Bioinformatics

The Burrows-Wheeler transform and the FM-index

TD/TP

As you progress within the tutorial, you should develop your intuition on the data structure, as well as gaining automy for good practices in algorithmics (proper definition of problems, running time analysis, high-level description of algorithms) and implementations (extensive code documentation with docstrings and comments, robust tests).

The aim of this tutorial is to develop a toy implementation of the FM-index, as a Python class. We start with compression aspects of the Burrows-Wheeler transform, relying on naive algorithms for computing and inverting the BWT. Then, we build a reasonably efficient textbook version of the FM-index, and later shrink its space requirements by subsampling arrays' components of the index.

(1) ■ Compressing with the Burrows-Wheeler transform

(1.1) ◆ A simple compression scheme, ...

▶ **Question 1.** *(✎) Encode the string "AAAABBBCCDAA" using run-length encoding.*

▶ **Question 2.** *(✎) Retrieve the string that is encoded as `X3Y12WZ5X` through run-length encoding.*

▶ **Question 3.** *(💻) Implement the algorithms you used as two `Python` functions: `rle` and `irle`.*

(1.2) ◆ and a reversible transformation...

▶ **Question 4.** *(✎) Compute the Burrows-Wheeler transform of "ANANAS", deriving it from the corresponding Burrows-Wheeler matrix.*

▶ **Question 5.** *(✎) Do you think the practical complexity of the algorithm you used readily derives from its asymptotical complexity ? Justify.*

▶ **Question 6.** *(💻) Implement the algorithm you used as a `Python` function: `bwt`.*

▶ **Question 7.** *(✎) Retrieve the string S whose Burrows-Wheeler transform is "YIVVI$F", by iteratively reconstructing the Burrows-Wheeler matrix.*

▶ **Question 8.** *(💻) Implement the algorithm you used as a `Python` function: `ibwt`.*

(1.3) ◆ ...that makes runs out of redundancy

▶ **Question 9.** *(🔍) Compare the compression capabilities of the BWT-RLE scheme on the files `dickens.txt`, `dudh.txt`, `random.txt`. In particular, highlight the crucial role of the Burrows-Wheeler transform.*

(2) ■ Indexing with the BWT: the FM-index

Now, open `fmindex_template.py`. We start by building the `TextbookFMindex` class.

(2.1) ◆ Efficient construction of the BWT

▶ **Question 10.** *(📖) Recall the definition of the suffix array, and the definition of the BWT that relies on the latter.*

▶ **Question 11.** *(✎) Use these definitions to compute the Burrows-Wheeler transform of "ANANAS".*

▶ **Question 12.** *(💻) Fill in the methods `_compute_sa` and `_compute_bwt` accordingly. For the former, use the function `simple_kark_sort` from the file `ks.py`: it returns the suffix array of S when given S\$ as input.*

(2.2) ◆ Efficient inversion of the BWT

► **Question 13.** *(✎) Retrieve the string S whose Burrows-Wheeler transform is "YIVVI$F", by leveraging the LF-mapping property.*

► **Question 14.** *(✎) What are the* Count *map and* $(\mathsf{Rank}_x)_{x\in\{A,N,S\}}$ *arrays of the FM-index of "ANANAS".*

► **Question 15.** *(✎) Propose an algorithm that compute the* $(\mathsf{Rank}_x)_{x\in\Sigma}$ *arrays of the FM-index of a string S in time $O(|S|\cdot|\Sigma|)$.*

► **Question 16.** *(✎) Propose an algorithm that derives the* Count *map from the* $(\mathsf{Rank}_x)_{x\in\Sigma}$ *arrays of an FM-index in time $O(|\Sigma|)$.*

► **Question 17.** *(💻) Fill in the methods* `_compute_ranks` *and* `_compute_countmap` *accordingly.*

► **Question 18.** *(✎) Express the LF-mapping as a function of the* Count *map and the* $(\mathsf{Rank}_x)_{x\in\Sigma}$ *arrays.*

► **Question 19.** *(💻) Fill in the method* `_lf_mapping` *accordingly.*

► **Question 20.** *(💻) Fill in the method* `get_string`*, that retrieve the string encode within the BWT in linear time.*

(2.3) ◆ Efficient answers to pattern matching queries

► **Question 21.** *(✎) Represent graphically the successive F/L-intervals considered while looking for the pattern "NAN" using the FM-index.*

► **Question 22.** *(💻) Fill in the method* `_get_pattern_interval`*.*

► **Question 23.** *(💻) Fill in the methods* `membership`*,* `count` *and* `locate`*, to answer pattern matching queries.*

(2.4) ◆ Subsampling arrays: trading time for space

We now shift to the real `FMindex` class. It inherits from the `TextbookFMindex` class, but will only keep one out of σ entries of the suffix array, and one out of ρ entries of the ranks arrays. Naturally, dedicated methods will be needed to access these subsampled arrays.

► **Question 24.** *(💻) Fill in the method* `_subsample_ranks`*. The sparsity of the subsampled arrays should depend on a new attribute of the* `FMindex` *class you will introduce.*

► **Question 25.** *(💻) Fill in the method* `_get_ranks_entry`*, that should act as* `TextbookFMindex`*'s* `ranks` *arrays, recomputing non-stored values on-the-fly.*

► **Question 26.** *(💻) Fill in the method* `_subsample_sa`*. The sparsity of the subsampled suffix array should depend on a new attribute of the* `FMindex` *class you will introduce.*

► **Question 27.** *(💻) Fill in the method* `_get_sa_entry`*, that should act as* `TextbookFMindex`*'s* `sa` *array, recomputing non-stored values on-the-fly.*

► **Question 28.** *(💻) Override all methods inherited from* `TextbookFMindex` *that makes call to the ranks and suffix arrays, so that they work with subsampled arrays.*

► **Question 29.** *(🔍 **BONUS**) Elaborate on the space-time tradeoff offered by the subsampling rates ρ and σ.*