Fast and Space Efficient Linear Suffix Array Construction

Sen Zhang
Dept. of Math., Comp. Sci. and Stat.
SUNY College at Oneonta
NY 07104, U.S.A.
zhangs@oneonta.edu

Ge Nong
Computer Sci. Dept.
Sun Yat-Sen University
Guangzhou 510275, P.R.C.
issng@mail.sysu.edu.cn

Let S be an n-character string terminated with an unique smallest sentinel, its suffix array $SA(S)$ is an array of pointers for all the suffixes in S sorted in the lexicographically ascending order. Specially, the Burrows-Wheeler transform for building efficient compression solutions can be quickly computed by fast suffix sorting based on suffix array construction algorithms (SACAs). The existing well-known practical linear SACAs are those two contemporarily reported in 2003 by Kärkkäinen and Sanders (KS) [1], and Ko and Aluru (KA) [2]. We recently proposed a novel fast and space efficient linear SACA, whose core is the concept of Critical Substring introduced by us as following: $S[i..i+d+1]$ is said to be the d-critical substring for the d-critical character $S[i]$ in S; for $i \geq n-d$, $S[i..i+d+1] = S[i..n-2]\{S[n-1]\}^{n-i+2}$, where $\{S[n-1]\}^{n-i+2}$ denotes that $S[n-1]$ is repeated $n-i+2$ times. In addition, we have the following definitions. (I) a character $S[i]$ is said to be d-critical, where $d \geq 2$, iif (1) $S[i]$ is a LMS character; or else (2) $S[i-d]$ is a d-critical character, $S[i+1]$ is not a LMS character and no character in $S[i-d+1..i-1]$ is d-critical. (II) a suffix $suf(S,i)$ is said to be type-S or type-L if $suf(S,i) < suf(S,i+1)$ or $suf(S,i) > suf(S,i+1)$, respectively; the last suffix $suf(S,n-1)$ consisting of only the sentinel is defined as type-S. (III) a character $S[i]$ is said to be type-S or type-L if $suf(S,i)$ is type-S or type-L, respectively. (IV) Leftmost type-S (LMS) character: $S[i]$ is said to be a LMS character if $S[i]$ is type-S and $S[i-1]$ is type-L, where $i \in (0, n-1]$. (V) Leftmost type-S (LMS) suffix: $suf(S,i)$ is said to be a LMS suffix if $S[i]$ is a LMS character. By sampling the fixed-size *d-critical* substrings to divide-and-conquer the problem, our new algorithm is very simple, for which a fully-functioning sample implementation is embodied in only about 100 lines of C code. The experimental results on the Canterbury and Manzini-Ferragina corpora show that our algorithm outperforms both the KS and KA algorithms: compared with the KS, ours can be more than twice faster and use more than 50% fewer space; compared with the KA, ours can be 9% faster and use 40% fewer space. To approach the lightweight space extreme, we further improve our linear algorithm to use an extra working space of only $0.25n + O(1)$ bytes to construct the suffix array for any size-n string of a constant or integer alphabet, where the characters of an integer alphabet are in $[0..n-1]$. Besides using less space, our lightweight linear algorithm still runs more than 1.5 times faster than the KS algorithm in the experiments.

References

[1] J. Kärkkäinen and P. Sanders, “Simple linear work suffix array construction,” in *30th International Colloquium on Automata, Languages and Programming (ICALP ’03)*, 2003, pp. 943–955.

[2] P. Ko and S. Aluru, “Space efficient linear time construction of suffix arrays,” in *Proceedings 14th Annual Symp. Combinatorial Pattern Matching, LNCS 2676, Springer-Verlag*, 2003, pp. 200–210.

1068-0314/08 $25.00 © 2008 IEEE
DOI 10.1109/DCC.2008.61