International Workshop On Combinatorial Algorithms
IWOCA

http://www.iwoca.org—Problems section.

Computing The Palindromic Length Faster

Mikhail Rubinchik
Ural Federal University, Russia
mikhail.rubinchik@gmail.com

Arseny M. Shur
Ural Federal University, Russia
arseny.shur@urfu.ru

presented at IWOCA 2015
06-10-2015

A palindrome is a string equal to its reversal. Since letters are palindromes, each string can be obtained by concatenating palindromes. The minimum number of factors in such concatenation is the *palindromic length* of a string, denoted by $\mathsf{PL}(S)$. Thus, if $S = 01001001$, then $\mathsf{PL}(S) = 2$ because $S = 0 \cdot 1001001$ and S is not a palindrome.

Dynamic programming is a natural approach to the computation of the palindromic length: for a string S of length n define an array $\mathsf{PL}[0..n]$ such that $\mathsf{PL}[i]$ is the palindromic length of the string $S[1..i]$ (in particular, $\mathsf{PL}[n] = \mathsf{PL}(S)$). Now

$$\mathsf{PL}[0] = 0, \quad \mathsf{PL}[i] = 1 + \max\{\mathsf{PL}[j] \mid S[j{+}1..i] \text{ is a palindrome}\}.$$

In a naive way, the implementation of such an algorithm requires $\Omega(n^2)$ time and space but this bound can be lowered to $O(n \log n)$ by processing "series" of palindromic suffixes of the current string rather than individual such suffixes [1]. (Essentially the same algorithm was obtained independently in [2].) There are $O(\log n)$ series of palindromic suffixes in each position of the word and every series can be processed in constant time. On the other hand, [1] contains an example of a string having $\Omega(n \log n)$ series of palindromes in total. This example speaks in favor of an idea that $\Theta(n \log n)$ is the best possible time for finding the palindromic length.

In our IWOCA 2015 paper [3][1] we exhibit a new data structure called *eertree*, which allows one to store all data about palindromes in a string in a compact form. This data structure can be built in linear time (offline; the online bound is only slightly worse). Up to now, we were able to get a new $O(n \log n)$ algorithm for the palindromic length using eertree. However, the potential of this data structure seems to be big, so we ask the following question:

Is it possible to avoid explicit processing of each single series of palindromic suffixes and improve the $O(n \log n)$ time bound for the palindromic length problem?

[1] Probably this citation should be updated later by the official publication.

References

[1] G. Fici, T. Gagie, J. Krkkinen, and D. Kempa: A subquadratic algorithm for minimum palindromic factorization. J. Discrete Algorithms 28: 41-48 (2014).

[2] T. I, S. Sugimoto, S. Inenaga, H. Bannai, and M. Takeda: Computing palindromic factorizations and palindromic covers on-line. Proc. CPM 2014, LNCS 8486, Springer, 2014, pp.150161.

[3] M. Rubinchik and A.M. Shur: EERTREE: An Efficient Data Structure for Processing Palindromes in Strings. IWOCA 2015. Available at: arXiv:1506.04862, 2015.