UNIVERSITY OF LIVERPOOL

Department of Computer Science
Sebastian Wild

Date: 2020-03-04
Version: 2020-03-04 22:08

Tutorial 4 for COMP 526 – Applied Algorithmics, Winter 2020

—including solutions—

It is highly recommended that you first try to solve the problems on your own before consulting the sample solutions provided below.

Problem 1 (Fibonacci language and failure function)

The sequence of Fibonacci words $(w_i)_{i\in\mathbb{N}_0}$ is defined recursively:

$$\begin{aligned} w_0 &= \texttt{a} \\ w_1 &= \texttt{b} \\ w_n &= w_{n-1}\cdot w_{n-2} \qquad (n\ge 2) \end{aligned}$$

Unfolding the recursion yields $w_2 = \texttt{ba}$, $w_3 = \texttt{bab}$, $w_4 = \texttt{babba}$, an so on.

(Note that the lengths $|w_0|, |w_1|, |w_2|, \ldots$ are *Fibonacci numbers*, hence the name. More precisely, we have $|w_n| = F_{n+1}$, with the Fibonacci numbers defined as $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$, for $n \ge 2$.)

Construct the failure function F and the draw the KMP automaton with failure links for w_6.

Solutions for Problem 1 (Fibonacci language and failure function)

We first construct w_6 and all words preceding it in Fibonacci language: $w_0 = \texttt{a}$, $w_1 = \texttt{b}$, $w_2 = \texttt{ba}$, $w_3 = \texttt{bab}$, $w_4 = \texttt{babba}$, $w_5 = \texttt{babbabab}$ and finally $w_6 = \texttt{babbababbabba}$.

Recall that the failure *function* is stored in the array $F[1..|w_6|]$, where $F[j]$ is the length of the longest prefix of $\pi_j = P[0..j]$ that is a suffix of $P[1..j]$. Moreover, the failure *links* are given by $\mathit{fail}[j] = F[j-1]$.

w_6 = *babbababbabba*

π_1 = *ba* $\rightsquigarrow F[1] = 0$
ba

π_2 = *bab* $\rightsquigarrow F[2] = 1$
bab

π_3 = *babb* $\rightsquigarrow F[3] = 1$
babb

π_4 = *babba* $\rightsquigarrow F[4] = 2$
babba

π_5 = *babbab* $\rightsquigarrow F[5] = 3$
babbab

π_6 = *babbaba* $\rightsquigarrow F[6] = 2$
babbaba

π_7 = *babbabab* $\rightsquigarrow F[7] = 3$
babbabab

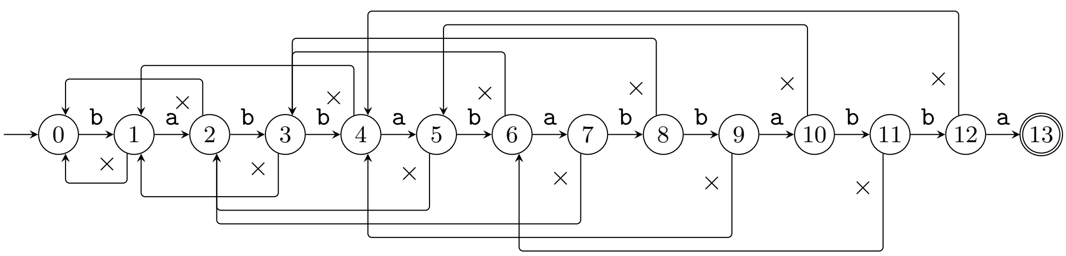
π_8 = *babbababb* $\rightsquigarrow F[8] = 4$
babbababb

π_9 = *babbababba* $\rightsquigarrow F[9] = 5$
babbababba

π_{10} = *babbababbab* $\rightsquigarrow F[10] = 6$
babbababbab

π_{11} = *babbababbabb* $\rightsquigarrow F[11] = 4$
babbababbabb

π_{12} = *babbababbabba* $\rightsquigarrow F[12] = 5$
babbababbabba



Problem 2 (How KMP uses itself)

Recall the example $T = \texttt{abababaabab}$ and $P = \texttt{ababaca}$ used in the lecture to illustrate the KMP failure-link automaton.

Now consider the string $S = S[0..m+n] = P\,\$\,T$ over the extended alphabet $\Sigma' = \Sigma \cup \{\$\} = \{\texttt{a}, \texttt{b}, \texttt{c}, \$\}$ and construct the failure-links array $fail[0..n+m]$.

Compare the result with the sequence of states from simulation the failure-link automaton for P on T; what do you observe?

Bonus: Can you compute the values $fail[0..n+m]$ using only $\Theta(P)$ extra space? Here, it is enough to have the values available at some time during the computation; we (obviously) cannot store all of them explicitly in the allowed space.

Solutions for Problem 2 (How KMP uses itself)

We have $m = 7$, $n = 11$; we find the following failure link values:

q	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
$q-m-1$										0	1	2	3	4	5	6	7	8	9	10
$S[q+1]$	–	a	b	a	b	a	c	a	\$	a	b	a	b	a	b	a	a	b	a	b
$fail[q]$	0	0	0	1	2	3	0	1	0	1	2	3	4	5	4	5	1	2	3	4

The entries can again be found using overlaps (see examples below) or by following the efficient code from the lecture.

```
ababaca$abababaabab

ababaca
      ababaca

ababaca$
        ababaca$

ababaca$aba
        ababaca$aba

ababaca$abab
        ababaca$abab

ababaca$ababab
          ababaca$ababab

ababaca$abababa
          ababaca$abababaa

ababaca$abababaa
               ababaca$abababaa

ababaca$abababaabab
               ababaca$abababaabab
```

We observe that the sequence of failure links after `$` is precisely the sequence of states obtained from simulating the KMP automaton for P!

This is of course no coincidence; indeed, the matching part of KMP does the *very same* steps in computing the current state q as the failureLinks procedure does for maintaining the failure state x. This is not totally obvious from the given pseudocodes in class (which are meant to most closely resemble the example constructions), but one can indeed rewrite the code so that the parallel structure becomes more visible.

In this sense, KMP is indeed using itself when it constructs the failure links for P.

Finally, we note that any computed failure links for S lead to a state left of the position of `$` since there is no way to overlap this character with any other. The computation of the next failure link thus never accesses values of *fail* beyond m, so that we need not store those to be able to compute all remaining values, but we can still compute all failure links one by one. (This is, again, exactly analogous to how the KMP matching procedure computes – but does not store – the sequence of states of the KMP automaton simulation.)

Finally, we note that whenever $\mathit{fail}[q] = m$, we have found a match; for the failure links based on S, this match begins at position $q - m - 1$, and we can report this match in passing.