Information Processing Letters 113 (2013) 604–608

Contents lists available at SciVerse ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Binary jumbled string matching for highly run-length compressible texts

Golnaz Badkobeh [a], Gabriele Fici [b], Steve Kroon [c], Zsuzsanna Lipták [d,*]

[a] Department of Informatics, King's College, London, UK
[b] Dipartimento di Matematica e Informatica, Università di Palermo, Italy
[c] Computer Science Division, Stellenbosch University, South Africa
[d] Dipartimento di Informatica, Università di Verona, Italy

ARTICLE INFO

Article history:
Received 23 November 2012
Received in revised form 10 May 2013
Accepted 14 May 2013
Available online 15 May 2013
Communicated by Ł. Kowalik

Keywords:
String algorithms
Data structures
Jumbled pattern matching
Parikh vectors
Prefix normal form
Run-length encoding

ABSTRACT

The Binary Jumbled String Matching Problem is defined as follows: Given a string s over $\{a, b\}$ of length n and a query (x, y), with x, y non-negative integers, decide whether s has a substring t with exactly x a's and y b's. Previous solutions created an index of size $O(n)$ in a pre-processing step, which was then used to answer queries in constant time. The fastest algorithms for construction of this index have running time $O(n^2/\log n)$ (Burcsi et al., 2010 [1]; Moosa and Rahman, 2010 [7]), or $O(n^2/\log^2 n)$ in the word-RAM model (Moosa and Rahman, 2012 [8]). We propose an index constructed directly from the run-length encoding of s. The construction time of our index is $O(n + \rho^2 \log \rho)$, where $O(n)$ is the time for computing the run-length encoding of s and ρ is the length of this encoding—this is no worse than previous solutions if $\rho = O(n/\log n)$ and better if $\rho = o(n/\log n)$. Our index L can be queried in $O(\log \rho)$ time. While $|L| = O(\min(n, \rho^2))$ in the worst case, preliminary investigations have indicated that $|L|$ may often be close to ρ. Furthermore, the algorithm for constructing the index is conceptually simple and easy to implement. In an attempt to shed light on the structure and size of our index, we characterize it in terms of the prefix normal forms of s introduced in Fici and Lipták (2011) [6].[1]

1. Introduction

Binary jumbled string matching is defined as follows: Given a string s over $\{a, b\}$ and a query vector (x, y) of non-negative integers x and y, decide whether s has a substring containing exactly x a's and y b's. If this is the case, we say that (x, y) *occurs* in s. The *Parikh set of* s, $\Pi(s)$, is the set of all vectors occurring in s.

For one query, the problem can be solved optimally by a simple sliding window algorithm in $O(n)$ time, where n is the length of the text. Here we are interested in the *indexing variant* where the text is fixed, and we expect a large number of queries. Recently, this problem and its variants have generated much interest [4,1,2,7,8,3,5]. The crucial observation is based on the following property of binary strings:

Interval Lemma. *(See [4].) Given a string s over $\Sigma = \{a, b\}$, $|s| = n$. For every $m \in \{1, \ldots, n\}$: if, for some $x < x'$, both $(x, m - x)$ and $(x', m - x')$ occur in s, then so does $(z, m - z)$ for all z, $x < z < x'$.*

It thus suffices to store, for every query length m, the minimum and maximum number of a's in all m-length substrings of s. This information can be stored in a linear size index, and now any query of the form (x, y) can be answered by looking up whether x lies between the

* Corresponding author.
E-mail addresses: golnaz.badkobeh@kcl.ac.uk (G. Badkobeh), gabriele.fici@unipa.it (G. Fici), kroon@sun.ac.za (S. Kroon), zsuzsanna.liptak@univr.it (Z. Lipták).

[1] A preliminary version of this paper was published on arXiv under the title *Binary jumbled string matching: Faster indexing in less space,* arXiv: 1206.2523v2, 2012.

0020-0190/$ – see front matter
http://dx.doi.org/10.1016/j.ipl.2013.05.007

minimum and maximum number of a's for length $m = x + y$. The query time is proportional to the time it takes to find $x + y$ in the index, which is constant in most implementations.

This index can be constructed naively in $O(n^2)$ time. In [1] and independently in [7], construction algorithms were presented with running time $O(n^2/\log n)$, using reduction to min-plus convolution. In the word-RAM model, the running time can again be reduced to $O(n^2/\log^2 n)$, using bit-parallelism [8]. More recently, a Monte Carlo algorithm with running time $O(n^{1+\varepsilon})$ was introduced [5], which constructs an approximate index allowing one-sided errors, with the probability of an incorrect answer depending on the choice of ε.

Any binary string s can be uniquely written in the form $s = a^{u_1}b^{v_1}a^{u_2}b^{v_2}\cdots a^{u_r}b^{v_r}$, where the u_i, v_i are non-negative integers, all non-zero except possibly u_1 and v_r. The *run-length encoding* of s is then defined as $rle(s) = (u_1, v_1, u_2, v_2, \ldots, u_r, v_r)$. This representation is often used to compress strings, especially in domains where long runs of characters occur frequently, such as the representation of digital images, multimedia databases, and time series.

In this paper, we present the Corner Index L which, for strings with good run-length compression, is much smaller than the linear size index used by all previous solutions. It is constructed directly from the run-length encoding of s, in time $O(\rho^2 \log \rho)$, where $\rho = |rle(s)|$. The Corner Index has worst-case size $\min(n, \rho^2)$ (measured in the number of entries, which fit into two computer words). We pay for this with an increase in lookup time from $O(1)$ to $O(\log |L|) = O(\log \rho)$.

In a recent paper [6], the *prefix normal forms* of a binary string were introduced. Given s of length n, $\mathrm{PNF}_a(s)$ is the unique string such that, for every $1 \leqslant m \leqslant n$, its m-length prefix has the same number of a's as the maximum number of a's in any m-length substring of s; $\mathrm{PNF}_b(s)$ is defined analogously. It was shown in [6] that two strings s and t have the same Parikh set if and only if $\mathrm{PNF}_a(s) = \mathrm{PNF}_a(t)$ and $\mathrm{PNF}_b(s) = \mathrm{PNF}_b(t)$. From this perspective, our index can be viewed as storing the run-length encodings of $\mathrm{PNF}_a(s)$ and $\mathrm{PNF}_b(s)$. This allows us a fresh view on the problem, and may point to a promising way of proving bounds on the index size. Moreover, our algorithm constitutes an improvement both for the computation and the testing problems on prefix normal forms (see [6]) whenever $rle(s)$ is short.

The construction time of $O(n + \rho^2 \log \rho)$, where $O(n)$ is for computing $rle(s)$ and $O(\rho^2 \log \rho)$ for constructing the Corner Index, is much better than the previous $O(n^2/\log n)$ time algorithms for strings with short run-length encodings, and no worse as long as $\rho = O(n/\log n)$. For strings with good run-length compression, the increase in lookup time from $O(1)$ to $O(\log |L|)$ is justified in our view by the reduced size and construction time of the new index. Finally, our algorithm is conceptually simple and easy to implement.

2. Preliminaries

A binary string $s = s_1 s_2 \cdots s_n$ is a finite sequence of characters from $\{a, b\}$. We denote the length of s by $|s|$. For two strings s, t, we say that t is a substring of s if there are indices $1 \leqslant i, j \leqslant |s|$ such that $t = s_i \cdots s_j$. If $i = 1$, then t is called a prefix of s. We denote by $|s|_a$ (resp. $|s|_b$) the number of a's (resp. b's) in s. The *Parikh vector* of s is defined as $p(s) = (|s|_a, |s|_b)$. We say that a Parikh vector q *occurs* in string s if s has a substring t such that $p(t) = q$. The *Parikh set* of s, $\Pi(s)$, is the set of all Parikh vectors occurring in s.

The Interval Lemma from the Introduction implies that, for any binary string s, there are functions F and f s.t.

$$(x, y) \text{ occurs in } s \text{ if and only if } f(x + y) \leqslant x \leqslant F(x + y), \tag{1}$$

namely, for $m = 0, \ldots, |s|$, $F(m) = \max\{x \mid (x, m - x) \in \Pi(s)\}$ and $f(m) = \min\{x \mid (x, m - x) \in \Pi(s)\}$. This can be stated equivalently in terms of the minimum and maximum number of b's in all substrings containing a *fixed number* i of a's. Let us denote by $bmin(i)$ (resp. $bmax(i)$) the minimum (resp. maximum) number of b's in a substring containing exactly i a's. Then

$$(x, y) \text{ occurs in } s \text{ if and only if } bmin(x) \leqslant y \leqslant bmax(x). \tag{2}$$

The table of functions F and f in (1) is the index used in most algorithms for binary jumbled string matching [2,7,8], while that of functions $bmin$ and $bmax$ in (2) was used in [3]. Even though the latter is always smaller, both are linear size in n. Note that one table can be computed from the other in linear time (e.g. $bmin(i) = \min\{m \mid F(m) = i\} - i$).

Table 1
Functions $bmin$ and $bmax$ for the string $s = aababababbaaabbaabbb$.

i	0	1	2	3	4	5	6	7	8	9
$bmin(i)$	0	0	0	0	2	2	4	4	6	6
$bmax(i)$	3	3	5	5	5	7	8	9	9	9

Example 1. Let $s = aababababbaaabbaabbb$. Then $(3, 3)$ occurs in s while $(5, 1)$ does not. We have $F(6) = 4$ and $f(6) = 2$, $bmin(3) = 0$ and $bmax(3) = 5$. We give the full table of values of the two functions $bmin$ and $bmax$ in Table 1.

3. The Corner Index

In Fig. 1, we plot both functions $bmin$ and $bmax$ for our example string. The x-axis denotes the number of a's and the y-axis the number of b's. It follows from (2) that the integer points within the shaded area correspond to the Parikh set of s. The crucial observation is: Since both functions $bmin$ and $bmax$ are monotonically increasing step functions, it is sufficient to store those points where they increase. These points are specially marked in Fig. 1.

Example 2. In our example, these points are, for $bmin$: $\{(3, 0), (5, 2), (7, 4), (9, 6)\}$, and for $bmax$: $\{(0, 3), (2, 5), (5, 7), (6, 8), (7, 9)\}$.

Definition 1. We define the *Corner Index* for the Parikh set of a given binary string s as two ordered sets $L_{\min}$ and $L_{\max}$, where