Information Processing Letters 115 (2015) 655–659

Contents lists available at ScienceDirect

Information Processing Letters

www.elsevier.com/locate/ipl

Constructing LZ78 tries and position heaps in linear time for large alphabets

Yuto Nakashima [a,*], Tomohiro I [b], Shunsuke Inenaga [a], Hideo Bannai [a], Masayuki Takeda [a]

[a] Department of Informatics, Kyushu University, Japan
[b] Department of Computer Science, TU Dortmund, Germany

ARTICLE INFO

Article history:
Received 27 May 2014
Received in revised form 23 February 2015
Accepted 1 April 2015
Available online 4 April 2015
Communicated by Tsan-sheng Hsu

Keywords:
Algorithms
Data structures
Lempel–Ziv 78 factorization
Suffix trees
Position heaps
Nearest marked ancestor queries

ABSTRACT

We present the first worst-case linear-time algorithm to compute the Lempel–Ziv 78 factorization of a given string over an integer alphabet. Our algorithm is based on nearest marked ancestor queries on the suffix tree of the given string. We also show that the same technique can be used to construct the position heap of a set of strings in worst-case linear time, when the set of strings is given as a trie.

1. Introduction

Lempel–Ziv 78 (*LZ78*, in short) is a well known compression algorithm [19]. LZ78 compresses a given text based on a dynamic dictionary which is constructed by partitioning the input string, the process of which is called LZ78 factorization. Other than its obvious use for compression, the LZ78 factorization is an important concept used in various string processing algorithms and applications [6,13].

In this paper, we show an LZ78 factorization algorithm which runs in $O(n)$ time using $O(n)$ working space for an integer alphabet, where n is the length of a given string and m is the size of the LZ78 factorization. Our algorithm does not make use of any randomization such as hashing, and works in $O(n)$ time *in the worst case*. To our knowledge, this is the first $O(n)$-time LZ78 factorization algorithm when the size of an integer alphabet is $O(n)$ and $2^{\omega(\log n \frac{\log\log\log n}{(\log\log n)^2})}$. Our algorithm computes the LZ78 trie (a trie representing the LZ78 factors) via the suffix tree [17] annotated with a semi-dynamic nearest marked ancestor data structure [18,1].

We also show that the same idea can be used to construct the *position heap* [7] of a set of strings which is given as a trie, and present an $O(\ell)$-time algorithm to construct it, where ℓ is the size of the given trie.

Our results are valid for a standard word RAM model, where each integer fits in a single machine word and can be manipulated in $O(1)$ time. Space complexities will be determined by the number of computer words (not bits).

Some of the results of this paper appeared in the preliminary versions [14,2].

Comparison to previous work

The LZ78 trie (and hence the LZ78 factorization) of a string of length n can be computed in $O(n)$ expected time

* Corresponding author.
E-mail addresses: yuto.nakashima@inf.kyushu-u.ac.jp (Y. Nakashima), tomohiro.i@cs.tu-dortmund.de (T. I), inenaga@inf.kyushu-u.ac.jp (S. Inenaga), bannai@inf.kyushu-u.ac.jp (H. Bannai), takeda@inf.kyushu-u.ac.jp (M. Takeda).

http://dx.doi.org/10.1016/j.ipl.2015.04.002
0020-0190/

and $O(m)$ space, if hashing is used for maintaining the branching nodes of the LZ78 trie [10]. In this paper, we focus on algorithms without randomization, and we are interested in the worst-case behavior of LZ78 factorization algorithms. If balanced binary search trees are used in place of hashing, then the LZ78 trie can be computed in $O(n \log \sigma)$ worst-case time and $O(m)$ working space. Our $O(n)$-time algorithm is faster than this method when $\sigma \in \omega(1)$ and $\sigma \in O(n)$. On the other hand, our algorithm uses $O(n)$ working space, which can be larger than $O(m)$ when the string is LZ78 compressible. Jansson et al. [12] proposed an algorithm which computes the LZ78 trie of a given string in $O(n(\log\log n)^2/(\log_\sigma n \log\log\log n))$ worst-case time, using $O(n(\log\sigma + \log\log_\sigma n)/\log_\sigma n)$ bits of working space. Our $O(n)$-time algorithm is faster than theirs when $\sigma \in 2^{\omega(\log n \frac{\log\log\log n}{(\log\log n)^2})}$ and $\sigma \in O(n)$, and is as space-efficient as theirs when $\sigma \in \Theta(n)$. Tamakoshi et al. [16] proposed an algorithm which computes the LZ78 trie in $O(n + (s+m)\log\sigma)$ worst-case time and $O(m)$ working space, where s is the size of the run length encoding (RLE) of a given string. Our $O(n)$-time algorithm is faster than theirs when $\sigma \in 2^{\omega(\frac{n}{s+m})}$ and $\sigma \in O(n)$.

The position heap of a single string of length n over an alphabet of size σ can be computed in $O(n \log \sigma)$ worst-case time and $O(n)$ space [7], if the branches in the position heap are maintained by balanced binary search trees. Independently of this present work, Gagie et al. [11] showed that the position heap of a given string of length n over an integer alphabet can be computed in $O(n)$ time and $O(n)$ space, via the suffix tree of the string.

2. Preliminaries

2.1. Notations on strings

We consider a string w of length n over integer alphabet $\Sigma = \{1, \ldots, \sigma\}$, where $\sigma \in O(n)$. The length of w is denoted by $|w|$, namely, $|w| = n$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. For a string $w = xyz, x, y$ and z are called a *prefix*, *substring*, and *suffix* of w, respectively. The set of suffixes of a string w is denoted by $Suffix(w)$. The i-th character of a string w is denoted by $w[i]$ for $1 \le i \le n$, and the substring of a string w that begins at position i and ends at position j is denoted by $w[i..j]$ for $1 \le i \le j \le n$. For convenience, let $w[i..j] = \varepsilon$ if $j < i$. For any string w, let w^R denote the reversed string of w, i.e., $w^R = w[n]w[n-1]\cdots w[1]$.

2.2. Suffix trees

We give the definition of a very important and well known string index structure, the suffix tree. To assure property 4 below for the sake of presentation, we assume that string w ends with a unique character that does not occur elsewhere in w.

Definition 1 *(Suffix trees).* (See [17].) For any string w, its suffix tree, denoted $STree(w)$, is a labeled rooted tree which satisfies the following:

1. each edge is labeled with a non-empty substring of w;
2. each internal node has at least two children;
3. the labels x and y of any two distinct out-going edges from the same node begin with different symbols in Σ;
4. there is a one-to-one correspondence between the suffixes of w and the leaves of $STree(w)$, i.e., every suffix is spelled out by a unique path from the root to a leaf.

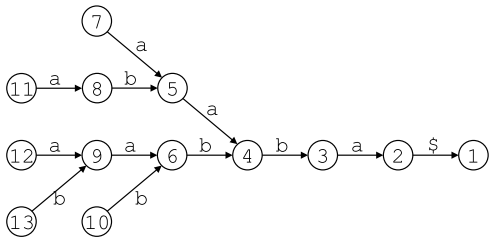

Fig. 1. $CST(W)$ for $W = \{\texttt{aaba\$}, \texttt{bbba\$}, \texttt{ababa\$}, \texttt{aabba\$}, \texttt{babba\$}\}$. Each node u is associated with $id(u)$.

Since any substring of w is a prefix of some suffix of w, all substrings of w can be represented as a path from the root in $STree(w)$. For any node v, let $str(v)$ denote the string which is a concatenation of the edge labels from the root to v. A locus of a substring x of w in $STree(w)$ is a pair (v, γ) of a node v and a (possibly empty) string γ, such that $str(v)\gamma = x$ and γ is the shortest. A locus is said to be an explicit node if $\gamma = \varepsilon$, and is said to be an implicit node otherwise. It is well known that $STree(w)$ can be represented with $O(n)$ space, by representing each edge label x with a pair (i, j) of positions satisfying $x = w[i..j]$.

Theorem 1. *(See [8].) Given a string w of length n over an integer alphabet, $STree(w)$ can be computed in $O(n)$ time.*

2.3. Suffix trees of multiple strings

A *generalized* suffix tree of a set of strings is the suffix tree that contains all suffixes of all the strings in the set. Generalized suffix trees for a set $W = \{w_1\$, \ldots, w_k\$\}$ of strings over an integer alphabet can be constructed in linear time in the total length of the strings. We assume that \$ is a unique character that does not occur in w_i $(1 \le i \le k)$, and for any $1 \le i, j \le k$ we assume that w_i is not a suffix of w_j.

The set W of strings can be represented as a *reversed trie* called a common-suffix trie, which is defined as follows.

Definition 2 *(Common-suffix tries).* (See [5].) The common-suffix trie of a set W of strings, denoted $CST(W)$, is a reversed trie such that

1. each edge is labeled with a character in Σ;
2. any two in-coming edges of any node are labeled with distinct characters;
3. each node v represents the string that is a concatenation of the edge labels in the path from v to the root;
4. for each string $w \in W$ there exists a unique leaf which represents w.

An example of $CST(W)$ is illustrated in Fig. 1.

Let ℓ be the number of nodes in $CST(W)$, and let $Suffix(W)$ be the set of suffixes of the strings in W, i.e.,