# Move-to-Front Encoding

Satyadev Nandakumar

August 4, 2017

## 1 Move-to-Front Encoding

This is a method that assigns smaller codes to letters which have appeared in the recent past. It does this while using only a finite amount of memory. The algorithm keeps an array consisting of the letters in the alphabet, initially arranged in lexicographic order. When we read a character in the input, we bring it to the first position in the array, moving the other letters back.

We illustrate the execution of the algorithm on an input string "mississippi", illustrating its efficacy on strings with a few characters which occur often.

| Current character | Array | Output |
|---|---|---|
| $\underline{\mathrm{m}}$ississippi | abcdefghijkl$\underline{\mathrm{m}}$nopqrstuvwxyz | 13 |
| m$\underline{\mathrm{i}}$ssissippi | $m$abcdefgh$\underline{\mathrm{i}}$jklnopqrstuvwxyz | 10 |
| mi$\underline{\mathrm{s}}$sissippi | $i$mabcdefghjklnopqr$\underline{\mathrm{s}}$tuvwxyz | 19 |
| mis$\underline{\mathrm{s}}$issippi | $\underline{s}$imabcdefghjklnopqrtuvwxyz | 1 |
| miss$\underline{\mathrm{i}}$ssippi | $s$$\underline{\mathrm{i}}$mabcdefghjklnopqrtuvwxyz | 2 |
| missi$\underline{\mathrm{s}}$sippi | $i$$\underline{\mathrm{s}}$mabcdefghjklnopqrtuvwxyz | 2 |
| missis$\underline{\mathrm{s}}$ippi | $\underline{s}$imabcdefghjklnopqrtuvwxyz | 1 |
| mississ$\underline{\mathrm{i}}$ppi | $s$$\underline{\mathrm{i}}$mabcdefghjklnopqrtuvwxyz | 2 |
| mississi$\underline{\mathrm{p}}$pi | $i$smabcdefghjklno$\underline{\mathrm{p}}$qrtuvwxyz | 13 |
| mississip$\underline{\mathrm{p}}$i | $\underline{p}$ismabcdefghjklnoqrtuvwxyz | 1 |
| mississipp$\underline{\mathrm{i}}$ | $p$$\underline{\mathrm{i}}$smabcdefghjklnoqrtuvwxyz | 2 |

If recent letters repeat, then the output is relatively short. This is the pattern that the Move-to-Front compressor exploits. We now compute the expected codelength of the MTF compressor with respect to the uniform distribution, and with respect to a favorable distribution.

**Example 1.** Suppose every letter in $[a-z]$ is equally likely, and every position is independently distributed of others. Then the expected codelength for encoding the first character is

$$\frac{1}{26}\log_2 1 + \frac{1}{26}\log_2 2 + \dots \frac{1}{26}\log_2 26 \quad = \quad \frac{1}{26}\sum_{i=1}^{26}\log_2 i \quad \approx \quad 4.08$$

Since the digits are identically and independently distributed and MTF does not remember any previous characters, it is easy to observe that the expected codelength for encoding the character in any position is the same - some character does get a shorter encoding, but others incur longer codes in such a manner that the expected codelength is identical to the first position.

The expected number of bits to represent an input character is the same, hence MTF on average neither compresses nor expands the input. $\Diamond$

**Example 2.** Suppose we have a distribution in which the top 3 elements of the matrix occur with probability 1/4 each, and the remaining elements are equally likely (each with probability 1/92). The expected codelength to encode the input at any given position is

$$\frac{1}{4}\left[\log_2 1 + \log_2 2 + \log_2 3\right] + \frac{1}{92}\left[\sum_{j=4}^{26} \log_2 j\right] = 1.579$$

which is shorter than the 4.034 expected bits per input character for the uncompressed input. ◊