Abstract

Suffix tree (and the closely related suffix array) are fundamental structures capturing all substrings of a given text essentially by storing all its suffixes in the lexicographical order. In some applications, such as sparse text indexing, we work with a subset of b interesting suffixes, which are stored in the so-called sparse suffix tree. Because the size of this structure is $\Theta(b)$, it is natural to seek a construction algorithm using only $O(b)$ words of space assuming read-only random access to the text. We design a linear-time Monte Carlo algorithm for this problem, hence resolving an open question explicitly stated by Bille et al. [TALG 2016]. The best previously known algorithm by I et al. [STACS 2014] works in $O(n \log b)$ time. As opposed to previous solutions, which were based on the divide-and-conquer paradigm, our solution proceeds in n/b rounds. In the r-th round, we consider all suffixes starting at positions congruent to r modulo n/b. By maintaining rolling hashes, we can lexicographically sort all interesting suffixes starting at such positions, and then we can merge them with the already considered suffixes. For efficient merging, we also need to answer LCE queries efficiently (and in small space). By plugging in the structure of Bille et al. [CPM 2015] we obtain $O(n + b \log b)$ time complexity. We improve this structure by a recursive application of the so-called difference covers, which then implies a linear-time sparse suffix tree construction algorithm. We complement our Monte Carlo algorithm with a deterministic verification procedure. The verification takes $O(n\sqrt{\log b})$ time, which improves upon the bound of $O(n \log b)$ obtained by I et al. [STACS 2014]. This is obtained by first observing that the pruning done inside the previous solution has a rather clean description using the notion of graph spanners with small multiplicative stretch. Then, we are able to decrease the verification time by applying difference covers twice. Combined with the Monte Carlo algorithm, this gives us an $O(n\sqrt{\log b})$-time and $O(b)$-space Las Vegas algorithm.