TEXT DATABASE COMPRESSION USING REPLACEMENT AND BIT REDUCTION

Anish Kumar[1], Sk Sakir Ali[1], Debashis Chakraborty[2]

[1] Department of Information Technology,
St. Thomas' College of Engineering & Technology, Kolkata, West Bengal
anish_kumar05@ymail.com, skrali03@gmail.com

[2] Assistant Professor
Department of Computer Science & Engineering,
St. Thomas' College of Engineering & Technology, Kolkata, West Bengal
sunnydeba@gmail.com

ABSTRACT

This is a simple compression algorithm which is based on repetition of words and number system theory as well. It employs a technique in which frequently occurring words are replaced by special characters and the modified file is considered as n-base number system, where n is the number of different characters in the file. Further, compression process is carried out by converting this n-base number system to binary number system. The main idea behind using this algorithm is to represent the whole data into lower number system thereby saving bits requirement. It is a simple compression and decompression technique which can be widely used on database as database contains frequently occurring words like last name etc.

KEYWORDS

Number system, Bit saving, Bit loss.

1. INTRODUCTION

Data compression, often termed as source coding, is a process of representing or say encoding information using lesser number of bits than an unencoded representation would use. It follows from the fact that the receiver of the data must be aware of the encoding scheme used by sender and is capable of decoding it to retrieve the original form of data. The main reason behind compression is to reduce the storage space required to store the data, reduce the bandwidth requirement in order to transmit it, thereby reducing total cost. Although a large storage space is available for storing the data but it can go beyond the transmission capacity. Data compression schemes may be classified under two topics – 1. Lossless compression and 2. Lossy compression. Lossless compression algorithms usually exploit statistical redundancy in such a way that the sender's data can be represented more concisely without error. Lossless compression is possible because most of the real world data has statistical redundancy. In lossless compression technique, data loss is unacceptable. Original data can be reconstructed from the compressed one. Another compression technique, called lossy data compression is possible if minor data loss is acceptable.

Natarajan Meghanathan, et al. (Eds): SIPM, FCST, ITCA, WSE, ACSIT, CS & IT 06, pp. 407–416, 2012.
 DOI : 10.5121/csit.2012.2340

In this case, original data cannot be reconstructed from the compressed data due to the removal of some redundant data while compression process [10,11,12]. The data compression can be made more efficient by the hybridization of different algorithms. The main advantage of such technique is that it can compress the output file which is produced after applying certain compression techniques on a file. This yields a better result.

2. ALGORITHM STRATEGY

First we need to read all the words in the input file and count the occurrence of each word. Then sort the words according to the order of their occurrence in descending order and store them in a file. If any word appears more than twice, replace the same using a special character and then modify the original file by replacing those words with their respective assigned special characters. Now we need to count the number of different characters from the modified file and store them in a sequential file. If the total no of distinct characters is n, then consider the whole file as an n base number system where each character is one of the elements of this n base number system. Now, assign index numbers to the distinct characters present in the sequential file from 0 to (n-1). Pick 'g' number of characters from the modified file at a time and then by taking the index number of each character calculate it's decimal value according to number system theory. Represent this decimal value by 's' numbers of bits in binary format to get the compressed file. There will be a proper relation between 'g' and 's' to achieve better compression ratio.

3. CALCULATION

First read the input file and count the number of distinct characters.

Let the number of distinct characters be 'n'.

Consider the whole file as an n base number system.

Index each distinct character from 0 to (n-1).

Take 'g' number of elements from this n base number system file at a time.

Then the maximum value of this 'g' number of elements will be :

Max_Val = (n-1)*n^0 + (n-1)*n^1 + (n-1)*n^2 + ……..... + (n-1)*n^(g-1)

(This is case when in 'g' number of elements, all the elements with maximum index number appears.)

Max_Val = (n-1)*[n^0 + n^1 + n^2 +……+n^(g-2) + n^(g-1)]

= (n-1)*(n^g-1)/(n-1)

= (n^g-1)

Similarly, if we want to denote these elements in binary format, the maximum binary value will be (2^s-1) where 's' is the maximum number of bits required to store the elements in binary format.

Now if we transfer the n base number system to binary number system, then we can say,

=> n^g-1 = 2^s-1

=> n^g = 2^s

Taking log(e) i.e. ln both the sides, we'll get,

=> ln(n^g) = ln(2^s)

=> g*ln(n) = s*ln(2)

=> s/g = ln(n)/ln(2) = 1.443*ln(n)

=> s = 1.443*ln(n)*g

Now, if we consider a file as a number system where each character is an element of this system then actually g bytes can be replaced by s bits. So, we can say 8g bits will be replaced by s no. of bits. Now, we can say the %compression = (1-s/(8g))*100 % = (1-0.41525*log(n))*100 % where n is the no. of different types of characters in the file.

This way, we can convert the text file of any number system format into a format that will require lesser number of bits for storage.

4. ALGORITHM

4.1 Steps for Compression

1. Begin
2. Read all the words in the input file.
3. Initialize a counter for each word.
4. Sort the words according to the number of their occurrences in descending order.
5. If a word appears more than twice, replace it with a special character (ASCII range 128-254) and maintain a dictionary or replaced words in an array Special [ ].
6. If the count of special characters used reaches 127, combine two special characters to replace the further words.
7. Now again read all the strings from this modified input file.
8. Store the distinct characters in an array Distinct [ ].
9. Find out the number of distinct characters in this modified file.
10. Count the number of distinct characters. Let it be ‘n’. So consider the modified file as a n-base number system.
11. Select an integer ‘g’.
12. Define a variable s=ceiling value of (1.443 * ln(n) * g).
13. Do until End of file.
14. Pick up ‘g’ number of characters from the modified file at a time.
15. Find the position of each picked up character within the array Distinct [ ] and store the positional value into an another array pos [ ]. Let pos [ ] = {a0,a1,a2,…..,a(g-1)}.
16. Pos_val = a0*(n^0) + a1*(n^1) + ……..+ a(g-1)*(n^(g-1)).
17. Now, represent the pos_val with ‘s’ number of bits in the binary system. Put these bits into a new file Result.
18. Repeat the process from step 13.

19. End of process.
20. Result file is the final compressed file.
21. End.

4.2. Steps for Decompression

1. Begin
2. Do until end of compressed string or File.
3. Take s no. of bits at a time and then calculate its decimal value.
4. Let the decimal value is val.
5. Divide val by n , until we store g no. of remainder in an array Rem[ ]. Let Rem[ ]={ c0, c1, c2, …..,cg-2, cg-1}.
6. Now map the each element of the array Rem[ ] with the array Distinct[ ] which is mentioned in Compression routine.
7. Let Rem[ ]={c0, c1, c2, …..,cg-2, cg-1}.Then put the Distinct[c0], Distinct [c1],…. Distinct[cg-1] into a file Extract.
8. Go to step 2
9. End of Loop
10. Now map each special character appearing in the file Extract from the dictionary Special [ ] mentioned in the Compression routine and replace them to retrieve the final Decompressed file.
11. Store the final results into a file Get.
12. End

5. ALGORITHM ILLUSTRATION

5.1 Compression

Consider a text file written :-

“ My name is Johan. Johan is a good boy. Johan lives in Kolkata.”

Here ‘is’ and ‘Johan’ appears twice so these words will be replaced by some special characters say ‘$’ and ‘#’.

Then the modified file will become like this :-

“ My name $ #.# $ a good boy.# lives in Kolkata.”

Now, the count of total number of characters in this modified file is 45 but the distinct character set is [M, y, n, a, m, e, $, #,., g, o, d, b, l, v, s, K, k, t, ,] which can be indexed as :-

{M=0, y=1, n=2, a=3, m=4, e=5, $=6, #=7, .=8, g=9, o=10, d=11, b=12, l=13, v=14, s=15, K=16, k=17, t=18 and ‘space’=19}

Total number of distinct characters is 20, so take n = 20 and g = 2.

So, $s = 1.443 * \ln(20) * 2 = 8.64568$

Take the ceiling value of s i.e. 9.

Now take 2 characters from the modified file at a time and calculate the decimal value of the index in order to convert them into equivalent binary form.

The first 3 characters are {M, y} and the respective index numbers are {0, 1}.

Calculating this in decimal value, the result will be :-

0*20^0 + 1*20^1 = 20

Again convert 20 into binary to get a 9 bit value i.e. $(000010100)_2$.

Put this binary value in the modified file at the place of My.

Continue this process by extracting g = 2 characters at a time and converting them to it's equivalent binary form until the end of file.

So by representing in this way, the total size of the compressed file will be (10*9) bits = 90 bits whereas the original size of the text file was (60*8) bits = 480 bits.

Hence, a compression percentage of around 81.25% is achieved.

This way, the whole file can be compressed thereby achieving a better compression ratio.

5.2 Decompression

In decompression process, convert the compressed binary bit string into its equivalent decimal value

So, $(000010100)_2 = (20)_{10}$

Again, convert 20 in n base number system, then we'll get {1,0} and on reversing it, we'll get {0,1}. After mapping this index numbers with the characters, the original text 'My' will be retrieved.

This way, the whole file can be decompressed to retrieve the whole original file without any loss of information.

6. ALGORITHM DISCUSSION

Table 1. Compression of files

Original File	Original size	No. of records, attributes	LZW	Huffman	Proposed
S1..xml	7.66KB	294, 4	4.04kb (47.25%)	5kb (34.72%)	4.12kb (46.21%)
S2.txt	14.90KB	98, 14	5.74 KB (61.48%)	9.94 KB (33.30%)	5.72 KB (61.61%)
S3.txt	178 KB	550,2	35 KB (80.34%)	111 KB (37.64%)	65.5 KB (63.2%)
S4.xml	2.15MB	5683, 59	936kb (57.48%)	1.39mb (35.34%)	1.04mb (51.63%)

S5.xml	2.48MB	5683, 4	1.12mb (54.83%)	1.66mb (33.06%)	1.21mb (51.2%)
S6.xml	3.53MB	74123, 4	702kb (80.58%)	2.16mb (38.81%)	1.23mb (65.15%)
S7.xml	7.28MB	65533, 11	4.98mb (31.59%)	5.04mb (30.76%)	2.73mb (62.5%)

Here we can see the comparison between the compression percentages achieved after applying different compression techniques on the different files. The result shows that LZW and Huffman gives varying compression percentages but our proposed algorithm gives an average compression percentage of around 60%. The result will be better if the repetition of words in a file will be more. So we can say that our proposed algorithm gives an average result and can be used instead of other algorithms to get a better compression percentage.

6.1. Characters Set and Compression Ratio

The above mentioned table just also shows how the compression ratio varies with size of characters set. It gives better compression for some files because they contain repetitive terms. Here we can find some similarity with other well known statistical data compression techniques like Huffman algorithm.

6.2 Experimental result and Theoretical result

It has been observed that the experimental results are almost consistent with the theoretical results.

6.3 Selection of g and s in the Compression Algorithm

To get better compression, selection of proper g and s in the compression algorithm is very necessary. Actually we are taking ceiling value for s in the compression algorithm. For that reason there appears a certain fractional bit loss which reduces a certain compression gain. An algorithm is written below for this selection process where the system can allocate maximum m no. of bits:

Selection Algorithm:

1. First, set g = 1 and r = +9999 (large value) Calculate s = ceiling value of (1.443*ln(n)*g) where n is the total number of distinct characters present in the file till (g<=n)
2. Store the value of s/g in a variable x (x=s/g)
3. If (r>x)

   select_s = s

   select_g = g
4. End of if statement
5. Then store the value s/g in r i.e. r = s/g
6. Increase g by 1 i.e. g = g+1
7. End of while

8. select_s and select_g are the required value of s and g.

Illustrative Example of above Algorithm:

Table 2. Finding the value of s/g

Taking the value of n=50

g	s=1.443*ln(n)*g	s = Ceiling Value (s)	s/g
1	5.645	6	6
2	11.28	12	6
3	16.93	17	5.67
4	22.57	23	5.75
5	28.22	29	5.80
6	33.86	34	5.67
7	39.51	40	5.71
8	45.15	46	5.75
9	50.80	51	5.67
10	56.44	57	5.70
11	62.08	63	5.72
12	67.73	68	5.67

So, here the g=3,g=6,g=9 will be selected for the calculation of s . From the %Compression calculation, we can say when the s/g is minimum then %Compression is increased. According to the previous calculation table for g=3;g=6;g=9 , the s/g ratio is minimum.

6.4 Bit Loss

Bit can not be fractional value. Here we are taking the ceiling value of s in the compression routine. This increases certain amount of fractional bit value which is more than the required bit value. Actually we loss certain fractional amount of bits from there usage. e.g. Let value of s=7.23 ,but we have to consider s=8. So a fractional amount (8-7.23)=0.77 bit can not be used though we have to take it into account. Here it is a compulsory loss for the rounding of . 0.77 is the bit loss amount in the example.

6.5. Testing with Sample Data

Table 3. Original file

Name	Dept	Zipcode	City	State
Anish Kumar	Information Technology	360021	Patna	Bihar
Sumit Kumar	Computer Science	360021	Patna	Bihar
Gaurav Singh	Information	410039	Jaipur	Rajasthan

	Technology			
Sumit Yadav	Computer Science	700003	Kolkata	West Bengal
Arka Ghosh	Information Technology	700001	Kolkata	West Bengal

Table 4. Modified file

Name	**Dept**	**Zipcode**	**City**	**State**
Anish £	∑	Ô	Њ	♣
≠ £	♫	Ô	Њ	♣
Gaurav Singh	∑	410039	Jaipur	Rajasthan
≠ Yadav	♫	700003	╪	☻
Arka Ghosh	∑	700001	╪	☻

Number of distinct characters, n = 39

Take g = 2

So, s = 1.443*ln(39)*3 = 15.8596

Ceiling value of s = 16

So after picking g=3 characters at a time, we will denote it in

s =16 bits in binary format thereby reducing the total number

of bit requirement to store the file.

Result:

Original file size: 288bytes

Modified file size: 154bytes

Compressed file size: 105bytes

6.6. Query Processing

In query writing,

First compress the desired field or attribute. Then place the compressed form of name of attribute(s) (which is wanted to be extracted) in the query.

Then the result will be the required column(s) or particular field from the compressed file.

Finally, decompress the extracted part of the compressed file to view the desired result.

In this way. The required field(s) or attributes(s) can be extracted from the compressed file without decompressing the whole file.

7. CONCLUSION

In this paper we have introduced a new compression algorithm to reduce the size of the text files. The technique of 'saving bits' is employed in this algorithm. The result suggests that the compression ratio is higher for files having more number of repetitive words. Repetitive words are represented by less number of bits. The algorithm has very specific goals and we tried to achieve them to the best of our ability. We have downloaded the database of varying size for testing from a website http://www.medicare.gov/download/downloaddb.asp.

ACKNOWLEDGEMENT

First, we would like to thank Professor Subarna Bhattacharjee[4] and our project evaluation committee, for their valuable advice, support and constant encouragement. Their constant criticisms and reviews gave us the conceptual clarity. We owe a significant lot to all faculty members of the Department of Computer Science and Engineering and the Department of Information Technology. We are also thankful to our seniors who helped us a lot in the development of this paper. It was in one such class of Design and Analysis of Algorithms that we first envisioned the above algorithm. We would also like to thank our friends for patiently enduring our explanations. Their reviews and comments were extremely helpful. And of course, we owe our ability to complete this project to our families whose love and encouragement has been our cornerstone.

REFERENCES

[1]J.Ziv and A. Lempel, "Compression of individual sequences via variable length coding", IEEE Transaction on InformationTheory ,Vol 24: pp. 530 – 536, 1978.

[2]J.Ziv and A. Lempel, "A universal algorithm for sequential data compression", IEEE Transaction on InformationTheory , Vol 23: pp.337 – 343, May 1977.

[3]Gonzalo Navarro and Mathieu A Raffinot, "General Practical Approach to Pattern Matchingover Ziv-LempelCompressed Text",Proc. CPM'99, LNCS 1645,Pages14-36.

[4]S. Bhattacharjee, J. Bhattacharya, U. Raghavendra, D.Saha, P. Pal Chaudhuri, "A VLSI architecture for cellular automata based parallelCdata compression", IEEE-2006,Bangalore, India, Jan 03-06.

[5]Khalid Sayood, "An Introduction to Data Compression", Academic Press,1996.

[6]David Solomon, " Data Compression:The Complete Reference", SpringerPublication,2000.

[7]M. Atallah and Y. Genin, "Pattern matching text compression: Algorithmicand empirical results", International Conference on Data Compression, vol II: pp. 349-352,Lausanne,1996.

[8]Mark Nelson and Jean-Loup Gaily, " The Data Compression Book", Second Edition, M&T Books.

[9]Timothy C. Bell, "Text Compression", Prentice Hall Publishers,1990.

[10]Ranjan Parekh," Principles of Multimedia", Tata McGraw-Hill Companies,2006.

[11]Amiya Halder, Sourav Dey, Soumyodeep Mukherjee and Ayan Banerjee, "An Efficient Image Compression Algorithm Based on Block Optimization and Byte Compression", ICISA-2010, Chennai, Tamilnadu, India, pp.14-18, Feb 6, 2010.

[12]Ayan Banerjee and Amiya Halder, "An Efficient Image Compression Algorithm Based on Block Optimization, Byte Compression and Run-Length Encoding along Y-axis", IEEE ICCSIT 2010, Chengdu, China, IEEE Computer Society Press, July 9-11, 2010.

[13]Debashis Chakraborty, Sandipan Bera, Anil Kumar Gupta and Soujit Mondal,, "Efficient Data Compression using Character Replacement through Generated Code", IEEE NCETACS 2011, Shillong, India, March 4-5,2011.

[14] http://www.medicare.gov/download/downloaddb.asp.

Author

Mr. Debashis Chakraborty received the B.Sc. degree with Honours in Computer Science and M.Sc. degree in Computer and Information Science from the University of Calcutta, West Bengal, India, in 2001 and 2003, respectively. He obtained the M.Tech. degree in Computer Science and Engineering, from Vinayaka Mission University, Salem, Tamilnadu, India, in 2007. At present he is pursuing his Ph.D. in the field of Data and Image Compression from University of Calcutta, West Bengal, India. Mr. Chakraborty is an Assistant Professor in the department of Computer Science and Engineering, St. Thomas' College of Engineering and Technology, Kolkata, West Bengal, India. He has authored or co-authored over 6 conference papers in area of Data and Image Compression.

.