Modified Front Coding with variable Blocking Factor

Abstract

We know the importance of text compression in Information Retrieval. We are dealing with huge databases and thus the more space we save, the better. While front coding, there can be many instances where we can save space just by having a variable block length. In this project we aim to attain a better compression algorithm while front coding using the idea of variable block length.

1. Introduction

Dictionary as a string instead of a dictionary as an array is one of many ways to improve the space complexity of postings list. We can exploit the fact that consecutive entries in an alphabetically sorted list share common prefixes, for example, the word list [active, actively, activities] has the common prefix ‘active’. This observation leads to the idea of front coding. In front coding with blocking factor, k=4 we get 6a*ctive7◊ctively9◊ctivities6◊lgebra for the word list [active, actively, activities, algebra]. It can be seen that in this case, the front coding with blocking factor k = 4 is not optimal. So instead of taking k as constant, if we vary k according to consecutive words with longest common prefixes, then better compression is achieved. For example, if we split the block in two parts, with k=3 and k=1, we get, 6activ*e3◊ely5◊ities 6algebra. So, we can see that more than 10 bytes of space is being saved by doing the above modification.

More Examples:

1) Front coding with constant blocking factor k=8
9i* nfection5◊nform10◊nformation7◊nformed5◊nfuse10◊nstitution4◊ssue5◊ssues
Front coding with variable blocking factors k=1,3,2,2 respectively
9infection 5inform*5◊ation2◊ed 6in*fuse9◊stitution 5issue*1◊s
The variable blocking factor saves more than 10 bytes of space.
2) Front Coding with constant blocking factor k=4
5extra*2◊ct6◊ctable5◊ction 9extra*ctor4◊dite6◊dition6◊polate
Front coding with variable blocking factors k=8
5extra*2◊ct6◊ctable5◊ction4◊ctor4◊dite6◊dition6◊polate
The variable blocking factor saves 5 bytes of space.

2. Algorithm

As we can see in the above examples, in the approach of front coding with a constant blocking factor we realise that sometimes when the words do not have common prefixes they tend to use up more space than required or in some cases even more space than the space used without blocking factor. On the other hand, sometimes when many words have common prefixes, more space can be saved by using a larger blocking factor.

Using this particular motivation we came up with the idea of variable blocking factor. While doing so we also need to make sure that we don’t shoot up the algorithm complexity.

This is how we approach the problem:

1) While compressing every two words, we use up an extra 2 byte space.
   Using this 2 byte space is not taxing only if the compressed length is more than 2 bytes.
2) As long as the consecutive words have a common prefix of more than 3 characters, we keep on compressing those words together into one using the front coding.
3) This idea leads to front coding multiple words together as long as they have more than 3 characters in common prefix.
4) As soon as we encounter a word which has less than 3 characters in the common prefix of previous words, we start a new block from that word.

3. Results

We check our algorithm on four different datasets and then compare the results with constant k = 4, 6, 8, where k denotes the blocking factor. The values in the table represent the length of the dictionary as a string. The datasets namely Very Small, Small, Medium and Large have 1129, 12624, 79509 and 186928 number of characters in the datasets respectively.

Dataset	k = 4	k = 6	k = 8	Variable k
Very Small	**1378**	1433	1437	1459
Small	13765	13926	14056	**13574**
Medium	77418	78619	79659	**76802**
Large	**173299**	174993	177984	177319

(N.B. The links to the four datasets are given in the table above itself.)

As we can see in the above table of results, Variable k performs significantly better on the Small and Medium, while it falls slightly short on the Very Small dataset, and falls significantly on the Large dataset.
This happens since, when we are dealing with medium size datasets, the instances of multiple words with common prefix is significantly larger than those in the very small and large datasets.
Also the Medium size dataset has greater instances of words which don't have less than 3 characters in common prefix with the previous words. However, in our algorithm if we get a common prefix of length 3 of a large block we are not looking for a larger common prefix than that. This leads to inefficiency for large datasets.

4. Conclusion

We conclude that our algorithm with variable blocking factor performs better on the Medium sized datasets and helps significantly on the space compression without taxing the time complexity much.
Although as we observed the algorithm falls short on larger and very small datasets. We can safely ignore the very small datasets since they don’t really need the compression to that extent, but we definitely need to improve the results on the large datasets.

5. Thinking Forward

Here we introduce a few basic concepts on how the above algorithm can be improved further.
If we take a list of lexicographically sorted terms and find the common prefixes for each consecutive terms then we get a different kind of list. For example:
From the list ["infect", "infected", "infection", "inform", "informant", "information", "informed"], we create the list [6,6,3,6,6,6].
In this list we introduce the concept of :
Drop: A reduction in number of common prefixes for pairs of consecutive terms, e.g. 6,3,6 from the above list.
Elevation: An increase of at least 2 in number of common prefixes for pairs of consecutive terms, e.g. 3,6,6 from the above list.
Plane: No change in number of common prefixes for pairs of consecutive terms for at latest 3 consecutive pairs, e.g. 6,6,6 from the above list.

A **Drop** signifies the ending of a block. We have utilised this idea in our algorithm.
An **Elevation** signifies a more efficient blocking can be achieved from a blocking based on Drop only. This concept is crucial for large datasets. But as a tradeoff, this idea increases the model complexity so, we have dropped this idea in the current proposal.
A **Plane** gives a more efficient blocking than a blocking based only on Drop and Elevation. This concept is crucial for large datasets. But as a tradeoff, this idea increases the model complexity even more so, we have dropped this idea in the current proposal.