(2^N)$**

$O(2^N)$ denotes an algorithm whose growth doubles with each additon to the input data set. The growth curve of an $O(2^N)$ function is exponential - starting off very shallow, then rising meteorically. An example of an $O(2^N)$ function is the recursive calculation of Fibonacci numbers.

IV. FIBONACCI ENCODING

An efficient self-delimited representation of arbitrary numbers is the Fibonacci encoding. The

Position	1	2	3	4	5	6	…			10	Rank's Fibonacci Representation	Rank_Fibo_Code= Rank's Fibonacci Representation + {suffix 1}
Fibonacci_Value F(n)	1	2	3	5	8	13	…			89		
Rank												
1	1										1	11
2	0	1									01	011
3	0	0	1								001	0011
4	1	0	1								101	1011
5	0	0	0	1							0001	00011
6	1	0	0	1							1001	10011
7	0	1	0	1							0101	01011
8	0	0	0	0	1						00001	000011
9	1	0	0	0	1						10001	100011
10	0	1	0	0	1						01001	010011
.	.	.	.	.	.	.	.			.	.	.
100	0	0	1	0	1	0	0			1	00101010001	00101000011

Fibonacci encoding is based on the fact that any positive integer n can be uniquely expressed as the sum of the distinct Fibonacci numbers, so that no two consecutive Fibonacci numbers are used in the representation. This means that if we use binary representation of a Fibonacci encoding of a number, there are no two consecutive 1 bits.

Input: A positive integer n

Initialization: S = {}

Algorithm:

1. Let i be the largest number such that F(i) <= n
2. Add i to S
3. Replace n with n-F(i)
4. Go To 1

Table 2: Fibonacci Code Value

So, by adding a 1 after the 1 corresponding to the biggest Fibonacci number in the sum, the representation becomes self-delimited.

COMPRESSION TECHNIQUE

The compression process is performed in clearly separated stages:

- Read input file and determine character frequencies.
- Sort the characters by the frequencies in descending order.
- Compute the Fibonacci code of each ranking.
- Output the ranking as the header of the compressed file.
- Reread the input file, using the code table to generate output to the compressed file.

DECOMPRESSION TECHNIQUE

The compression process is performed in clearly separated stages:

- Read compressed file data.
- The decoding process read the input, character by character until the number 11 found.
- Then the entire binary code decoded to corresponding unique character.

V. COMPLEXITY ANALYSIS OF FIBONACCI ENCODED ALGORITHM

Complexity Analysis of Fibonacci Sequence:

```
int Fibonacci(int n
{
   if (n <= 1)
      return n;
   else
      return Fibonacci(n - 1) + Fibonacci(n - 2);
}
```

In general, total number of states are approximately equal to 2^N for computing n^{th} Fibonacci number (F(N)). Notice that each state denotes a function call to 'Fibonacci ()' which does nothing but to make another recursive call. Therefore total time taken to compute nth number of Fibonacci sequence is **$O(2^N)$.**

In this Fibonacci encoded compression algorithm method for sorting purpose we use bubble sort.

Complexity Analysis of Bubble Sort:

Bubble Sort is an algorithm which is used to sort N elements that are given in a memory for e.g.: an Array with N number of elements. Bubble Sort compares the entire element one by one and sort them based on their values.

It is called Bubble sort, because with each iteration the smaller element in the list bubbles up towards the first place, just like a water bubble rises up to the water surface.

```
begin BubbleSort(list)
   for all elements of list
      if list[i] > list[i+1]
         swap(list[i], list[i+1])
      end if
   end for
   return list
end BubbleSort
```

```
begin swap(x,y)
        x = x + y;
        y = x - y;
        x = x - y;
end swap
```

Table 4: Algorithm for Bubble Sort

In Bubble Sort, N-1 comparisons will be done in 1st pass, N-2 in 2nd pass, N-3 in 3rd pass and so on. So the total number of comparisons will be (N-1)+ (N-2) + (N-3) +.....+3+2+1

Sum = $\frac{N(N-1)}{2}$ I.e. O (N^2)

Hence the complexity of Bubble Sort is O (N^2).

Best-case Time Complexity will be O (N); it is when the list is already sorted.

We assume list is an array of n elements. We further assume that swap function swaps the values of the given array elements.

The complexity of any comparison sort is O (N^2).

The complexity of Fibonacci encoded compression technique is O (N^2).

VI. SAMPLE INPUT

Mr. Saptarshi Bhattacharyya presented this paper in NCCCI conference at TGRI, Tamil Nadu, Chennai on 27.01.2017.

Char	Frequency	Rank	Rank's Fibonacci Representation	Rank's_Fibo_Code	Probability	Code Word Length	Code Word Total Bits
	15	1	1	11	0.1339286	2	30
a	11	2	01	011	0.0982143	3	33
e	8	3	001	0011	0.0714286	4	32
n	7	4	101	1011	0.0625	4	28
t	6	5	0001	00011	0.0535714	5	30
r	6	6	1001	10011	0.0535714	5	30
i	5	7	0101	01011	0.0446429	5	25
h	5	8	00001	000011	0.0446429	6	30
.	4	9	10001	100011	0.0357143	6	24
p	4	10	01001	010011	0.0357143	6	24
C	4	11	00101	001011	0.0357143	6	24
s	3	12	10101	101011	0.0267857	6	18
c	3	13	000001	0000011	0.0267857	7	21
0	2	14	100001	1000011	0.0178571	7	14
7	2	15	010001	0100011	0.0178571	7	14
2	2	16	001001	0010011	0.0178571	7	14
1	2	17	101001	1010011	0.0178571	7	14
,	2	18	000101	0001011	0.0178571	7	14
y	2	19	100101	1001011	0.0178571	7	14
o	2	20	010101	0101011	0.0178571	7	14
d	2	21	0000001	00000011	0.0178571	8	16
N	2	22	1000001	10000011	0.0178571	8	16
I	2	23	0100001	01000011	0.0178571	8	16
T	2	24	0010001	00100011	0.0178571	8	16
u	1	25	1010001	10100011	0.0089286	8	8
m	1	26	0001001	00010011	0.0089286	8	8
l	1	27	1001001	10010011	0.0089286	8	8
f	1	28	0101001	01010011	0.0089286	8	8
R	1	29	0000101	00001011	0.0089286	8	8
G	1	30	1000101	10001011	0.0089286	8	8
B	1	31	0100101	01001011	0.0089286	8	8

S	1	32	0010101	00101011	0.0089286	8	8
M	1	33	1010101	10101011	0.0089286	8	8

Table 5: Fibonacci code representation of frequency of characters of an input.

COMPRESSED ENCODED INPUT DATA

1010101110011100011110010101101101001100011011100111010110000110101111010010110000110110001100011011000001100001101111
110010001110001011000010110100001100010111100100011011000100110101110010011111000001101100000011101000110001011110010

Table 6: Compressed Encoded Input Data

VII. COMPRESSION & DECOMPRESSION

The given input ***Mr. Saptarshi Bhattacharyya presented this paper in NCCCI conference at TGRI, Tamil Nadu, Chennai on 27.01.2017.*** after encoding will be

10101011100111000111100101011011010011000110111001110101100001101011110100101100001101100011000110110000011000011011100111001011100101101111010011100110011101011001110110001100110000001111000110000110101110101111010011011010011001110011110101110111110000011001011001011001011010000111100000011010101110110101001100111001100111011000000110011110110001111001000111000101100001011010000110001011110010001101100010011010111001001111110000011011000000111010001100010111100101100001100111011101101101011110101011101111001001101000111000111000011101001110001100100111000011101001101000111000011

During Decompression

After Step 1:

M1001110001111001010110110100110001101110011101011000011010111101001011000011011000110001101100000110000110111001110010111001011011110100111001100111010110011101100011001100000001111000110000110101110101111010011011010011001110011110101110111110000011001011001011001011010000111100000011010101110110101001100111001100111011000000110011110110001111001000111000101100001011010000110001011110010001101100010011010111001001111110000011011000000111010001100010111100101100001100111011101101101011110101011101111001001101000111000111000011101001110001100100111000011101001101000111000011

After Step 58:

Mr. Saptarshi Bhattacharyya presented this paper in NCCCI 0000011010101110110101001100111001100111011000001100111101100011110010001110001011000010110100001100010111100100011011000100110101110010011111000001101100000011101000110001011110010110000110011101110110110101111010101110111100100110100011100011100001110100111000110010011100001110100110100011100011

After Step 100:

Mr. Saptarshi Bhattacharyya presented this paper in NCCCI conference at TGRI, Tamil Nadu, Chennai on 110010011010001110001110000111010011100011001001110000111010011010001110001

After Step 112:

Mr. Saptarshi Bhattacharyya presented this paper in NCCCI conference at TGRI, Tamil Nadu, Chennai on 27.01.2017.

VIII. RESULTS AND DISCUSSION

A compression algorithm can be evaluated in a number of different ways. We could measure the relative complexity of the algorithm, the memory required to implement the algorithm, how fast the algorithm performs on a given machine, the amount of compression, and how closely the reconstruction resembles the original. In this paper, I will mainly be concerned with the last one criteria. A very logical way of measuring how well a compression algorithm compresses a given set of data is to look at the ratio of the number of bits required to represent the data before compression to the number of bits required to represent the data after compression. This ratio is called the compression ratio

The size of the input as uncompressed: = (112*8)
= 896 bits.

The total number of bits required after compression:
=(30+33+32+28+30+30+25+30+24+24+24+18+21+14+14+14+14+14+14+14+16+16+16+16+8+8+8+8+8+8+8+8+8) = 583 bits.

The compression ratio, space savings and average bits calculated for input data set:

Compression ratio= 583:896
= 0.650669642:1

Space Savings: =1-(583/896)
=0.349330357
=34.93%

Average bits: =583/112
=5.205357143 bits per character.

IX. CONCLUSION

The Fibonacci coding is a data compression technique that based on Fibonacci series. It produces static variable length code for representing the data. The Shannon-Fano coding and Huffman Coding provides dynamic variable length code. Both, the fewer amount and more amount input data produces good compression ratio, space savings, and average bits per character. But for less number of input characters with more probability, Fibonacci coding gives better compression ratio, space savings, and average bits per character. In this paper, I have compressed the

text type data. My current research work is going on to compressed the different types of data.

REFERENCES

[1] A. Apostolico and A. Fraenkel. Robust Transmission of Unbounded Strings Using Fibonacci Representations. IEEE Transactions on Information Theory, 33(2):238– 245, 1987.

[2] C. E. Shannon, "A Mathematical Theory of Communication", Bell System Technical Journal, Volume 27, pp. 398-403, 1948.

[3] R.M. Fano, "The transmission of information", Technical Report 65, Research Laboratory of Electronics, M.I.T, Cambridge, Mass, 1949.

[4] Leonardo of Pisa (known as Fibonacci). Liber Abaci. 1202.

[5] D.A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes", Proceedings of the I.R.E, pp. 1098–1102, 1952.

[6] P. Elias, "Universal Codeword Sets and Representations of Integers," IEEE Trans. Inform. Theory, IT-21, pp. 194-203, 1975.

[7] Sigler, L.E. (2002) Fibonacci's Liber Abaci. Springer, New York.