

EFFICIENT PATTERN MATCHING ON BINARY STRINGS*

SIMONE FARO

Dip. di Matematica e Informatica University of Rouen, LITIS EA 4108
Università di Catania, Italy

THIERRY LECROQ

Mont-Saint-Aignan Cedex, France

INTRODUCTION

We consider the problem of searching for a pattern p of length m in a text t of length n , with both strings are built over a binary alphabet, where each character of p and t is represented by a single bit. This is an interesting problem in computer science, since binary data are omnipresent in telecom and computer network applications. Many formats for data exchange between nodes in distributed computer systems as well as most network protocols use binary representations. Binary images often arise in digital image processing as masks or as the results of certain operations such as segmentation, thresholding and dithering.

Binary vectors are usually structured in blocks of k bits, typically bytes ($k = 8$), halfwords ($k = 16$) or words ($k = 32$), which can be processed at the cost of a single operation. If p is a binary string of length m we use the symbol $P[i]$ to indicate the $(i + 1)$ -th block of p and use $p[i]$ to indicate the $(i + 1)$ -th bit of p . If B is a block of k bits we indicate with symbol B_j the j -th bit of B , with $0 \leq j < k$.

(A) Patt	0	1	2	3	(C) Last
0	11001011	00101100	10110000		2
1	01100101	10010110	01011000		2
2	00110010	11001011	00101100		2
3	00011001	01100101	10010110		2
4	00001100	10110010	11001011	00000000	3
5	00000110	01011001	01100101	10000000	3
6	00000011	00101100	10110010	11000000	3
7	00000001	10010110	01011001	01100000	3

(B) Mask	0	1	2	3
0	11111111	11111111	11111000	
1	01111111	11111111	11111100	
2	00111111	11111111	11111110	
3	00011111	11111111	11111111	
4	00001111	11111111	11111111	10000000
5	00000111	11111111	11111111	11000000
6	00000011	11111111	11111111	11100000
7	00000001	11111111	11111111	11110000

In our high level model we define a sequence of several copies of the pattern memorized in the form of a matrix of bytes, $Patt$, of size $k \times (\lceil m/k \rceil + 1)$. Each row i of the matrix $Patt$ contains a copy of the pattern shifted by i position to the right. The i leftmost bits of the first byte remain undefined and are set to 0. Similarly the rightmost $k - ((m + i) \bmod k)$ bits of the last byte are set to 0. We define also a matrix of bytes, $Mask$, of size $k \times (\lceil m/k \rceil + 1)$, containing binary masks of length k . In particular a bit in the mask $Mask[i, h]$ is set to 1 if and only if the corresponding bit of $Patt[i, h]$ belongs to P . Finally we compute an array, $Last$, of size k where $Last[i]$ is defined to be $\lceil (m + i)/k \rceil$. More formally, for $0 \leq i < k$ and $0 \leq h < \lceil (m + i)/k \rceil$

$$Patt[i, h]_j = \begin{cases} p[kh - i + j] & \text{if } 0 \leq kh - i + j < m \\ 0 & \text{otherwise} \end{cases}$$

$$Mask[i, h]_j = \begin{cases} 1 & \text{if } 0 \leq kh - i + j < m \\ 0 & \text{otherwise} \end{cases}$$

The procedure used to precompute the tables defined above requires $\mathcal{O}(k \lceil m/k \rceil) = \mathcal{O}(m)$ time and $\mathcal{O}(m)$ extra-space. We check whether s is a valid shift without making use of bitwise operations but processing pattern and text byte by byte. In particular, for a given shift position s (so that $j = \lfloor s/k \rfloor$ and $i = (s \bmod k)$), we report a match if

$$Patt[i, h] = T[j + h] \ \& \ Mask[i, h], \text{ for } h = 0, 1, \dots, Last[i].$$

THE BINARY-HASH-MATCHING ALGORITHM

The BINARY-HASH-MATCHING algorithm is an adaptation of algorithms in the q -HASH family for exact pattern matching. The idea of the BINARY-HASH-MATCHING algorithm is to consider factors of the pattern of length q . If the pattern p is a binary string each sub-string of length q can be associated with its numeric value in the range $[0, 2^q - 1]$ without making use of the $hash$ function. In order to exploit the block structure of the text we take into account sub-strings of length $q = k$. Thus we define a function Hs , such that for each byte $0 \leq B < 2^k$

$$Hs(B) = \min \left\{ \{0 \leq u < m \mid p[m - u - k .. m - u - 1] \sqsupseteq B\} \cup \{m\} \right\}$$

If $B = p[m - k .. m - 1]$ then $Hs[B]$ is defined to be 0.

The preprocessing phase of the algorithm consists in computing the function Hs defined above and requires $\mathcal{O}(m + k2^{k-1})$ time complexity and $\mathcal{O}(m + 2^k)$ extra space. During the search phase the algorithm reads, for each shift position s of the pattern in the text, the block $B = t[s + m - q .. s + m - 1]$ of k bits. If $Hs(B) > 0$ then a shift of length $Hs(B)$ is applied. Otherwise, when $Hs(B) = 0$ the pattern p is naively checked in the text block by block. After the test an advancement of length $shift$ is applied where $shift$ is

$$\min \left\{ \{0 < u \leq m \mid p[m - u - k .. m - u - 1] \sqsupseteq p[m - k .. m - 1]\} \right\}.$$

The BINARY-HASH-MATCHING algorithm has a $\mathcal{O}(\lceil m/k \rceil n)$ time complexity and requires $\mathcal{O}(m + 2^k)$ extra space.

THE BINARY-SKIP-SEARCH ALGORITHM

The BINARY-SKIP-SEARCH algorithm is an adaptation of the SKIP-SEARCH algorithm, presented in 1998 by Charras, Lecroq and Pehoushek. The idea of the algorithm is straightforward.

For each possible block B of k bits, a bucket collects all pairs (i, h) in the table $Patt$ such that $Patt[i, h] = B$. When a block of bits occurs more times in the pattern, there are different corresponding pairs in the bucket of that block. Observe that for a pattern of length m there are $m - k + 1$ different blocks of length k corresponding to the blocks $Patt[i, h]$ such that $kh - i \geq 0$ and $k(h + 1) - i - 1 < m$.

However, to take advantage of the block structure of the text, we compute buckets only for blocks contained in the suffix of the pattern of length $m' = k \lfloor m/k \rfloor$. In such a way m' is a multiple of k and we could reduce to examine a block for each m'/k blocks of the text. Formally, for $0 \leq B < 2^k$

$$S_k[B] = \{(i, h) : (m \bmod k) \leq kh - i \leq m - k \ \& \ Patt[i, h] = B\}.$$

The preprocessing phase of the algorithm consists in computing the buckets for all possible blocks of k bits. The space and time complexity of this preprocessing phase is $\mathcal{O}(m + 2^k)$. The main loop of the search phase consists in examining every (m'/k) th text block. For each block $T[j]$ examined in the main loop, the algorithm inspects each pair (i, pos) in the bucket $S_k[T[j]]$ to obtain a possible alignment of the pattern against the text (line 6). For each pair (i, pos) the algorithm checks whether p occurs in t by comparing $Patt[i, h]$ and $T[j - pos + h]$, for $h = 0, \dots, Last[i]$ (lines 7-10). The BINARY-SKIP-SEARCH algorithm has a $\mathcal{O}(\lceil m/k \rceil n)$ quadratic worst case time complexity and requires $\mathcal{O}(m + 2^k)$ extra space.

EXPERIMENTAL RESULTS

We have computed experimental results in order to compare, in terms of running time and number of text character inspections, the following string matching algorithms: the BINARY-NAIVE algorithm (BNAIVE), the BINARY-BOYER-MOORE algorithm by Klein (BBM) presented in 2007, the BINARY-HASH-MATCHING algorithm (BHM), and the BINARY-SKIP-SEARCH algorithm (BSKS).

For experimental results on running times we have also tested the following algorithms for standard pattern matching: the q -HASH algorithm with $q = 8$ (HASH8) and the EXTENDED-BOM algorithm (EBOM). These algorithms have been tested on the same texts and patterns but in their standard form, i.e. each character is an ASCII value of 8-bit.

The algorithms have been tested on three $Rand(1/0)_\gamma$ problems, for $\gamma = 50, 70$ and 90 . Each $Rand(1/0)_\gamma$ problem consists of searching a set of 1000 random patterns of a given length in a random binary text of 4×10^6 bits. The distribution of characters depends on the value of the parameter γ : bit 0 appears with a percentage equal to $\gamma\%$. In the following tables, running times (on the left) are expressed in hundredths of seconds. Tables with the number of text character inspections are presented on the right. Best results are bold faced.

m	BNAIVE	BBM	BSKS	BHM	HASH8	EBOM	BNAIVE	BBM	BSKS	BHM
20	41.53	13.53	3.66	3.40	5.12	8.89	9.00	1.82	1.04	0.90
60	41.72	7.77	1.16	1.60	1.72	3.85	9.00	0.85	0.20	0.31
100	41.68	6.80	0.70	1.44	1.64	3.06	9.00	0.63	0.13	0.20
140	42.11	6.21	0.89	1.24	1.54	2.67	9.00	0.54	0.10	0.15
180	41.95	5.76	0.66	1.10	1.80	2.25	9.00	0.47	0.08	0.13
220	41.93	5.36	0.74	1.24	1.79	1.87	9.00	0.44	0.07	0.11
260	41.95	5.08	0.54	1.05	1.47	2.09	9.00	0.41	0.07	0.10
300	41.74	5.07	0.54	1.11	1.82	1.48	9.00	0.39	0.06	0.09
340	41.93	4.86	0.39	1.07	1.56	1.56	9.00	0.38	0.06	0.09
380	41.97	4.59	0.46	0.97	1.87	1.43	9.00	0.37	0.06	0.08
420	42.07	4.52	0.31	1.23	1.59	1.23	9.00	0.36	0.05	0.08
460	41.99	4.68	0.23	1.04	1.52	1.19	9.00	0.35	0.05	0.08
500	42.06	4.61	0.37	0.81	1.53	1.32	9.00	0.35	0.05	0.07

Experimental results for a $Rand(0/1)_{50}$ problem

m	BNAIVE	BBM	BSKS	BHM	HASH8	EBOM	BNAIVE	BBM	BSKS	BHM
20	43.26	17.25	4.01	4.21	4.86	10.92	9.41	2.27	1.12	1.01
60	43.15	10.26	1.66	2.09	2.03	4.27	9.40	1.14	0.29	0.38
100	43.80	8.44	1.60	2.26	1.95	2.54	9.38	0.89	0.21	0.26
140	43.70	8.13	1.28	1.61	1.52	2.68	9.38	0.77	0.18	0.21
180	43.22	7.37	1.02	1.67	2.08	2.33	9.37	0.71	0.17	0.18
220	43.29	6.82	1.08	1.34	1.94	2.50	9.39	0.65	0.16	0.16
260	42.93	6.67	1.07	1.53	1.79	1.94	9.38	0.61	0.15	0.15
300	43.66	6.46	0.89	1.22	1.59	1.94	9.39	0.59	0.15	0.14
340	43.53	6.35	0.97	1.23	1.28	1.86	9.39	0.57	0.15	0.13
380	43.76	6.15	0.70	1.42	1.31	1.65	9.38	0.55	0.14	0.12
420	43.29	6.03	0.85	1.34	1.67	1.48	9.38	0.54	0.14	0.12
460	43.45	6.00	0.92	1.27	1.37	1.43	9.38	0.53	0.14	0.11
500	43.31	6.00	0.70	1.28	1.41	1.48	9.37	0.51	0.14	0.11

Experimental results for a $Rand(0/1)_{70}$ problem

m	BNAIVE	BBM	BSKS	BHM	HASH8	EBOM	BNAIVE	BBM	BSKS	BHM
20	50.61	41.19	18.51	21.00	24.30	24.95	12.46	6.88	3.79	4.87
60	51.68	30.62	13.61	14.32	13.65	7.23	12.53	5.14	2.82	3.28
100	53.00	28.44	12.22	12.64	11.64	5.15	12.72	4.70	2.78	2.76
140	51.78	27.16	11.86	11.44	10.31	4.09	12.46	4.47	2.63	2.53
180	51.51	24.78	11.80	10.21	9.83	3.21	12.45	4.11	2.59	2.22
220	52.54	24.60	11.50	9.63	9.13	3.12	12.69	4.02	2.65	2.09
260	52.38	23.59	11.85	8.74	8.61	2.31	12.55	3.87	2.58	1.97
300	52.00	22.68	11.15	8.73	8.11	2.63	12.59	3.67	2.64	1.80
340	51.98	21.72	11.30	8.02	7.24	2.29	12.53	3.53	2.60	1.70
380	52.33	21.79	11.39	7.66	7.57	2.17	12.56	3.53	2.64	1.69
420	52.35	21.16	10.94	7.58	7.43	1.82	12.60	3.46	2.56	1.60
460	52.29	20.54	11.09	6.75	6.45	2.12	12.51	3.28	2.59	1.48
500	51.68	20.68	11.12	7.42	6.99	1.79	12.34	3.39	2.57	1.55

Experimental results for a $Rand(0/1)_{90}$ problem

* An extended version of this work appeared in S. Faro and T. Lecroq. Efficient pattern matching on binary strings. Report arXiv:0810.2390, Cornell University Library, 2008. <http://arxiv.org/abs/0810.2390>