Discrete Applied Mathematics 163 (2014) 352–360

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

String matching with lookahead☆

Hannu Peltola*, Jorma Tarhio

Department of Computer Science and Engineering, Aalto University, P.O. Box 15400, FI-00076 Aalto, Finland

ARTICLE INFO

Article history:
Received 11 December 2011
Received in revised form 23 October 2013
Accepted 28 October 2013
Available online 16 November 2013

Keywords:
String matching
Bit-parallelism
BNDM
2-byte read
q-grams

ABSTRACT

Forward-SBNDM is a recently introduced variation of the BNDM algorithm for exact string matching. Forward-SBNDM inspects a 2-gram in the text such that the first character is the last one of an alignment window of the pattern and the second one is then outside the window. We present a generalization of this idea by inspecting several lookahead characters beyond an alignment window and integrate it with SBNDMq, a q-gram variation of BNDM. As a result, we get several new variations of SBNDMq. In addition, we introduce a greedy skip loop for SBNDM2. We tune up our algorithms and the reference algorithms with 2-byte read. According to our experiments, the best of the new variations are faster than the winners of recent algorithm comparisons for English, DNA, and binary data.

1. Introduction

After the advent of the Shift-Or [2] algorithm, bit-parallel string matching methods have gained more and more interest. The BNDM (Backward Nondeterministic DAWG Matching) algorithm [20] is a nice example of an elegant, compact, and efficient piece of code for exact string matching. BNDM simulates the nondeterministic finite automaton of the reverse pattern even without constructing the actual automaton.

SBNDM2 [6,11] is a simplified variation of BNDM. SBNDM2 starts processing of an alignment window of the pattern by reading two characters. Recently Faro and Lecroq [8] introduced Forward-SBNDM, a lookahead version of the SBNDM2 algorithm. Forward-SBNDM inspects a 2-gram (a string of two characters) where the latter text character follows an alignment window of the pattern. In this paper, we present a generalization of the lookahead idea and give new variations of SBNDMq [6], which is a q-gram extension of SBNDM2. In addition, we introduce a greedy skip loop for SBNDM2. Our point of view is the practical efficiency of exact string matching algorithms. According to our experiments, the best of the new variations are clearly faster than the winners of recent algorithm comparisons [6,9] for English, DNA, and binary data.

We use the following notations. Let a pattern $P = p_1p_2 \dots p_m$ and a text $T = t_1t_2 \dots t_n$ be two strings over a finite alphabet Σ. The task of exact string matching is to find all occurrences of P in T. Formally we search for all positions k such that $t_kt_{k+1} \dots t_{k+m-1} = p_1p_2 \dots p_m$. In the pseudocode of the algorithms we use the following notations of the programming language C: '|', '&', '∼', '<<', and '>>' represent bitwise operations OR, AND, one's complement, left shift, and right shift, respectively. The register width (or word size informally speaking) of a processor is denoted by w.

The rest of the paper is organized as follows. Since our work is based on SBNDMq and Forward-SBNDM, we start with presenting these algorithms in Section 2. In Section 3 we generalize Forward-SBNDM with wider lookahead and longer q-grams. In Section 4 the greedy skip loop is presented. Section 5 reviews the results of our experiments before concluding remarks in Section 6.

☆ Supported by the Academy of Finland (grant 134287).
* Corresponding author.
E-mail addresses: hpeltola@cs.hut.fi, hannu.peltola@aalto.fi (H. Peltola), jorma.tarhio@aalto.fi (J. Tarhio).

0166-218X/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.dam.2013.10.034

2. Previous algorithms

2.1. BNDM, SBNDM, and SBNDMq

In BNDM [20] the precomputed table B associates each character with a bit mask called an occurrence vector expressing its locations in the pattern. In each alignment window of the pattern, the algorithm reads the text from right to left until the whole pattern is recognized or the processed text string is not any factor (i.e. a substring) of the pattern. Between alignments, the algorithm shifts the pattern forward to the start position of the longest found prefix of the pattern, or if no prefix is found, over the current alignment window. With the bit-parallel shift-and technique, the algorithm maintains a state vector D, which has one in each position where the currently processed substring is a factor of the pattern. SBNDM [19,21] is a simplified version of BNDM. SBNDM does not explicitly care about prefixes, but shifts the pattern simply over the text character which caused D to become zero when the alignment is not a match.

SBNDMq [6] is a variation of SBNDM applying q-grams. In each alignment window, SBNDMq first processes q text characters $t_i, \ldots, t_{i+q-1}$ before testing the state vector D. If D is zero, this q-gram is not a factor of P, and then the pattern can be shifted forward $m-q+1$ positions. If D is not zero, a single character at a time is read to the left until D becomes zero, which means that the suffix of the alignment window is no more a factor of P, or an occurrence of the pattern has been found. The pseudocode of SBNDMq is shown as Algorithm 1. $F(i, q)$ on line 6 is a shorthand notation for the expression

$$B[t_i]\&(B[t_{i+1}] << 1)\& \cdots \&(B[t_{i+q-1}] << (q-1)).$$

Algorithm 1 SBNDMq ($P = p_1p_2 \cdots p_m$, $T = t_1t_2 \cdots t_n$)

```
Require: 1 ≤ q ≤ m ≤ w
    /* Preprocessing */
 1: for all c ∈ Σ do B[c] ← 0
 2: for j ← 1 to m do
 3:     B[p_j] ← B[p_j] | (1 << (m − j))
    /* Searching */
 4: i ← m − q + 1
 5: while i ≤ n − q + 1 do
 6:     D ← F(i, q)
 7:     if D ≠ 0 then
 8:         j ← i − (m − q + 1)
 9:         repeat
10:             i ← i − 1
11:             D ← (D << 1) & B[t_i]
12:         until D = 0
13:         if j = i then
14:             report occurrence at j + 1
15:             i ← i + s_0
16:     i ← i + m − q + 1
```

In the original BNDM, the inner loop also recognizes the prefixes of the pattern. The leftmost one of the found prefixes determines the next alignment window of BNDM. Like SBNDM, SBNDMq does not care about prefixes, but shifts the pattern over the text character which nullifies D when the alignment is not a match.

When an occurrence of the pattern is found, the shift is s_0, which corresponds to the distance to the leftmost prefix of the pattern in itself and which is easily computed from the pattern (see [6]). We skip the details, because a conservative value $s_0 = 1$ works well in practice. In the subsequent algorithms of this paper, we use the value $s_0 = 1$.

2.2. Forward-SBNDM

Forward-SBNDM, a lookahead version of SBNDM2, was introduced by Faro and Lecroq [8]. The idea of the algorithm is the following. The occurrence vectors B are obtained from the occurrence vectors of SBNDM2 by shifting them one position to the left and placing a set bit to the right end. As in SBNDM2, a 2-gram x_1x_2 is read before testing the state vector D. In SBNDM2, x_1x_2 is matched with the end of the pattern. In Forward-SBNDM (FSB for short), only x_1 is matched with the end of the pattern, and x_2 is a lookahead character following the current alignment window of the pattern. Note that $B[x_2]$ can nullify several bits of D, and therefore x_2 possibly enables longer shifts. The pseudocode of FSB is shown as Algorithm 2.

The basic shift of FSB is m positions, which is one more than in SBNDM2. Therefore FSB is faster than SBNDM2 for large alphabets [9]. Because the length of the occurrence vector B of each character is $m+1$ in FSB, the upper limit for the pattern length is thus $w-1$.

In a way, FSB is a cross of SBNDM2 and Sunday's QS [22]. QS was the first algorithm to use a lookahead character for shifting. Another famous algorithm using two lookahead characters is by Berry and Ravindran [3].