Data Compression Using Word based Indexing

ARIHANT JAIN, ARJUN SINGH, ASHVINI MEENA, DR. AYAN SEAL

PDPM IIITDM JABALPUR, MADHYA PRADESH

19 APRIL, 2018

ABSTRACT

A new way to compress blocks of data by using the index of the word in the file where words are considered as a single unit and processed further for data compression and decompression.

INTRODUCTION

Finding long common sequences is still an open problem to work for reaserchers as its time time complexity is N^2 . As the data is increasing enormously in todays world so its being difficult to keep all this generated data as it requires more and more data storage devices. But data compression techniques revolutionises data storage as data is converted to smaller data without loosing its originality. In any particular data format there is always some redundancy present in that data so compression algorithm find redundant blocks and start mapping some smaller data values to them and finally reducing overall size of the file a same approach is used in [1].

Study

Suppose a 5MB book on compression gives 2MB's zip file and 2 copy of same book gives 4MB's zip file but Idealy after compression zip file should be of 2 to 3 MB size because both the books in the folder are same but this the flaw of the zip.

A detailed study was performed before writing this paper ,people all around the world are working on this field and also worked in [1]

In [1] rabin karp method of finger printing is used to find long common string in the file and then replaced by the index number and block size like <50,32> where 50 is the index and 32 is the size of the block so string of size from 50th index is copied in place of <50,32> .

Finding long common string is such a hard problem and is not easy to use because of time and space constraints so in this [1] paper an sliding window of some constant block size is used with hashing function. An array is used to store hash values by following function

Val= 2^0*a0+2^1*a2+2^2*a2…………….2^30*a30+2^31*a31

Where a0,a1,a2………a30,a31 are the characters in the window size of 32. If computed values val is already present in hash array then this window is replaced by the index present at this hash value with block size 32 but if not then only this hash value is updated with the index of a0th element . Now next window is calculated by using previous window where first element of previous window is to be removed by

val=val-2^0*a0;

and a new last element to be appended in the window by

val=val/2+2^31*a32;

Now this is the next window values which can be used again for checking if this window is already present or not the hash table. Although it looks a linear algorithm but a lot of mathematics is included in this algorithm a it's a waste of computing power in some manner so there are some drawbacks for this approach

1. it only checks for the common strings of size equivalent to its window size even if the string matches with the b-1 character it doesn't recognizes it because it works only on calculating the hash values of 32 characters .

2. there is some probability that two different combination of string of size b have same hash value in this case it creates a state of confusion and originality can be decreased.

PROPOSED APPROACH 1

Its quite hard to process characters so a new method of word based indexing is used in this paper where each word is separated by spaces and a track of these words is used while running these compression and decompression programs.

Compression

A vector of string s is used to keep track of words used in the file till know whenever it encounters a word first it check the vector of strings if this word is already present in vector or not if present then it simply replaces the word with the index of the first occurring of the word if not then it saves this word in the vector with a new index and this is how it keep taking the input words . An extra feature is also used in this compression program if there are some consecutive matches present in the file then index number are separated by – because then is considered as a single word and less space is used .for example <1 2 3 4 5> is present previously but after this feature <1-2-3-4-5> this then later one will occupy less space it is observed practically while using compression techniques like .zip .tar .each operation of taking a word is contributing following time in overall time time complexity.

If N words are present in vector then logN is the time complexity for the search operation and logN also for insertion operation so for a file having N words overall time complexity of compression is

Log1+log2+log3……..log(N-1)+log(n)< log(N)+ log(N)+ log(N)… log(N)+ log(N)

T(N)<Nlog(n)

So upper bound of time complexity is Nlog(N) .

<u>Decompression</u>

After compression move to the decompression algorithm here also a vector of words is made . Program starts taking words from input one by one and keeps adding words in the vector on the other hand if index is recognized then it replaces this index with the word in that index by this approach decompression program works.

Time complexity is Nlog(N) where N is the number of words presentin file as same as compression's code because same operations are being performed here also.

<u>Compression example:</u>

Example: consider a line,

When you play the game of Thrones you win or you die. When you play the game of Thrones you win or you die.

<u>On compression</u> : When you play the game of Thrones 2 win or 2 die. 1-2-3-4-5-6-7-2-9-2-10

<u>Decompression example:</u>

When you play the game of Thrones 2 win or 2 die. 1-2-3-4-5-6-7-2-9-2-10

<u>On decompression</u>: When you play the game of Thrones you win or you die. When you play the game of Thrones you win or you die.

PROPOSED APPROACH 2

In this approach we are using longest common substring algorithm to find largest string which is a substring two strings. The whole text file is divided into blocks of size b bytes , therefore total blocks is N/b where N is the total size in bytes of the file . We are comparing two consecutive blocks one by one and finding the longest common substring among them. Now we are replacing the longest common substring in block 2 with the <start,end>, where start is the starting index of the longest common string in block1 and end is the last index of the longest common string in block1.

For example,we have a file of size N bytes and we divide them in block of size=b,than total no of blocks=N/b

B1	B2	B3	B4
B5	B6	B7	B8
B9	B10	B11	B12
B13	B14	B15	B16

We are now comparying B1 and B2 and replacing the long common string of B2 by <start,end> of B1. We apply same process for B3 and B4 and so on.

This method can be effective for general data as well in some special case where we append two same books one after another.

Time Complexity:

Time complexity in the best and average case is O(N),actually it is O(Nxb),where b is constant between [1,1000],so b is ignored

In the worst case time complexity is $O(N^2)$,where b = N.

Results and Comparaisions

Orignal data	[1]	Algo1	Algo2
Raw data(1.6Kb)	11%	17%	2%
Book1+Book1	~50%	10-15%	~50%
999999th block matches the last block of size 10 and in between contain random Small common string	-0.001%(if no other compression block found)	10%	2%

Conclusion

A new compression technique is used in this paper . It is easy to implement and less calculation are used in this approach. It discards large number indexing problem in [1] . In [1] hash values are stored in table which consumes memory and time which is not the case in our work. In general scenario our approach 1 provides constant compression ratio while approach 2 is provides massive compression ratio in some special case.

References

[1] "Data Compression Using Long Common Strings" by Jon Bentley Bell Labs, Room 2C-514 600 Mountain Avenue Murray Hill, NJ 07974 jlb@research.bell-labs.com and Douglas McIlroy Department of Computer Science Dartmouth College Hanover, New Hampshire 03755 doug@cs.dartmouth.edu