Border Array on Bounded Alphabet [1]

Jean-Pierre Duval[2], Thierry Lecroq[2], Arnaud Lefebvre[3]

[2]LIFAR – ABISS, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France
[3]UMR 6037 – ABISS, Université de Rouen, 76821 Mont-Saint-Aignan Cedex, France

e-mail: `Jean-Pierre.Duval`, `Thierry.Lecroq`, `Arnaud.Lefebvre` `@univ-rouen.fr`

Abstract. In this article we present an on-line linear time algorithm, to check if an integer array f is a border array of some string x built on a bounded size alphabet, which is simplest that the one given in [2]. Furthermore if f is a border array we are able to build, on-line and in linear time, a string x on a minimal size alphabet for which f is the border array.

Key words: String algorithms, border array

1 Introduction

A border u of a string x is a prefix and a suffix of x such that $u \neq x$. The computation of the borders of each prefix of a string x is strongly related to the string matching problem: given a string x, find the first or, more generally, all its occurrences in a longest string y. The border array of x is better known as the "failure function" introduced in [4] (see also [1]). Recently, in [2] a method is presented to check if an integer array f is a border array for some string x. The authors first give an on-line linear time algorithm to verify if f is a border array on an unbounded size alphabet. Then they give a more complex algorithm that works on a bounded size alphabet. Here we present a more simple algorithm for this case. Furthermore if f is a border array we are able to build, on-line and in linear time, a string x on a minimal size alphabet for which f is the border array. The resulting algorithm is elegant and integrates three parts: the checking on an unbounded alphabet, the checking on a bounded size alphabet and the design of the corresponding string if f is a border array. The first two parts can work independently.

The remaining of this article is organized as follows. The next section introduces basic notions and notations on strings and results from [2]. Section 3 presents our new algorithm together with its correctness proof. Finally we give our conclusions in Sect. 4.

2 Background and basic string definitions

A *string* is a sequence of zero or more symbols from an alphabet Σ; the string with zero symbols is denoted by ε. The set of all strings over the alphabet Σ is denoted

[1]This work was partially supported by a NATO grant PST.CLG.977017.

by Σ^*. We consider an alphabet of size s; for $1 \leq i \leq s$, $\sigma[i]$ denotes the i-th symbol of Σ. A string x of length n is represented by $x[1..n]$, where $x[i] \in \Sigma$ for $1 \leq i \leq n$. A string u is a *prefix* of x if $x = uw$ for $w \in \Sigma^*$. Similarly, u is a *suffix* of x if $x = wu$ for $w \in \Sigma^*$. A string u is a *border* of x if u is a prefix and a suffix of x and $u \neq x$.

Let $f[1..n]$ be an integer array such that $f[i] < i$ for $1 \leq i \leq n$. For $1 \leq i \leq n$, we define $f^1[i] = f[i]$ and for $f[i] > 0$, $f^\ell[i] = f[f^{\ell-1}[i]]$. We use the following notations:

- $L(f, i-1) = (f[i-1], f^2[i-1], \ldots, f^m[i-1] = 0)$;
- $C(f, i) = (1 + f[i-1], 1 + f^2[i-1], \ldots, 1 + f^m[i-1])$ where $f^m[i-1] = 0$.

Note that $L(f, 1) = (0)$ and that $C(f, 1)$ is not defined.

A border u of $x[1..i]$ with $i > 0$ has one of the two following forms:

- $u = \varepsilon$;
- $u = x[1..j]x[j+1]$ with $j + 1 < i$ and where $x[1..j]$ is a border of $x[1..i-1]$ and $x[i] = x[j+1]$.

For $1 \leq i \leq n$ we denote by $\beta_x[i]$ the length of the longest border of $x[1..i]$. The array $\beta_x[1..n]$ is said to be the border array of the string x.

The lengths of the different borders of $x[1..i-1]$ are given by the decreasing sequence

$$L(\beta_x, i-1) = (\beta_x[i-1], \beta_x^2[i-1], \ldots, \beta_x^m[i-1])$$

where $\beta_x^m[i-1] = 0$ i.e. it is the length of the longest border $\beta_x[i-1]$ followed by the lengths of the borders of this longest border $L(\beta_x, \beta_x[i-1])$.

For $i \geq 2$, we say that an integer $j+1$ is candidate to be the length of the longest border of $x[1..i]$ if $x[1..j]$ is a border of $x[1..i-1]$. In other words, for $i \geq 2$, saying that $j+1$ is candidate means that $j \in L(\beta_x, i-1)$. The decreasing sequence of candidates for the length of the longest border of $x[1..i]$ is

$$C(\beta_x, i) = (1 + \beta_x[i-1], 1 + \beta_x^2[i-1], \ldots, 1 + \beta_x^m[i-1])$$

where $\beta_x^m[i-1] = 0$.

We say that an array $f[1..n]$ is a *valid border array*, or simply that it is *valid* if and only if it is the border array of at least one string x of length n.

The longest border of $x[1]$ is necessarily the empty word, thus $\beta_x[1] = 0$. The length $\beta_x[i]$ of the longest border of $x[1..i]$, if it is not empty, is taken among the candidates $C(\beta_x, i)$. Thus we have a first necessary condition for an array $f[1..n]$ to be valid:

$$NC_1\colon\ f[1] = 0 \text{ and for } 2 \leq i \leq n, f[i] \in \{0\} + C(f, i)\,.$$

If $x[1..i]$ has the empty word for only border then we have $\beta_x[i] = 0$.

If $x[1..i]$ has a non-empty border, the length of the longest border verifies

- $\beta_x[i] = \max\{j + 1 \mid j \in L(\beta_x, i-1) \text{ and } x[i] = x[j+1]\}$, or equivalently
- $\beta_x[i] = \max\{j + 1 \mid j + 1 \in C(\beta_x, i) \text{ and } x[i] = x[j+1]\}$.

The length $j+1$ of the longest border of $x[1..i]$ is the first candidate in the list $C(\beta_x, i)$ for which $x[j+1] = x[i]$ if it exists, otherwise the longest border has length 0. This is the basis of the computation of the function β_x known as a "failure function" given in [4].

Saying that $j+1$ is the largest candidate for which $x[j+1] = x[i]$ implies that this is not true for any candidate $j'+1$ larger than $j+1$, which imposes that $x[1..j+1]$ cannot be a border of $x[1..j'+1]$ for a candidate $j'+1$ larger than $j+1$. In other words, $\beta_x[j'+1]$ is different from $j+1$ for any candidate $j'+1$ larger than $j+1$.

This is thus a second necessary condition for an array f to be valid:

NC_2: for $i \geq 2$ and for every $j'+1 \in C(f, i)$ with $j'+1 > f[i]$ we have $f[j'+1] \neq f[i]$.

Theorem 2.2 in [2] states that conditions NC_1 and NC_2 form a sufficient condition for f to be a valid border array. The authors give, for any valid array f, thus satisfying conditions NC_1 and NC_2, the computation of a string x such that $f = \beta_x$, without any restriction on the alphabet size. They give a simple linear time algorithm (Theorem 2.3) to test if an array f satisfies conditions NC_1 and NC_2, on a unbounded size alphabet. They give a more complex algorithm in the case of a bounded size alphabet. Here we present a more simple algorithm which determines in linear time, for a given array $f[1..n]$, for i from 1 to n, the minimum size of an alphabet necessary to build a string $x[1..i]$ which border array is $f[1..i]$.

3 New algorithm

We propose, in this section, a linear time algorithm, which determines, for an array $f[1..n]$ and an alphabet size s given as input:

1 – validity: if $f[1..n]$ is a valid border array for at least one string $z[1..n]$. This point is essentially the same as in [2];

2 – alphabet: up to which index it is possible to build a string which border array is f using an alphabet of size s;

3 – string: a string x, on a minimal size alphabet, which border array is f.

Point 1 is independent from the other two points. Point 2 can work without the other two points, in particular when one assumes that the array f is valid and does not want to build a corresponding string. Point 3 uses point 2.

The algorithm BABA (for Border Array on Bounded Alphabet) is given figure 1.

We now state our main result.

Theorem 1 *When applied to an integer array $f[1..n]$ and an alphabet size s:*

- *The algorithm BABA runs in time $\Theta(n)$.*
- *If the array f given as input of the algorithm BABA is a valid border array at index $i-1$ but not at index i, the algorithm stops and returns "f invalid at index i". The lines* {**alphabet**} *and* {**string**} *can be deleted without changing this result.*

```
BABA(f, n, s)
 1  if f[1] ≠ 0                                          ▷ validity
 2     then return f invalid at index 1                  ▷ validity
 3  k[1] ← 1                                             ▷ alphabet
 4  x[1] ← σ[1]                                          ▷ string
 5  for i ← 2 to n
 6       do if f[i] = 0
 7             then k[i] ← 1 + k[f[i − 1] + 1]           ▷ alphabet
 8                  if k[i] > s                          ▷ alphabet
 9                     then return s exceeded at index i ▷ alphabet
10                  x[i] ← σ[k[i]]                       ▷ string
11             else j ← f[i − 1]                         ▷ validity
12                  while j + 1 > f[i] and f[j + 1] ≠ f[i] ▷ validity
13                      do j ← f[j]                      ▷ validity
14                  if j + 1 ≠ f[i]                      ▷ validity
15                     then return f invalid at index i  ▷ validity
16                  k[i] ← k[f[i − 1] + 1]               ▷ alphabet
17                  x[i] ← x[f[i]]                       ▷ string
18  return x
```

Figure 1: Algorithm BABA

- *If there exists a string for which $f[1..i-1]$ is the border array and there is none at index i with an alphabet of size s, the algorithm BABA stops and returns "s exceeded at index i". Lines* {**string**} *can be deleted without changing this result. If the array f is valid, lines* {**validity**} *can also be deleted.*
- *As long as $f[i..1]$ is valid, the algorithm BABA builds a string $x[1..i]$ on a minimal size alphabet for the border array $f[1..i]$. Lines* {**validity**} *can be deleted without changing the construction of the string. It is clear that if f is invalid, it is not the border array of the string which is built by the algorithm.*

Before giving the proof of the previous theorem we first give a definition and establish some intermediate results.

Definition 1 *Given a string $x[1..n]$ and its border array β_x, we denote by $A(x,i)$ the set of symbols that extend the prefix $x[1..i-1]$ and its borders, in x: $A(x,i) = \{x[i]\} \cup \{x[j+1] \mid j+1 \in C(\beta_x, i)\}$.*

Figure 2 gives a description of $L(\beta_x, i-1)$, $C(\beta_x, i)$ and $A(x,i)$.

Lemma 1 *For every string $x[1..i]$ we have*

1. $\{x[j+1] \mid j+1 \in C(\beta_x, i)\} = A(x, \beta_x[i]+1)$;
2. *If $\beta_x[i] \neq 0$ then $x[i] = x[\beta_x[i]]$, $\beta_x[i] \in C(\beta_x, i)$ and $A(x,i) = A(x, \beta_x[i-1]+1)$.*
3. *If $\beta_x[i] = 0$ then $\beta_x[i] \notin C(\beta_x, i)$ and $A(x,i) = \{x[i]\} \cup A(x, \beta_x[i-1]+1)$.*

Proof:

1. Immediate;

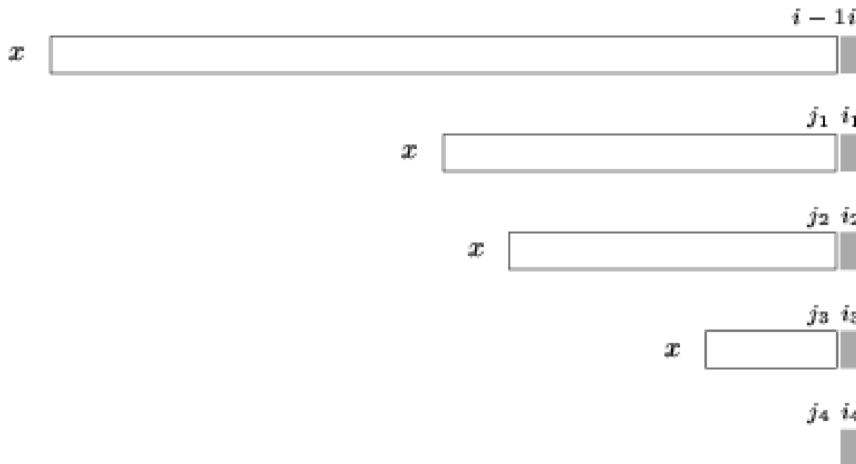

Figure 2: If for $1 \le \ell \le 4$, $j_\ell = \beta_x^\ell[i-1]$, $j_4 = \beta_x^4[i-1] = 0$, $i_\ell = 1 + \beta^\ell[i-1]$, then $L(\beta_x, i-1) = (j_1, j_2, j_3, j_4 = 0)$, $C(\beta_x, i) = (i_1, i_2, i_3, i_4 = 1)$ and $A(x, i)$ is the set which is composed of the gray symbols.

2. If $\beta_x[i] \neq 0$ then $\beta_x[i]$ is a candidate of $C(\beta_x, i)$. Concerning the index of the longest border we have $x[i] = x[\beta_x[i]]$, $\beta_x[i]$ is a candidate in $C(\beta_x, i)$, $x[i]$ is in $A(x, \beta_x[i-1]+1)$;

3. $\beta_x[i] = 0$ implies that there exists no candidate $j+1 \in C(\beta_x, i)$ such that $x[i] = x[j+1]$.

□

Corollary 1 *Let $x[1..n]$ be a string and $k[1..n]$ the array computed by the algorithm BABA with the input $f = \beta_x$ ignoring the* {**validity**} *and* {**string**} *lines. Then, for $1 \le i \le n$ we have $k[i] = card A(x, i)$.*

Proof: The proof of the corollary immediately follows from the algorithm BABA and properties 2 and 3 of lemma 1. □

Corollary 2 *For every string x which border array is f, the minimal cardinality of an alphabet necessary to build each prefix $x[1..i]$ is greater or equal to* $\max\{k[1], k[2], \ldots, k[i]\}$ *where $k[1..n]$ is the array computed by the algorithm BABA with the input $f = \beta_x$, ignoring lines* {**validity**} *and* {**string**}.

Proof: All the symbols of $A(x, j)$ for $1 \le j \le i$ are symbols of the string $x[1..i]$. Thus the cardinality is greater or equal to the cardinality of each $A(x, j)$. □

Proposition 1 *Assume that array $f[1..n]$ is valid. The string x build by the algorithm BABA satisfies the following properties:*

1. *For $1 \le i \le n$, $\beta_{x[1..i]} = f[1..i]$ and $A(x, i) = \{\sigma[1], \sigma[2], \ldots, \sigma[k[i]]\}$;*

2. *The cardinality of the alphabet for each prefix $x[1..i]$ is equal to*

$$\max\{k[1], k[2], \ldots, k[i]\};$$

3. *The border array β_x of the string x is equal to f.*

Proof:

- We show the point 1 by induction on i. For $i = 1$: $f[1] = 0$, $\beta_{x[1..1]} = f[1..1]$ and $A(x, 1) = \{x[1]\} = \{\sigma[1]\}$. The property holds at index 1.

  Assume that the property holds up to index $i-1$, then we have $A(x, f[i-1]+1) = \{\sigma[1], \sigma[2], \ldots, \sigma[k[f[i-1]+1]]\}$ (since $f[i-1] < i-1$ thus $f[i-1]+1 \leq i-1$) and $\beta_{x[1..i-1]} = f[1..i-1]$.

  If $f[i] \neq 0$ then since $f[1..i-1] = \beta_{x[1..i-1]}$ and f satisfies conditions NC_1 and NC_2 at index i, $f[i]$ is the largest candidate j of $C(f, i)$ such that $x[j] = x[f[i]]$. Thus, by setting $x[i] \leftarrow x[f[i]]$ we get $\beta_x[i] = f[i]$, $k[i] = k[f[i-1]+1]$ and $A(x, i) = A(x, f[i-1]+1) = \{\sigma[1], \sigma[2], \ldots, \sigma[k[i]]\}$.

  If $f[i] = 0$ then $k[i] = 1 + k[f[i-1]+1]$ and $x[i] \leftarrow \sigma[k[i]]$ does not belong to $A(x, f[i-1]+1)$ thus $\beta_x[i] = 0$, $A(x, \beta_x[i-1]+1) = \{\sigma[1], \sigma[2], \ldots, \sigma[k[f[i-1]+1]]\}$, $A(x, i) = \{\sigma[k[i]]\} \cup A(x, \beta_x[i-1]+1) = \{\sigma[1], \sigma[2], \ldots, \sigma[k[i]]\}$.

  The property holds for i in both cases.

- Properties 2 and 3 are immediate consequences of property 1.

□

Proposition 2 *Let $f[1..n]$ be an integer array.*

1. *The algorithm BABA returns "f invalid at index i" if and only if $f[1..i-1]$ is valid and $f[1..i]$ is not;*

2. *The array $f[1..i-1]$ is the border array of the string $x[1..i-1]$ which is built by the algorithm BABA.*

Proof: From proposition 1, as long as $f[1..i]$ is valid, it is the border array of the string $x[1..i]$ which is built by the algorithm BABA which establishes the point 2.

If the algorithm BABA stops at index $i = 1$ and returns "f invalid at index 1", it means that $f[1] \neq 0$ thus $f[1..i]$ is invalid (note that this case cannot happen if the condition $f[i] < i$ is fulfilled).

Now assume that at the beginning of iteration i we have: $z[1..i-1]$ is a string which border array is $f[1..i-1]$ and z can be extended with a symbol $z[i]$ for which $\beta_z[i] = f[i]$.

We have $z[i] = z[f[i]]$, and $\beta_z[i] = f[i]$ is the largest candidate $j'+1 \in C(\beta_z, i) = (1 + \beta_z[i-1], 1 + \beta_z^2[i-1], \ldots, 1 + \beta_z^m[i-1])$, such that $z[j'+1] = z[i]$ thus it is the largest for which $z[j'+1] = z[f[j]]$.

The three lines {**validity**} of the algorithm BABA reviews in decreasing order the candidates $j+1$ of $C(\beta_z, i)$.

- If the algorithm exits the while loop with $j+1 > f[i]$ and $f[j+1] = f[i]$, it means that $j+1$ is a candidate larger that $f[i]$ for which $\beta_z[j+1] = f[i]$ thus $z[j+1] = z[f[i]]$ which contradicts the fact that $j'+1$ is the largest candidate such that $z[j'+1] = z[f[i]]$. This contradicts the assumption that the string $z[1..i-1]$ can be extended and that $f[1..i]$ is valid.

- If the algorithm exits the while loop with $j+1 < f[i]$, it means that no candidate $j'+1$ equal to $f[i]$ were found. This contradicts the fact that $f[i] = \beta_z[i]$ and that $f[1..i]$ is valid.

In both cases, no string $z[1..i-1]$, which border array is $f[1..i-1]$, can be extended, then the algorithm returns "f invalid at index i".

If $f[1..i]$ is valid then the algorithm does not stop at this index.

Assume now that at the beginning of iteration i we have: $z[1..i-1]$ is a string which border array is $f[1..i-1]$ and the while loop exits at index i with $j+1 = f[i]$.

Let us set $z[i] = z[f[i]]$. Then $f[i] = j+1$ is a candidate of $C(\beta_z, i)$ for which $z[j+1] = z[i]$ thus $z[1..j+1]$ is a border of $z[1..i]$. Assume that $z[1..j+1]$ is not the longest border of $z[1..i]$. Let $j'+1$ be the smallest candidate which is larger than $j+1$ and such that $z[1..j'+1]$ is a border of $z[1..i]$. Then $z[1..j+1]$ is the longest border of $z[1..j'+1]$ and we have $f[j'+1] = f[i]$ which means that the loop should have stop with this test and with $j+1 > f[i]$. This is a contradiction.

Thus the algorithm BABA runs as long as $f[1..i]$ is valid, it stops at index i and returns "f invalid at index i" if and only if f is valid up to index $i-1$ and is not at index i. □

The proof of Theorem 1 becomes then immediate.

Proof:[of Theorem 1] The point 1 (linearity of the algorithm BABA) comes from [4]. The other two points follow from propositions 1 and 2. □

Figures 3 and 4 show two examples.

i	1	2	3	4	5	6	7	8	9	10	11	12	symbols	candidates	valid
$x[i]$	a	b	a	a	b	a	b	a	a	b	a				
$f[i]$	0	0	1	1	2	3	2	3	4	5	6	?			
$k[i]$	1	2	1	2	2	1	2	1	2	2	1				
						a	b	a	a	b	a		b	7	YES
									a	b	a		a	4	YES
											a		b	2	NO
											ε		a	1	NO
													c	0	YES IF $s > 2$

Figure 3: The array $f[1..11]$ is a valid border array. The string $x[1..11]$ is the smallest string for which $f[1..11]$ is a valid border array. Then $x[1..11] =$ abaababaaba has borders abaaba, aba, a and ε of respective lengths 6, 3, 1 and 0 ($L(f, 11) = (6, 3, 1, 0)$). Thus the candidates for $f[12]$ are 7, 4, 2 and 1 ($C(f, 12) = (7, 4, 2, 1)$) together with 0 which is always a potential candidate. The values 7 and 4 are valid candidates. The value 2 is not valid since $f[7] = 2$ and 1 is not valid because $f[4] = 1$. The value 0 is a valid candidate if $s > 2$ because then $k[12]$ would be equal to $1 + k[f[12-1]+1] = 3$.

4 Conclusions

We presented in this article an elegant algorithm that verify, on-line and in linear time, if an integer array f is a border array of some string on a bounded size alphabet.

i	1	2	3	4	5	6	7	8	9	10	11	12	symbols	candidates	valid
$x[i]$	a	a	b	a	a	c	a	a	b	a	a				
$f[i]$	0	1	0	1	2	0	1	2	3	4	5	?			
$k[i]$	1	1	2	1	1	3	1	1	2	1	1				
							a	a	b	a	a		c	6	YES
										a	a		b	3	YES
											a		a	2	YES
											ε		a	1	NO
													d	0	YES IF $s > 3$

Figure 4: The array $f[1..11]$ is a valid border array. The string $x[1..11]$ is the smallest string for which $f[1..11]$ is a valid border array. Then $x[1..11]$ = aabaacaabaa has borders aabaa, aa, a and ε of respective lengths 5, 2, 1 and 0 ($L(f, 11) = (5, 2, 1, 0)$). Thus the candidates for $f[12]$ are 6, 3, 2 and 1 ($C(f, 12) = (6, 3, 2, 1)$) together with 0 which is always a potential candidate. The values 6, 3 and 2 are valid candidates. The value 1 is not valid since $f[2] = 1$. The value 0 is a valid candidate if $s > 3$ because then $k[12]$ would be equal to $1 + k[f[12 - 1] + 1] = 4$.

In the case where f is a border array, we are also capable to build a string x, on a minimal size alphabet for which f is the border array.

After studying the case of the "failure function" of the Morris and Pratt string matching algorithm, it is natural to ask the question if this work can be extended to the "failure function" of the Knuth, Morris and Pratt string matching algorithm [3].

References

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

[2] 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. Comput.* **42** (2002) to appear.

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

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