Optimal-time text indexing with FM indexes

Nicola Prezza

References and Reading

[1] Gagie, T., Navarro, G., Prezza, N. *Optimal-time text indexing in BWT-runs bounded space*. In Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, NA, USA, January 7-10. Pre-print of full version: https://arxiv.org/abs/1705.10382

Intro and Material

The goal of these exercises is to go through the main results presented in the paper. Be aware that some of the solutions to the exercises are different than those described in the paper (they are actually simpler: we discovered these simpler versions after publishing the paper), so you won't find everything in the paper (and, if you solve the exercises yourself, you don't actually need to read the paper).

If you still want to read the paper to get some ideas, in the exercises we go through the material of Lemma 2, Lemma 3, Lemma 6, and Theorem 2.

Notation

Let r be the number of bits set in a bitvector of length n. To simplify notation, with $t_p = \log(n/r)$ we will indicate the time needed to perform rank/predecessor queries on the Elias-Fano representation of the bitvector. Plugging this structure and wavelet trees in our (static) run-length string representation (see previous exercises), we obtain a structure of $O(r)$ words of space supporting rank, select, access queries in $t_q = t_p + \log\sigma = \log(n/r) + \log\sigma$ time (where r now is the number of equal-letter runs in the string and σ is the alphabet size). If we use this string to represent the BWT, we obtain a basic run-length FM index supporting FL/LF function computation and count queries.

Exercises

Let r be the number of equal-letter runs in the BWT of a text T of length n. Let m be the length of a pattern P. As seen in the previous lectures and exercises, we can build a run-length FM-index taking $O(r + n/s)$ words of space and supporting these queries:

- Count the number of occurrences of P in T in $O(m \cdot t_q)$ time
- Locate the occ occurrences of P in T in $O((m + s \cdot occ) \cdot t_q)$ time

Here, s is a *sample rate*: we sample the suffix array entries corresponding to one out of s text positions. To extract a suffix array entry $SA[i]$, we apply the LF mapping starting from position i at most s times. The goal of the following exercise is to remove the dependency from s in space and query times: we will describe a suffix array sampling of size r that supports locating each pattern occurrence in t_p (predecessor) time.

1 Locate in predecessor time Consider the following suffix array sampling: we sample the first position of every equal-letter run in the BWT[1] [2]. Example: $T = abracadabra\$$, $BWT(T) = \underline{ard}\underline{\$}\underline{rca}\underline{aaa}\underline{b}\underline{b}$ (sampled positions are underlined). We store the text position corresponding to the sampled BWT positions: $11, 10, 7, 12, 3, 5, 8, 1, 9$.

[1] to be precise, in a suffix array sampling we sample the F column; in our case, we sample instead the L column.

[2] This is the first difference with the paper: in Lemma 2 of the paper, we store samples at the beginning and end of every run (not just at the beginning).

1.1 Suppose that, using our run-length FM-index, we found the suffix array range $[l, r]$ of a pattern P of length m (i.e. suffix array positions $SA[l, ..., r]$ contain all text occurrences of P). Show how to compute, in $O(m \cdot t_q)$ time, suffix array entry $SA[l]$ [3]. Hint: note that the longest common prefix between the suffixes starting in $SA[l]$ and $SA[l-1]$ must be shorter than m (otherwise $SA[l-1]$ would be an occurrence of P). Think what happens when applying the FL mapping repeatedly starting from positions l and $l-1$.

1.2 Our goal now is to compute efficiently $SA[i+1]$ given $SA[i]$. Since (from exercise 1.1) we know $SA[l]$, this will allow us to compute the whole range $SA[l, ..., r]$ efficiently.

Suppose that now we are marking the *last* position of each BWT equal-letter run. Since we allow $O(r)$ space, we can associate to each marked position a constant number of memory words of information. Suppose we know $SA[i]$. We want to compute $SA[i+1]$. Look inside $BWT[i, i+1]$: if the two characters are different, then $BWT[i]$ is marked. In this case, the information associated with $BWT[i]$ could simply be the text position x corresponding to $BWT[i+1]$: then, $SA[i+1] = x+1$ and we are done.

On the other hand, if $BWT[i, i+1] = cc$, for some $c \in \Sigma$, these BWT positions are not marked and therefore the above strategy doesn't seem to help. However, note that we could apply the LF mapping to BWT positions i and $i+1$, and repeat recursively the strategy (since $BWT[i, i+1] = cc$, then $LF(i+1) = LF(i)+1$).

Clearly, if we directly implement the above strategy using the LF mapping, in the worst case we might have to compute LF many times before finding a marked position. Can we avoid using the LF mapping at all and "skip" directly to the marked position? Hint: we are "virtually" computing $i, LF(i), LF(LF(i)) = LF^2(i), LF^3(i), ...$ What are the text positions associated to those BWT positions? Remember that we know text position $SA[i]$, and that we are allowed storing $O(r)$ extra words of information.

2 Trade-offs

2.1 Describe a compressed FM index taking $O(r)$ words of space and able to locate the occ occurrences of a pattern of length m in $O(m \cdot t_q + occ \cdot t_p)$ time.

2.2 Describe a compressed FM index taking more space and able to locate the occ occurrences of a pattern of length m in optimal $O(m + occ)$ time, under the assumption that σ is constant.

3 Extract text Now we know how to support efficient count and locate queries, but we still miss a solution for efficiently extracting text. Solve the following exercises:

3.1 Suppose we are marking the first position of every equal-letter BWT run. Now, mark the positions on the text corresponding to those marked BWT positions. Prove the following Lemma: every text substring $T[i..j]$ has at least one occurrence $T[i'..j'] = T[i..j]$ such that $T[i'..j']$ contains a marked position (hint: we essentially already proved this in exercise 1.1).

3.2 Let j be a marked text position, and let $e \leq \log n$. Consider the substring $S = T[j-2^e..j-1]$ (for simplicity, assume that $j - 2^e \geq 1$) preceding position j. The property of exercise 3.1 tells us that the two halves of S, each of length 2^{e-1}, have at least one occurrence crossing some other marked positions. The reasoning can be applied recursively to these (shorter) substrings, yielding substrings of length 2^{e-2} surrounding some marked position. Exploit this observation and propose a structure of size $O(r \log n)$ words that permits to extract any $T[i]$ in $O(\log n)$ time.

[3] This is the second difference with the paper: in Lemma 2, we only guarantee to find a $SA[i]$, for some $l \leq i \leq r$. In this exercise, we are guaranteed to find $SA[l]$.