Theoretical Computer Science 635 (2016) 94–101

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Note

Document retrieval with one wildcard ☆

Moshe Lewenstein [a], J. Ian Munro [b], Yakov Nekrich [b], Sharma V. Thankachan [c,*]

[a] Department of Computer Science, Bar-Ilan University, Israel
[b] Cheriton School of Computer Science, University of Waterloo, Canada
[c] School of Computational Science and Engineering, Georgia Institute of Technology, USA

ARTICLE INFO

Article history:
Received 1 October 2014
Received in revised form 4 September 2015
Accepted 18 May 2016
Available online 19 May 2016
Communicated by G.F. Italiano

Keywords:
Document retrieval
String searching
Wildcards
Compressed data structures

ABSTRACT

In this paper we extend several well-known document listing problems to the case when documents contain a substring that approximately matches the query pattern. We study the scenario when the query string can contain a wildcard symbol that matches any alphabet symbol; all documents that match a query pattern with one wildcard must be enumerated. We describe a linear space data structure that reports all documents containing a substring P in $O(|P| + \sigma\sqrt{\log\log\log n} + \mathtt{docc})$ time, where σ is the alphabet size and $\mathtt{docc}$ is the number of listed documents. We also describe a succinct solution for this problem, as well as a solution for an extension of this problem. Furthermore our approach enables us to obtain an $O(n\sigma)$-space data structure that enumerates all documents containing both a pattern P_1 and a pattern P_2 in the special case when P_1 and P_2 differ in one symbol.

Published by Elsevier B.V.

1. Introduction

The ever-growing abundance of data in databases presents unique challenges to the search community that need to be addressed with both speed and space considerations taken into account. For many years the search community was focused on offline problems. In classical *pattern matching* one is given a pattern P and text T and is required to find all occurrences of P in T. Numerous solutions for pattern matching have addressed many different variations of this problem, e.g. [2,3]. In *text indexing* one is given a text T to preprocess for subsequent pattern queries P for which one is required to find all occurrences of P in T. The practical need for text indexing was already recognized long ago. The explosion in data that stemmed from the onset of the Internet and applications in computational biology set the quest for more succinct data structures and posed new challenges in this important research area. Several broad research directions are related to extensions of the standard indexing problem.

First, we can keep a collection of strings (or documents) in the data structure and ask queries about documents that contain a query string. For instance, we may wish to find all documents that contain P at least once. This problem, called document listing problem, is more difficult than the standard indexing because we are interested in reporting each document only once even if it contains multiple occurrences of P. More complicated queries, that are relevant for document retrieval applications, were also studied [4–6]. We may be interested in documents that contain P at least k times for a

☆ Early parts of this work appeared in MFCS 2014 [1]. Work is supported by NSERC of Canada and the Canada Research Chairs program.
* Corresponding author.
E-mail addresses: moshe@macs.biu.ac.il (M. Lewenstein), imunro@uwaterloo.ca (J.I. Munro), ynekrich@uwaterloo.ca (Y. Nekrich), sharma.thankachan@gmail.com (S.V. Thankachan).

http://dx.doi.org/10.1016/j.tcs.2016.05.024
0304-3975/Published by Elsevier B.V.

query parameter k, or the k documents in which P occurs most frequently, etc. We refer to e.g. [7,8] for an overview of problems and results in this area. Second, we are frequently interested in substrings of T that approximately match the query string P. Approximate string matching is important for computational biology applications. There are several definitions of approximate matching. In wildcard pattern matching, considered in this paper, the query string P contains a symbol ϕ that matches any alphabet symbol. Indexing for patterns with wildcards and approximate indexing was considered in e.g., [9–19].

In this paper we consider generalizations of the document listing problem for the case when the query pattern P contains one wildcard symbol ϕ that matches any alphabet symbol, and present a linear space indexing solution.

Previous and related results. The suffix tree provides a linear space solution for the original indexing problem. Using suffix trees, we can report all $\mathtt{occ}$ occurrences of a substring P in optimal $O(|P| + \mathtt{occ})$ time. Henceforth $|S|$ denotes the length of string S. Indexing for patterns with wildcards appears to be significantly more difficult than the standard indexing. Even in the simplest scenario when P contains only one wildcard, we either need more space or have to spend more time to answer a query.

The naive solution is to store all possible combinations of suffixes resulting from replacing an arbitrary symbol by a wildcard. The space usage of this naive solution is $O(n^2)$. Cole et al. [11] described an elegant data structure that significantly reduced the space usage of this naive approach.[1] Their data structure uses $O(n \log n)$ space and answers queries in $O(|P| + \mathtt{occ})$ time. Another solution of one-wildcard indexing is based on reducing this problem to range reporting. This approach, introduced in [9] and used in [21], needs $O(n \log^{\epsilon} n)$ space to achieve optimal query time (see also [22]).

Cole et al. [11] described another data structure that uses $O(n \log n)$ space and answers queries in $O(|P| + \sigma \log\log n + \mathtt{occ})$ time. Here and further σ denotes the alphabet size. Bille et al. [10] showed how to reduce the space usage to $O(n)$ and obtained the first linear space data structure. Very recently, Lewenstein et al. [14] described a linear space data structure that supports queries in $O(|P| + \sigma\sqrt{\log\log\log n} + \mathtt{occ})$ time. This is the fastest currently known data structure that uses linear space.

Matias et al. [23] and later Muthukrishnan [24] addressed the document listing problem. The problem is to index a collection $\mathcal{D} = \{d_1, d_2, .., d_D\}$ of D documents, such that whenever a pattern P comes as a query, we can report all those $\mathtt{docc}$ unique documents containing P. The solution of Muthukrishnan [24] uses $O(n)$ space and optimal $O(|P| + \mathtt{docc})$ query time. In previous works on document listing it was assumed that the pattern P contains alphabet symbols only.

Our results. We consider the document listing problem in the case when the pattern P contains one wildcard symbol. Our solution uses linear space and reports all $\mathtt{docc}$ documents that contain a substring matching $P_1\phi P_2$ in $O(|P_1| + |P_2| + \sigma\sqrt{\log\log\log n} + \mathtt{docc})$ time, where P_1 and P_2 are arbitrary strings containing alphabet symbols only. Thus we match the complexity of the best currently known linear space data structure for reporting all occurrences of $P_1\phi P_2$. We also describe a compact data structure that uses $|\mathsf{CSA}| + O(n)$ bits of space, where $|\mathsf{CSA}|$ denotes the space (in bits) needed to store a compact suffix array. We also consider the following extension of this problem: each document d_i is associated with a *rank* (denoted by $docrank(i)$), the query consists of a string $P_1\phi P_2$ and an interval $[\alpha, \beta]$, and we are required to output only documents d_i containing $P_1\phi P_2$ and satisfying $docrank(i \in [\alpha, \beta]$. Our solution requires $O(n)$ space and $O(|P_1| + |P_2| + (\sigma + k + \log n)\log^{1+\varepsilon} n)$ query time, where k is the output size.

The main algorithmic challenge of document listing in the one-wildcard scenario is the requirement that each document must be reported $O(1)$ times. Our solution is based on a novel notion of unique prefixes, defined in Section 3. We believe this idea to be of independent interest and that it can find further applications.

As an additional bonus, we describe an efficient solution for a special case of the two-pattern document listing problem [25–28]. In the two-pattern document listing a query consists of two patterns P_1 and P_2; we want to report all documents that contain both P_1 and P_2. This problem is known to be very hard; the best known solution needs $\tilde{O}(n^2)$ space to achieve $\tilde{O}(|P_1| + |P_2| + \mathtt{docc})$ query time.[2] We consider the special case of this problem when P_1 and P_2 mismatch only at one position. For this special case, we describe an $O(n\sigma)$ space data structure that lists all $\mathtt{docc}$ relevant documents in optimal $O(|P_1| + |P_2| + \mathtt{docc})$ time. This result also relies on the notion of unique prefixes.

2. Document listing without wildcards

In this section, we briefly describe the indexing solution for the document listing problem proposed by Muthukrishnan [24]. Notice that the query string P does not contain any wildcard character in this case. The data structure consists of three main components: a generalized suffix tree GST, an array $E[1..n]$ and a range minimum query data structure over E [29].

The generalized suffix tree GST of our document collection $\mathcal{D} = \{d_1, d_2, .., d_D\}$ is essentially a suffix tree [30] of the text $\mathsf{T}[1..n] = d_1 d_2 \ldots d_D$ obtained by concatenating all documents in $\mathcal{D}$. We assume that the last character of each document is $\$ \notin \Sigma$, a special symbol that does not appear anywhere else in any document in $\mathcal{D}$. Each substring $\mathsf{T}[i..n]$, with $i \in [1, n]$, is called a *suffix* of T. The GST of $\mathcal{D}$ is a lexicographic arrangement of all these n suffixes in a compact trie structure, where the ith leftmost leaf represents the ith lexicographically smallest suffix. Let $str(u, w)$ be the string obtained by concatenation

[1] In = [11] the situation when the query string contains $k > 1$ wildcard symbols was also considered. We refer to e.g. [20] and references therein for an overview of previous results on general indexing with wildcards.

[2] The notation $\tilde{O}$ ignores poly-logarithmic factors. Precisely, $\tilde{O}(f(n)) \equiv O(f(n)\log^{O(1)} n)$.