Experimental bioinformatics

The Burrows-Wheeler transform and the FM-index

TP

M1-ENS

The aim of this tutorial is to develop a toy implementation of the FM-index, as a Python class. You are expected to put some efforts toward the quality of the implementation (public/private methods, docstrings, comments, tests, user interface, etc.).

(1) The Burrows-Wheeler transform

(1.1) Construction

Initialize the fields `self.sa` and `self.bwt` of the `FMindex` class. You will rely on the linear-time constuction of the suffix array provided in `ks.py` (Karkkainen-Sanders algorithm) as `simple_kark_sort`.

(1.2) Naive inversion

Implement a method `self.get_string__naive` that retrieves the original string by reconstructing the Burrows-Wheeler matrix.

(1.3) Compression capabilities

Implement compression and decompression functions for run-length and move-to-front encodings. Elaborate on the compression capabilities of these frameworks (separately and combined) on different types of text (random string, litterate text, genome).

(2) The FM-index

(2.1) Construction

Initialize the fields `self.fm_count`, `self.fm_rank`, `self.fm_ranks` and `self.next_smallest_letter` of the FMindex class. Add a logging option to the class constructor for reporting the initialization steps.

(2.2) Fast BWT inversion

Implement a public method `self.get_string` that takes advantage of the LF-mapping property of the Burrows-Wheeler transform to reconstruct the original string.

(2.3) Exact pattern matching

Implement three public methods `self.membership`, `self.count` and `self.locate` that allow one to perform pattern matching queries. Try to factorize implementations as much as possible.

(2.4) Lightweight FM-index

Implement a subsampling of the `self.ranks` arrays. Adapt the previous methods to compute ranks on the fly when needed. Estimate the space-time tradeoffs depending on the proportion of `self.ranks` arrays that are indeed stored.

(2.5) Approximate pattern matching

Adapt the pattern matching functions so that they tolerate a small amount `e` of `self.ranks` errors in the pattern. Evaluate the resulting complexity.