Maintaining the cycle structure of permutations efficiently

Zsuzsanna Lipták and Francesco Masillo

University of Verona

Permutations have a central role in many applications in mathematics and computer science, spanning from combinatorics and group theory to algorithms for random generation and genome rearrangements. It is well known that a permutation can be decomposed into disjoint cycles [1], and there exists a simple linear-time algorithm for computing the cycle decomposition of a given permutation.

In [2], a superset of the current authors studied a combinatorial problem on the Burrows-Wheeler Transform (BWT). For its efficient solution, we introduced a new data structure, the Forest of Splay Trees (FST), to store the standard permutation of a word. At its core, the FST consists of one splay tree per cycle and a counter for the number of cycles. With some additional tweaks, we exploited known properties of splay trees to answer cycle membership queries in amortized $\mathcal{O}(\log n)$ time, as well as to update the permutation using a specific type of transposition, also in amortized $\mathcal{O}(\log n)$ time, for permutations of n. Finally, it was key for the algorithm's running time that the data structure allow returning the number of cycles after each update in constant time.

In this work, we extend the range of operations of the FST. We generalize the specific transposition operation used in [2] to transpositions (i, j) and $(\pi(i), \pi(j))$, for two arbitrary elements i and j, maintaining the previous amortized $\mathcal{O}(\log n)$ time. Moreover, we describe how to use the FST for other common tasks on permutations, such as returning $\pi(i)$, $\pi^{-1}(i)$, $\pi^k(i)$, and $\pi^{-k}(i)$. We compare the extended set of operations to other data structures commonly used for permutations. In particular, we compare against the one-line notation (stored as an array), the one-line notation together with the inverse permutation (stored as two arrays), and the wavelet tree [3].

As far as space is concerned, the FST uses $3n \log n$ bits, while the classical data structures require $n \log n$, $2n \log n$, resp. $n \log n + o(n)$ bits. The FST can be slower for the different operations (lookup and update), but by at most a logarithmic factor. On the other hand, the FST is the only data structure that can efficiently update the number of cycles, as well as answering cycle membership queries. Thus the FST is the data structure of choice in applications where one needs to dynamically keep track of the cycle structure of a permutation.

References

[1] M. Bóna. *Combinatorics of Permutations, Second Edition*. Discrete mathematics and its applications. CRC Press, 2012.

[2] S. Giuliani, Zs. Lipták, F. Masillo, and R. Rizzi. When a dollar makes a BWT. *Theor. Comput. Sci.*, 857:123–146, 2021.

[3] G. Navarro. Wavelet trees for all. *J. Discrete Algorithms*, 25:2–20, 2014.