ARTICLE IN PRESS

Discrete Applied Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Discrete Applied Mathematics

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

Computing covers using prefix tables☆

Ali Alatabbi [a,*], M. Sohel Rahman [b], W.F. Smyth [a,c,d]

[a] Department of Informatics, King's College London, United Kingdom

[b] AℓEDA Group, Department of Computer Science & Engineering (CSE), Bangladesh University of Engineering & Technology (BUET), Dhaka-1205, Bangladesh

[c] Algorithms Research Group, Department of Computing & Software, McMaster University, Canada

[d] School of Engineering & Information Technology, Murdoch University, Western Australia, Australia

ARTICLE INFO

Article history:
Received 17 October 2014
Received in revised form 17 February 2015
Accepted 11 May 2015
Available online xxxx

Keywords:
Strings
Prefix tables
Covers
Cover array
Indeterminate strings
Linear time
Algorithm

ABSTRACT

An ***indeterminate string*** $\boldsymbol{x} = \boldsymbol{x}[1..n]$ on an alphabet Σ is a sequence of nonempty subsets of Σ; $\boldsymbol{x}$ is said to be ***regular*** if every subset is of size one. A proper substring $\boldsymbol{u}$ of regular $\boldsymbol{x}$ is said to be a ***cover*** of $\boldsymbol{x}$ iff for every $i \in 1..n$, an occurrence of $\boldsymbol{u}$ in $\boldsymbol{x}$ includes $\boldsymbol{x}[i]$. The ***cover array*** $\boldsymbol{\gamma} = \boldsymbol{\gamma}[1..n]$ of $\boldsymbol{x}$ is an integer array such that $\boldsymbol{\gamma}[i]$ is the longest cover of $\boldsymbol{x}[1..i]$. Fifteen years ago a complex, though nevertheless linear-time, algorithm was proposed to compute the cover array of regular $\boldsymbol{x}$ based on prior computation of the border array of $\boldsymbol{x}$. In this paper we first describe a linear-time algorithm to compute the cover array of regular $\boldsymbol{x}$ based on the prefix table of $\boldsymbol{x}$. We then extend this result to indeterminate strings.

1. Introduction

The idea of a ***quasiperiod*** or ***cover*** of a string $\boldsymbol{x}$ was introduced almost a quarter-century ago by Apostolico & Ehrenfeucht [4]: a proper substring $\boldsymbol{u}$ of $\boldsymbol{x}$ such that every position in $\boldsymbol{x}$ lies within an occurrence of $\boldsymbol{u}$. Thus, for example, $\boldsymbol{u} = aba$ is a cover of $\boldsymbol{x} = ababaababa$. In [5] a linear-time algorithm was described to compute the shortest cover of $\boldsymbol{x}$; this contribution was followed by linear-time algorithms to compute

- the shortest cover of every prefix of $\boldsymbol{x}$ [9];
- all the covers of $\boldsymbol{x}$ [17,18];
- all the covers of every prefix of $\boldsymbol{x}$ [16].

A ***border*** of a string $\boldsymbol{x}$ is a possibly empty proper prefix of $\boldsymbol{x}$ that is also a suffix of $\boldsymbol{x}$. (Thus a cover of $\boldsymbol{x}$ is necessarily also a border of $\boldsymbol{x}$.) In the ***border array*** $\boldsymbol{\beta} = \boldsymbol{\beta}[1..n]$ of the string $\boldsymbol{x} = \boldsymbol{x}[1..n]$, $\boldsymbol{\beta}[i]$ is the length of the longest border of $\boldsymbol{x}[1..i]$. Since for $\boldsymbol{\beta}[i] \neq 0$, $\boldsymbol{\beta}[\boldsymbol{\beta}[i]]$ is the length of a border of $\boldsymbol{x}$ as well as the length of the longest border of $\boldsymbol{x}[1..\boldsymbol{\beta}[i]]$ [2,20], it follows that $\boldsymbol{\beta}$ provides all the borders of every prefix of $\boldsymbol{x}$. For example:

	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x} =$	a	b	a	b	a	b	a	a	b	a
$\boldsymbol{\beta} =$	0	0	1	2	3	4	5	1	2	3

(1)

☆ This work was supported in part by the Natural Sciences & Engineering Research Council of Canada.

* Corresponding author.
E-mail addresses: ali.alatabbi@kcl.ac.uk (A. Alatabbi), msrahman@cse.buet.ac.bd (M. Sohel Rahman), smyth@mcmaster.ca (W.F. Smyth).

http://dx.doi.org/10.1016/j.dam.2015.05.019
0166-218X/

Please cite this article in press as: A. Alatabbi, et al., Computing covers using prefix tables, Discrete Applied Mathematics (2015), http://dx.doi.org/10.1016/j.dam.2015.05.019

As shown in [16], the ***cover array*** γ has a similar cascading property, giving the lengths of all the covers of every prefix of $\boldsymbol{x}$ in a compact form:

	1	2	3	4	5	6	7	8	9	10
$\gamma =$	0	0	0	2	3	4	5	0	0	3

Here $\boldsymbol{x}[1..7]$ has covers $\boldsymbol{u}_1 = \boldsymbol{x}[1..5] = ababa$ and $\boldsymbol{u}_2 = \boldsymbol{x}[1..3] = aba$, while the entire string $\boldsymbol{x}$ has cover $\boldsymbol{u}_2$. The main result of [16] is an algorithm that computes $\gamma = \gamma[1..n]$ from $\boldsymbol{\beta} = \boldsymbol{\beta}[1..n]$ in $\Theta(n)$ time, while making no reference to the underlying string $\boldsymbol{x}$.

The results outlined above all apply to a ***regular string*** — that is, a string $\boldsymbol{x}$ such that each entry $\boldsymbol{x}[i]$ is constrained to be a one-element subset of a given set Σ called the ***alphabet***. In this paper we show how to extend these ideas and algorithms to an ***indeterminate string*** $\boldsymbol{x}$ — that is, such that each $\boldsymbol{x}[i]$ can be any nonempty subset of Σ. Observe that every regular string is indeterminate.

The idea of an indeterminate string was first introduced in [12], then studied further in the 1980s as a "generalized string" [1]. Over the last 15 years Blanchet-Sadri has written numerous papers on the properties of "strings with holes" (each $\boldsymbol{x}[i]$ is either a one-element subset of Σ or Σ itself), together with a monograph on the subject [8]; while other authors have studied indeterminate strings in their full generality, together with related algorithms [6,19,14,15,21–23,10]. In the specific context of this paper, Voráček & Melichar [24] have done pioneering work on the computation of covers and related structures in generalized strings using finite automata.

For indeterminate strings, equality of letters is replaced by the idea of a "match" [14]: $\boldsymbol{x}[i]$ ***matches*** $\boldsymbol{x}[j]$ (written $\boldsymbol{x}[i] \approx \boldsymbol{x}[j]$) if and only if $\boldsymbol{x}[i] \cap \boldsymbol{x}[j] \neq \emptyset$, while $\boldsymbol{x} \approx \boldsymbol{y}$ if and only if $|\boldsymbol{x}| = |\boldsymbol{y}|$ and corresponding positions in $\boldsymbol{x}$ and $\boldsymbol{y}$ all match. It is important to note that matching is nontransitive: $b \approx \{b, c\} \approx c$, but $b \not\approx c$.

It is [10] that provides the point of departure for our contribution, as we now explain. The ***prefix table*** $\boldsymbol{\pi} = \boldsymbol{\pi}[1..n]$ of $\boldsymbol{x}[1..n]$ is an integer array such that $\boldsymbol{\pi}[1] = n$ and, for every $i \in 2..n$, $\boldsymbol{\pi}[i]$ is the length of the longest substring occurring at position i of $\boldsymbol{x}$ that matches a prefix of $\boldsymbol{x}$. Thus, for our example (1):

	1	2	3	4	5	6	7	8	9	10
$\boldsymbol{x} =$	a	b	a	b	a	b	a	a	b	a
$\boldsymbol{\pi} =$	10	0	3	0	3	0	1	3	0	1

It turns out [7] that the prefix table and the border array are "equivalent" for regular strings; that is, each can be computed from $\boldsymbol{x}$ in linear time, and each can be computed from the other, without reference to $\boldsymbol{x}$, also in linear time. However, for indeterminate strings, this is not true: the prefix table continues to determine all the borders of every prefix of $\boldsymbol{x}$, while the border array, due to the intransitivity of matching, is no longer reliable in identifying borders shorter than the longest one. Consider, for example:

	1	2	3
$\boldsymbol{x} =$	a	$\{a, b\}$	b
$\boldsymbol{\beta} =$	0	1	2

Here $\boldsymbol{x}$ does not have a border of length $\boldsymbol{\beta}[\boldsymbol{\beta}[3]] = 1$; on the other hand, $\boldsymbol{\pi} = 320$ correctly identifies all the borders of every prefix of $\boldsymbol{x}$.

Moreover, it was shown in [10] that every ***feasible*** array – that is, every array $\boldsymbol{y} = \boldsymbol{y}[1..n]$ such that $\boldsymbol{y}[1] = n$ and for every $i \in 2..n$, $\boldsymbol{y}[i] \in 0..n-i+1$ – is a prefix table of some (indeterminate) string. Thus there exists a many-many correspondence between all possible prefix tables and all possible indeterminate strings. Furthermore, [21] describes an algorithm to compute the prefix table of any indeterminate string, while [3] gives an algorithm to compute a lexicographically least indeterminate string corresponding to a given prefix table.

At this point let us discuss our motivation more precisely. First, realize that to exploit the fullest functionality of a border array of an indeterminate string we need to resort to the extended definition of the border array which in fact requires quadratic space [14,19,6]: unlike the border array of a regular string, which is a simple array of integers, the border array of an indeterminate string is an array of lists of integers. Here at each position, the list gives all possible borders for that prefix. On the other hand, the prefix array, even for the indeterminate string, remains a simple one-dimensional array, just as for a regular string. It thus becomes of interest to make use of the prefix table rather than the border array whenever possible, in order to extend the scope of computations to indeterminate strings.

In Section 2 of this paper, we describe a linear-time algorithm to compute the cover array γ of a regular string $\boldsymbol{x}$ directly from its prefix table $\boldsymbol{\pi}$. Then, Section 3 describes a limited extension of this algorithm to indeterminate strings. Finally, Section 4 outlines future research directions, especially making use of prefix tables to extend the utility and applicability of other data structures to indeterminate strings.

2. Prefix-to-cover for a regular string

In this section we describe our basic $\Theta(n)$-time Algorithm PCR to compute the cover array $\gamma = \gamma[1..n]$ of a regular string $\boldsymbol{x} = \boldsymbol{x}[1..n]$ directly from its prefix table $\boldsymbol{\pi} = \boldsymbol{\pi}[1..n]$. In fact, as noted in the Introduction, γ actually provides all the covers of every prefix of $\boldsymbol{x}$. Central to our algorithm are the following definitions:

Please cite this article in press as: A. Alatabbi, et al., Computing covers using prefix tables, Discrete Applied Mathematics (2015), http://dx.doi.org/10.1016/j.dam.2015.05.019