Efficient validation and construction of Knuth–Morris–Pratt arrays

Jean-Pierre Duval Thierry Lecroq Arnaud Lefebvre

LITIS, Université de Rouen
{Jean-Pierre.Duval,Thierry.Lecroq,Arnaud.Lefebvre}@univ-rouen.fr

Abstract

Knuth-Morris-Pratt (KMP) arrays are known as the "failure function" of the Knuth-Morris-Pratt string matching algorithm. We present an algorithm to check if an integer array is a KMP array. This gives a method for computing all the distinct KMP arrays.

1 Introduction

A border u of a string w is a prefix and a suffix of w such that $u \neq w$. The computation of the border array of a string w i.e. of the borders of each prefix of a string w is strongly related to the string matching problem: given a string w, find the first or, more generally, all its occurrences in a longer string y. The border array of w is better known as the "failure function" introduced in [5]. In [3] a method is presented to check if an integer array f is a border array for some string w. In [1], we gave a more elegant presentation of this result. In [2] we lowered the delay (time spent on one element of the array) from $O(|w|)$ to $O(\min\{|\Sigma|, |w|\})$ comparing to algorithms in [3, 1]. Moreover we presented new results concerning the relation between the border array f and the skeleton of the deterministic finite automaton recognizing $\Sigma^* \cdot w$.

In the present article we deal with KMP (Knuth-Morris-Pratt) arrays instead of border arrays. KMP arrays are used as "failure function" in the Knuth-Morris-Pratt string matching algorithms [4]. Given an integer array g, we can decide if g is the KMP array of some string w on a bounded alphabet of size s. If it is not, we can compute the longest prefix of g for which there exists a string w such that the prefix of g is the KMP array of w. Actually these results are completely independent from w. We are also capable of generating all the distinct KMP arrays in time proportional to their numbers.

2 Notations and definitions

In the following we use an alphabet Σ of size s and $\sigma[i]$ denotes the i-th letter of Σ. A string u is a *border* of w if u is a prefix and a suffix of w and $u \neq w$. *The* *border* of a string w is the longest of its borders. It is denoted by $Border(w)$. The *border array* f_w of a string w of length n is defined by: $f_w[i] = |Border(w[1\,..\,i])|$ for $1 \le i \le n$. It is also known as the "failure function" of the Morris and Pratt string matching algorithm [5].

The *KMP array* g_w of a string w of length n is defined by: $g_w[1] = 0$ and $g_w[j] = \max\{\{i \mid w[1\,..\,i-1] \text{ suffix of } w[1\,..\,j-1] \text{ and } w[i] \neq w[j]\} \cup \{0\}\}$ or equivalently $g_w[j] = 1 + \max\{\{i \mid w[1\,..\,i] \text{ border of } w[1\,..\,j-1] \text{ and } w[i+1] \neq w[j]\} \cup \{-1\}\}$ for $2 \le j \le n$. Array g_w is known as the "failure function" of the Knuth-Morris-Pratt string matching algorithm [4].

Example 1 The border and KMP arrays of `ababacaabcababa` are the following:

i	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$w[i]$	a	b	a	b	a	c	a	a	b	c	a	b	a	b	a
$f_w[i]$	0	0	1	2	3	0	1	1	2	0	1	2	3	4	5
$g_w[i]$	0	1	0	1	0	4	0	2	1	3	0	1	0	1	0

The following definition introduces the notion of valid arrays.

Definition 1 *An integer array $f[1\,..\,n]$ is a* valid *border array if and only if it is the border array of at least one string $w[1\,..\,n]$.*

Definition 2 *An integer array $g[1\,..\,n]$ is a* valid *KMP array if and only if it is the KMP array of at least one string $w[1\,..\,n]$.*

The deterministic finite automaton $\mathcal{D}(w)$ recognizing the language $\Sigma^* \cdot w$ is defined by $\mathcal{D}(w[1\,..\,n]) = (Q, \Sigma, q_0, T, F)$ where $Q = \{0, 1, \ldots, n\}$ is the set of states, Σ is the alphabet, $q_0 = 0$ is the initial state, $T = \{n\}$ is the set of accepting states and $F = \{(i, w[i+1], i+1) \mid 1 \le i \le n\} \cup \{(i, a, |Border(w[1\,..\,i]a)|) \mid 1 \le i \le n \text{ and } a \in \Sigma \setminus \{w[i+1]\}\}$ is the set of transitions. The underlying unlabeled graph is called the *skeleton* of the automaton. We denote by $\delta_w(i)$ the list $(j \mid (i, a, j) \in F \text{ with } a \in \Sigma \text{ and } j \neq 0)$ and by $\delta'_w(i)$ the list $(j \mid (i, a, j) \in F \text{ with } a \in \Sigma \text{ and } j \notin \{0, i+1\})$ for $0 \le i \le n$. In other words $\delta_w(i)$ is the list of the targets of the significant transitions leaving state i and $\delta'_w(i)$ is the list of the targets of the backward significant transitions leaving state i.

3 Known results

Let $f[1\,..\,n]$ be an integer array such that $f[i] < i$ for $1 \le i \le n$.

The following proposition shows how to build, from a border array f, the skeleton of the automaton recognizing $\Sigma^* \cdot w$ for any string w having f as its border array.

Proposition 1 $\delta(0) = (1)$ *and* $\delta(j) = (j+1) \uplus \delta(f[j]) \cup (f[j+1])$ *for* $1 \le j < n$ *and* $\delta(n) = \delta(f[n])$.

The next statement is a corollary of the previous proposition and gives the construction of the border array f from the skeleton of an automaton.

Corollary 1 *For $j > 0$:*

$$f[j+1] = \begin{cases} \delta(f[j]) \uplus \delta(j) & \text{if } \delta(f[j]) \uplus \delta(j) \text{ is not empty,} \\ 0 & \text{otherwise.} \end{cases}$$

4 New results

Two strings x and y can have the same KMP array and different border arrays.

Example 2 Consider the two strings $x = \texttt{abaab}$ and $y = \texttt{abacb}$.

i	1	2	3	4	5
$x[i]$	a	b	a	a	b
$f_x[i]$	0	0	1	1	2
$g_x[i]$	0	1	0	2	1

i	1	2	3	4	5
$y[i]$	a	b	a	c	b
$f_y[i]$	0	0	1	0	0
$g_y[i]$	0	1	0	2	1

Given a valid KMP array $g[1\,..\,i]$, an associated border array $f[1\,..\,i]$ and the skeleton of automaton δ' the following propositions hold.

Proposition 2 *Then $g[i+1]$ can either be equal to $f[i]+1$ or to $g[f[i]+1]$.*

Proposition 3 *If $g[i+1] = g[f[i]+1]$ then $f[i+1] = f[i]+1$.*

Proposition 4 *If $g[i+1] = f[i]+1$ then $f[i+1]$ can be any value in $\delta'(i) \uplus (f[i]+1) \uplus (0)$.*

In order to check if a given integer array g of length n is a valid KMP array, it is necessary to build along an associated border array f and a skeleton δ'. When Proposition 4 applies the different choices are tried until one succeeds or all fail.

The algorithm VERIFY(1), given in Figure 1, returns `TRUE` if an integer array g of length n is valid and `FALSE` otherwise. When g is valid it moreover builds a string w for which g is the KMP array. It assumes that the variables g, f, δ', α and w are global. It applies Propositions 2 to 4.

For instance, with the array $g = 0{\cdot}1{\cdot}0{\cdot}1{\cdot}0{\cdot}4{\cdot}0{\cdot}2{\cdot}1{\cdot}3{\cdot}0{\cdot}1{\cdot}0{\cdot}1{\cdot}0$ of Example 2, the algorithm VERIFY produces the string $w = \texttt{ababacaabbababa}$ enhancing the fact that `ababacaabcababa` is not the smallest lexicographic string having g as a KMP array.

Regarding the complexity, integer arrays $g(n)$ of the form $0 \cdot 1 \cdot 0 \cdot (2 \cdot 1 \cdot 0)^* \cdot (1|2 \cdot 0|2 \cdot 1 \cdot 1)$ requires the following number of calls of the function VERIFY:

- $3(((n/3) \times (n/3) + 1)/2)$ if $n \bmod 3 = 1$;
- $2 + 3((((n+1)/3) \times ((n+1/)3) + 1)/2) - n/3$ if $n \bmod 3 = 0$;
- $2 + 3((((n-1)/3) \times ((n-1/)3) + 1)/2) + n/3 + 1$ if $n \bmod 3 = 2$.

Experimentally we did not find other worse cases so we conjecture that the function VERIFY is quadratic.

```
Verify(j)
if j = n + 1 then
  return TRUE
else if g[j] ≠ f[j − 1] + 1 then
    if g[j] ≠ g[f[j − 1] + 1] then
      return FALSE
    else (f[j], w[j]) ← (f[j − 1] + 1, w[f[j]])
      δ(j − 1) ← δ(j − 1) ⊎ (f[j − 1] + 1) ⊎ (j)
      (α[j], δ(j)) ← (α[f[j]], δ(f[j]))
      return Verify(j + 1)
  else for k ∈ δ(j − 1) ⊎ (f[j − 1] + 1) do
      (f[j], w[j]) ← (k, w[f[j]])
      δ(j − 1) ← δ(j − 1) ⊎ (f[j]) ⊎ (j)
      (α[j], δ(j)) ← (α[f[j]], δ(f[j]))
      if Verify(j + 1) then
        return TRUE
      δ(j − 1) ← δ(j − 1) ⊎ (j) ⊎ (f[j])
    if α[j − 1] < s then
      (f[j], w[j]) ← (0, α[j − 1])
      α[j − 1] ← α[j − 1] + 1
      δ(j − 1) ← δ(j − 1) ⊎ (j)
      (α[j], δ(j)) ← (α[f[j]], δ(f[j]))
      return Verify(j + 1)
    else return FALSE
```

Figure 1: Verification of an integer array.

5 Counting distinct KMP arrays

In order to generate all the valid KMP array, we generate them along with an associated border array and an automaton skeleton. Since a valid KMP array can be generated from different border arrays, we need to store them. To that aim we can use a lexicographic trie.

Let $K(n)$ be the number of distinct KMP arrays of length n on an unbounded alphabet and let $K(n, s)$ be the number of distinct KMP arrays of length n on an alphabet of size s. Table 1 gives the number of distinct KMP arrays of length 1 to 18 for an unbounded alphabet and alphabets of size 2 to 4.

$K(5, 2) = K(5) - 1$: the missing KMP array is $0 \cdot 1 \cdot 0 \cdot 2 \cdot 0$, it is the KMP array of `abaca`. $K(10, 3) = K(10) - 2$: the two missing KMP arrays are $0 \cdot 1 \cdot 0 \cdot 2 \cdot 0 \cdot 1 \cdot 0 \cdot 4 \cdot 0 \cdot 1$ and $0 \cdot 1 \cdot 0 \cdot 2 \cdot 0 \cdot 1 \cdot 0 \cdot 4 \cdot 1 \cdot 1$, they are the KMP arrays of `abacabadab` and `abacabadbb` respectively. $K(18, 4) = K(18) - 1$: the missing KMP array is $0 \cdot 1 \cdot 0 \cdot 2 \cdot 0 \cdot 1 \cdot 0 \cdot 4 \cdot 0 \cdot 1 \cdot 0 \cdot 2 \cdot 0 \cdot 1 \cdot 0 \cdot 8 \cdot 1 \cdot 1$, it is the KMP array of `abacabadabacabaebb`. Let $w_1 = \sigma[1]$. Let $w_i = w_{i-1} \cdot \sigma[i] \cdot w_{i-1}$ for $i > 1$. Let $g_1 = 0$. Let $g_i = g_{i-1} \cdot 2^i \cdot g_{i-1}$ for $i > 1$. For $i \geq 4$, $K(2^i + 2, i) = K(2^i + 2) - 1$:

Table 1: Number of distinct KMP arrays on different alphabets.

i	$K(i)$	$K(i,2)$	$K(i,3)$	$K(i,4)$	i	$K(i)$	$K(i,2)$	$K(i,3)$	$K(i,4)$
1	1	1	1	1	10	1106	512	**1104**	1106
2	2	2	2	2	11	2656	1024	2644	2656
3	4	4	4	4	12	6414	2048	6365	6414
4	8	8	8	8	13	15,582	4096	15,406	15,582
5	17	**16**	17	17	14	38,011	8192	37,430	38,011
6	37	32	37	37	15	93,124	16,384	91,317	93,124
7	85	64	85	85	16	228,927	32,768	223,524	228,927
8	197	128	197	197	17	564,674	65,536	548,969	564,674
9	465	256	465	465	18	1,396,860	131,072	1,352,193	**1,396,859**

the missing KMP array is $g_i \cdot 1 \cdot 1$, it is the KMP array of $w_i \cdot \sigma[2] \cdot \sigma[2]$.

References

[1] J.-P. Duval, T. Lecroq, and A. Lefebvre. Border array on bounded alphabet. *J. Autom. Lang. Comb.*, 10(1):51–60, 2005.

[2] J.-P. Duval, T. Lecroq, and A. Lefebvre. Efficient validation and construction of border arrays. In *Proceedings of the Mons Days of Theoretical Computer Science (JM 2006)*, pages 179–189, Rennes, France, 2006.

[3] F. Franěk, S. Gao, W. Lu, P. J. Ryan, W. F. Smyth, Y. Sun, and L. Yang. Verifying a border array in linear time. *J. Comb. Math. Comb. Comp.*, 42:223–236, 2002.

[4] D. E. Knuth, J. H. Morris, Jr, and V. R. Pratt. Fast pattern matching in strings. *SIAM J. Comput.*, 6(1):323–350, 1977.

[5] J. H. Morris, Jr and V. R. Pratt. A linear pattern-matching algorithm. Report 40, University of California, Berkeley, 1970.