Applied Algorithmics COMP526 – tutorial 6

L.A. Gąsieniec and D. Cartwright

1 Questions

1.1 Move-to-front (MTF)

Let $S = (20, 30, 30, 20, 40, 30, 20, 20, 20)$ be an input sequence of numbers whose values are initially stored in the list $Q = [20, 30, 40]$. Build an output sequence and trace the content of Q throughout the execution of *MTF (Move-to-Front) algorithm.*

1.2 Searching for substrings via suffix array

Explain how one can search for a string z in another string w with the help of the suffix array constructed for w.

1.3 Lempel-Ziv-Welch text compression

Given word $w = babaaaba \in \{a, b\}^*$. Construct, step by step, the *LZW* (Lempel-Ziv-Welch) factorisation of w and provide the compressed representation of w.

2 Solutions

2.1 Move-to-front (MTF)

MTF algorithm traverses through an input sequence value by value and replaces each value by its current position in the list and at the same time it moves this value to the beginning of the list. The MTF procedure is use to decrease the size of the input sequence (potentially large values are replaced by smaller dynamic indexes in the list). In case of input sequence S and initial content of Q the transformation process is as follows:

0)	**[20,30,40]**	0	5)	$[40, 20, 30]$	2
1)	$[20, 30, 40]$	1	6)	$[30, 40, 20]$	2
2)	$[30, 20, 40]$	0	7)	$[20, 30, 40]$	0
3)	$[30, 20, 40]$	1	8)	$[20, 30, 40]$	0
4)	$[20, 30, 40]$	2	9)	**[20,30,40]**	

2.2 Searching for substrings via suffix array

Note that the suffix array constructed for an input string w contains indices of w suffixes sorted in the lexicographical order, see example below.

Index	0 1 2 3 4 5 6 7
$w =$	b a b b a b a b
	a b
	a b a b
	a b b a b a b
	b
	b a b
	b a b a b
	b a b b a b a b
	b b a b a b

Suffix Array	
Index	Suffix
0	$[6]$
1	$[4]$
2	$[1]$
3	$[7]$
4	$[5]$
5	$[6]$
6	$[0]$
7	$[2]$

The last column with indices of suffixes in w defines the suffix array, where for the input string $w[0, .., 7] = babbabab$ we get an array:

Suffixes	[6 4 1 7 5 3 0 2]
Index	0 1 2 3 4 5 6 7

The suffix-array has a property that all suffixes of w that start with the same prefix form a contiguous block of positions in the suffix array. This is no surprise due to the property of the lexicographical order. For example, all suffixes that start with symbol a are located in the contiguous block $[0, .., 2]$, and all suffixes that start with ba reside in the block $[4, .., 6]$, etc. Thus when we look for a string z in the suffix tree constructed for w, we first identify block of suffixes that start with symbol $z[0]$, then we narrow it to the block of suffixes that start with symbols $z[0]z[1]$, etc. Each step is implemented as the binary search procedure that takes the logarithmic time. Thus in general the whole search process is limited to $m \log n$ steps where $|w| = n$ and $z = |m|$.

2.3 Lempel-Ziv-Welch text compression

In LZW compression mechanism we start with the dictionary containing initially symbols from the alphabet, in this case $\{a, b\}$, where the code words are: $cw_{-2} = b$ and $cw_{-1} = a$. There are two rules. (1) Each new code word is an extension of already existing one by a single symbol. (2) Codewords cw_i and cw_{i+1}, for $i \geq 0$, overlap on one symbol. This allows to store only one integer for each code word. This is very important from the implementation point of view. Thus the consecutive code words (in factorisation of $w[0..7] = babaaaba$) are: $cw_{-2} = b$; $cw_{-1} = a$; $cw_0 = w[0..1] = ba$; $cw_1 = w[1..2] = ab$; $cw_2 = w[2..4] = baa$; $cw_3 = w[4..5] = aa$; and $cw_4 = w[5..7] = aba$;

Now since each code word cw_i, for $i \geq 0$, is concatenation of cw_j, for some $j < i$, and the first symbol of cw_{j+1} (apart from the last code word) it is enough to store references to codewords cw_j. Thus the compressed representation is: [-2, -1, 0, -1, 1,+a] (the last symbol is treated differently).