Theoretical Computer Science 557 (2014) 120–127

Contents lists available at ScienceDirect

Theoretical Computer Science

www.elsevier.com/locate/tcs

Note

Less space: Indexing for queries with wildcards ☆

Moshe Lewenstein [a], J. Ian Munro [b], Venkatesh Raman [c], Sharma V. Thankachan [d,*]

[a] Bar-Ilan University, Israel
[b] University of Waterloo, Canada
[c] The Institute of Mathematical Sciences, India
[d] Georgia Institute of Technology, USA

ARTICLE INFO

Article history:
Received 14 May 2014
Received in revised form 5 August 2014
Accepted 1 September 2014
Available online 8 September 2014
Communicated by G. Ausiello

Keywords:
String indexing
Data structures
Wildcards
Range searching
Suffix trees

ABSTRACT

Text indexing is a fundamental problem in computer science, where the task is to index a given text (string) $T[1..n]$, such that whenever a pattern $P[1..p]$ comes as a query, we can efficiently report all those locations where P occurs as a substring of T. In this paper, we consider the case when P contains wildcard characters (which can match with any other character). The first non-trivial solution for the problem was given by Cole et al. [11], where the index space is $O(n\log^k n)$ words or $O(n\log^{k+1} n)$ bits and the query time is $O(p+2^h\log\log n+occ)$, where k is the maximum number of wildcard characters allowed in P, $h\leq k$ is the number of wildcard characters in P and occ represents the number of occurrences of P in T. Even though many indexes offering different space-time trade-offs were later proposed, a clear improvement on this result is still not known. In this paper, we first propose an $O(n\log^{k+\epsilon} n)$ bits index achieving the same query time as the of Cole et al.'s index, where $0<\epsilon<1$ is an arbitrary small constant. Then we propose another index of size $O(n\log^k n\log\sigma)$ bits, but with a slightly higher query time of $O(p+2^h\log n+occ)$, where σ denotes the alphabet set size.

We also study a related problem, where the task is to index a collection of documents (of n characters in total) so as to find the number of distinct documents containing a query pattern P. For the case where P contains at most a single wildcard character, we propose an $O(n\log n)$-word index with optimal $O(p)$ query time.

Published by Elsevier B.V.

1. Introduction and related work

Text indexing is a fundamental problem in computer science, where the task is to index a given text (string) $T[1..n]$, such that whenever a pattern $P[1..p]$ comes as a query, we can efficiently report all those locations where P occurs as a substring of T. The classic data structures for solving this problem are suffix trees [32] and suffix arrays [24]. Both these linear space ($O(n\log n)$ bits) structures can perform pattern matching in optimal $O(p+occ)$ and $O(p+\log n+occ)$ time respectively, where occ is the number of occurrences of P in T.[1] Approximate string matching and wildcard matching

☆ Early part of this work appeared in ISAAC 2013 [21]. Work 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. Ian Munro), vraman@imsc.res.in (V. Raman), sharma.thankachan@gmail.com (S.V. Thankachan).
[1] All logarithms in this article are base 2.

http://dx.doi.org/10.1016/j.tcs.2014.09.003
0304-3975/Published by Elsevier B.V.

are two of the natural extensions of this problem, and have been studied extensively [2,11,15,8,18,30,31,14,6,19,20]. These problems have several applications in information retrieval, bioinformatics, data mining, and internet traffic analysis [7,13].

The focus of this paper is on the following problem: index T for handling matching of a query pattern P with at most k wildcards. A wildcard, also known as don't care character, represented by ϕ, can match with any other character in the alphabet set Σ of size σ. Therefore, the pattern P can be written as $P_0\phi P_1\phi..P_{h-1}\phi P_h$, the concatenation of substrings $P_0, P_1, ...P_{h-1}, P_h$ separated by ϕ and $h \leq k$ is the number of wildcards in P. The first non-trivial solution for this problem was proposed by Cole et al. [11], where the index space is $O(n\log^k n)$ words or $O(n\log^{k+1} n)$ bits and query time is $O(p + 2^h \log\log n + occ)$. Recently, Bille et al. [6] proposed an index, which is a generalization of Cole et al.'s index. The space and query time are $O(n\log n \log_\beta^{k-1} n)$ words and $O(p + \beta^h \log\log n + occ)$ respectively, where $2 \leq \beta \leq \sigma$. Note that Cole et al.'s [11] result can be obtained by substituting $\beta = 2$. Bille et al. [6] also proposed an optimal $O(p+occ)$ time index of space $O(n\sigma^{k^2}\log^k \log n)$ words. Another space-efficient index of $O(n\log n)$ words proposed by Cole et al. [11] can answer this query in $O(p + \sigma^h \log\log n + occ)$ time, and is improved to $O(n)$ words without affecting the query time [6]. Recently, Lewenstein et al. improved the query time to $O(p + \sigma^h\sqrt{\log\log\log n} + occ)$ [23]. Several other linear space structures also exist in literature, such as the ones by Iliopoulos and Rahman [25], and Lam et al. [18]. However, these indexes take $\Theta(nh)$ worst case time for answering the query. Despite of all these continued efforts, a clear improvement over the seminal result by Cole et al. [11] (i.e., $O(n\log^{k+1} n)$ bits and $O(p + 2^h \log\log n + occ)$ time) is still not known.

In this paper, we describe two results. The first one is an $O(n\log^{k+\epsilon} n)$ bits index with $O(p + 2^h \log\log n + occ)$ query time, where $0 < \epsilon < 1$ is an arbitrary small constant. The second one is an $O(n\log^k n \log\sigma)$ bits index, but with a slightly worse query time of $O(p + 2^h \log n + occ)$, where $\Sigma = [\sigma]$ denotes the alphabet set. Notice that our first result is a clear improvement over the earlier result by Cole et al., whereas the second one provides another space-time trade-off for this problem when the alphabet set is small.

Another problem that is strongly connected to the problem under consideration is to index the text wildcards. This was solved in Cole et al. [11] as well. However, for this case, better solutions have appeared in a succession of papers and indexes with succinct space and competitive query time [18,30,31,14] are available in the literature. Yet another related problem is that of indexing with gaps. Gaps are essentially longer wildcards. In [16] an index was proposed supporting queries of patterns containing one gap with a predefined length. This result builds on the result of [2]. The case of one gap was further improved by Bille et al. [5] with optimal query time. In [19] results were shown for the case when there is a larger number of gaps.

A related problem we study in this paper is the document-frequency computation (or document counting) of patterns with wildcards. Let $\mathcal{D} = \{d_1, d_2, d_3..\}$ be a collection of (string) documents of total length n. The document-frequency of an input pattern P is the number of distinct documents in $\mathcal{D}$ containing P as a substring. Such queries (without wildcards) can be answered in optimal $O(p)$ time using a simple linear space data structure of $O(n)$ words (which is essentially a suffix tree with constant amount of information augmented with its internal nodes). However the problem becomes more challenging if the pattern or documents contain wildcards or if we allow a bounded error in the substring matching. We consider the case when the query pattern P contains a single wildcard character, and propose an $O(n\log n)$-word data structure with optimal $O(p)$ query time. See [22] for a recent result on the reporting version of this problem, where the index space is $O(n)$ words and the query time is $O(p + \sigma\sqrt{\log\log\log n} + ndoc)$. Here $ndoc$ is the output size.

Outline Section 2 gives the preliminaries. In Section 3, we describe a classical framework for the case where $k = 1$ and then in Section 4, the framework by Cole et al.'s for $k \geq 1$. Section 5 describes our space-efficient data structures. Section 6 describes our result on indexing documents and we finally conclude in Section 7.

2. Preliminaries

2.1. Suffix trees and suffix arrays

Suffix trees [32] and suffix arrays [24] are two classic data structures for online pattern matching queries. For a text $T[1..n]$, substring $T[i..n]$, with $i \in [1, n]$, is called a suffix of T. The suffix tree for T 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. For each node v in the suffix tree, we use $path(v)$ to denote the concatenation of edge labels along the path from the root to v. For any pattern P (of length p), the locus of P in the suffix tree is defined to be the highest node v (i.e., the closest node from the root) such that P is a prefix of $path(v)$ and can be computed in $O(p)$ time.

The suffix array $SA[1..n]$ is an array of length n, such that $SA[i]$ is the starting position of the ith lexicographically smallest suffix of T. The suffix array has an important property that the starting positions of all suffixes with the same prefix are always stored in a contiguous region in SA. Based on this property, the suffix range of a pattern P in SA is defined as the maximal range $[sp, ep]$ such that for all $j \in [ep, ep], SA[j]$ is the starting point of a suffix of T with P as a prefix. In other words, the suffix range of a string represents the set of leaves in the subtree of its locus node in suffix tree. We also define its inverse, SA^{-1} to be an array such that $SA[i] = j$ if and only if $SA^{-1}[j] = i$. Both suffix trees and suffix arrays (along with an auxiliary data structure called LCP array) take $(n\log n)$ bits space and can perform pattern matching in optimal $O(p + occ)$ and $O(p + \log n + occ)$ time respectively, where occ is the number of occurrences of P in T.