MONI Can Find k-MEMs

Igor Tatarnikov
Dalhousie University, Halifax, Canada

Ardavan Shahrabi Farahani
Dalhousie University, Halifax, Canada

Sana Kashgouli
Dalhousie University, Halifax, Canada

Travis Gagie
Dalhousie University, Halifax, Canada

Abstract

Suppose we are asked to index a text $T[0..n-1]$ such that, given a pattern $P[0..m-1]$, we can quickly report the maximal substrings of P that each occur in T at least k times. We first show how we can add $O(r \log n)$ bits to Rossi et al.'s recent MONI index, where r is the number of runs in the Burrows-Wheeler Transform of T, such that it supports such queries in $O(km \log n)$ time. We then show how, if we are given k at construction time, we can reduce the query time to $O(m \log n)$.

2012 ACM Subject Classification Theory of computation → Pattern matching

Keywords and phrases Compact data structures, Burrows-Wheeler Transform, run-length compression, maximal exact matches

Digital Object Identifier 10.4230/LIPIcs.CPM.2023.26

Funding This research was supported by National Institutes of Health (NIH) NIAID (grant no. HG011392), the National Science Foundation NSF IIBR (grant no. 2029552), a Natural Science and Engineering Research Council (NSERC) Discovery Grant (grant no. RGPIN-07185-2020) and the Comisión Nacional de Investigación Científica y Tecnológica (CONICYT) through the Centre for Biotechnology and Bioengineering (grant CeBiB FB0001).

Acknowledgements The authors thank Christina Boucher, Ben Langmead, Manuel Mattheisen and Massimiliano Rossi for helpful discussions.

1 Introduction

In his foundational text *Compact Data Structures: A Practical Approach* [9, Section 11.6.1], Navarro posed the following problem:

> "Assume we have the suffix tree of a collection of genomes $T[0..n-1]$. We then receive a short DNA sequence $P[0..m-1]$ and want to output all the maximal substrings of P that appear at least k times in T... Those substrings of P are likely to have biological significance."[1]

He described how to solve the problem with a suffix tree for T in $O(m \,\mathrm{polylog}(n))$ time. Since T is a collection of genomes, it is likely to be highly repetitive and the theoretically best suffix-tree implementation is likely to be the $O(r \log(n/r))$-space one by Gagie, Navarro and Prezza [5], where r is the number of runs in the Burrows-Wheeler Transform (BWT) of T. That full data structure is complicated, however, and has never been implemented.

[1] We have changed T to $T[0..n-1]$ and $P[1..m]$ to $P[0..m-1]$ for consistency with the rest of this paper, and omitted a parameter bounding from below the length of the substrings (since we can filter them afterwards).

Very recently, Navarro [10] also gave solutions not based on a suffix tree, with the following bounds:

- $O(g_{rl})$ space and $O(km^2 \log^\epsilon n)$ query time;
- $O(\delta \log(n/\delta))$ space and $O(m \log m(\log m + k \log^\epsilon n))$ query time;
- $O(g)$ space and $O(m^2 \log^{2+\epsilon} n)$ query time when $k = \omega(\log^2 n)$;
- $O(\gamma \log(n/\gamma))$ space and $O(m \log m \log^{2+\epsilon} n)$ query time when $k = \omega(\log^2 n)$.

We refer readers to Navarro's paper and the references therein for definitions of g_{rl}, δ, g and γ.

In this paper we first show how we can add $O(r \log n)$ bits to Rossi et al.'s [13] recent MONI index to obtain a solution with $O(km \log n)$ query time. We then show how, if we are given k at construction time, we can reduce the query time to $O(m \log n)$, simultaneously using less space than Gagie et al.'s compressed suffix tree and less time than Navarro's solutions. The rest of the paper is laid out as follows: in Section 2 we review MONI in enough depth to build on it; in Section 3 we show how we can extend ϕ queries to support sequential access to the LCP array; in Section 4 we show how to use ϕ and ϕ^{-1} queries and LCP access to obtain a solution with $O(km \log n)$ query time; in Section 5 we show how, if we are given k at construction time, we can precompute some answers, reducing the query time to $O(m \log n)$; and we conclude in Section 6. For the sake of brevity we assume readers are familiar with the concepts in Navarro's text.

2 MONI

Bannai, Gagie and I [1] designed an index for T that takes $O(r \log n)$ bits plus the space needed to support fast random access to T, and lists all the maximal exact matches (MEMs) of P with respect to T – that is, all the substrings $P[i..j]$ of P occurring in T such that $i = 0$ or $P[i-1..j]$ does not occur in T, or $j = m-1$ or $P[i..j+1]$ does not occur in T – in $O(m \log \log n)$ time plus the time needed for $O(m)$ random accesses to T. MEMs are widely used in DNA alignment [8] and they are the substrings of P Navarro asks for when $k = 1$. Generalizing to arbitrary k, we refer to the substrings he asks for as k-MEMs.

Bannai et al. did not give an efficient construction algorithm or an implementation, but Rossi et al. later did. They called their implementation MONI, the Finnish word for "multi", since it is intended to store a multi-genome reference. Boucher et al. [2] then gave a version of MONI that processes P online using longest common extension (LCE) queries on T instead of random access. We can support those LCE queries in $O(\log n)$ time with a balanced straight-line program for T, which in practice takes significantly less space than the rest of MONI.

We now sketch how Boucher et al.'s version of MONI works, incorporating ideas from Nishimoto and Tabei [12] and Brown, Gagie and Rossi [4] about replacing rank queries by table lookup and assuming we have an LCE data structure. Suppose

$$T = \texttt{GATTACAT\#AGATACAT\#GATACAT\#GATTAGAT\#GATTAGATA\$}$$

with $\$ \prec \# \prec \texttt{A} \prec \cdots \prec \texttt{T}$, and consider Table 1, in which the permutation FL is just the inverse of the more familiar permutation LF.

For each of the value j between 0 and $r - 1 = 13$, we conceptually extract from this table the starting and ending positions $\mathrm{head}(j)$ and $\mathrm{tail}(j)$ of run j in the BWT, $\mathrm{SA}[\mathrm{head}(j)]$, $\mathrm{SA}[\mathrm{tail}(j)]$, $\mathrm{BWT}[\mathrm{head}(j)]$, $\mathrm{LF}[\mathrm{head}(j)]$ and the rank of the predecessor of $\mathrm{LF}[\mathrm{head}(j)]$ in the set

$$\{\mathrm{head}[0], \ldots, \mathrm{head}[r-1]\}\,.$$

Table 1 The full table from which we conceptually start when building MONI for our example text $T =$ `GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$`.

i	SA[i]	LCP[i]	lexicographically ith cyclic shift of T	BWT[i]	LF(i)	FL(i)
0	44	0	`$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA`	A	5	29
1	8	0	`#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT`	T	32	11
2	17	1	`#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT`	T	33	28
3	25	4	`#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT`	T	34	30
4	34	9	`#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT`	T	35	31
5	43	0	`A$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGAT`	T	36	0
6	4	1	`ACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATT`	T	37	22
7	13	5	`ACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGAT`	T	38	23
8	21	8	`ACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GAT`	T	39	24
9	30	1	`AGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATT`	T	40	25
10	39	4	`AGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATT`	T	41	26
11	9	5	`AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#`	#	1	27
12	6	1	`AT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTAC`	C	22	32
13	15	3	`AT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATAC`	C	23	33
14	23	6	`AT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATAC`	C	24	34
15	32	11	`AT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAG`	G	25	35
16	41	2	`ATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAG`	G	26	36
17	11	3	`ATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AG`	G	27	38
18	19	10	`ATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#G`	G	28	39
19	1	2	`ATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$G`	G	29	42
20	27	4	`ATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#G`	G	30	43
21	36	7	`ATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#G`	G	31	44
22	5	0	`CAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTA`	A	6	12
23	14	4	`CAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATA`	A	7	13
24	22	7	`CAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATA`	A	8	14
25	31	0	`GAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTA`	A	9	15
26	40	3	`GATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTA`	A	10	16
27	10	4	`GATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#A`	A	11	17
28	18	11	`GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#`	#	2	18
29	0	3	`GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$`	$	0	19
30	26	5	`GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#`	#	3	20
31	35	8	`GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#`	#	4	21
32	7	0	`T#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACA`	A	12	1
33	16	2	`T#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACA`	A	13	2
34	24	5	`T#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACA`	A	14	3
35	33	10	`T#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGA`	A	15	4
36	42	1	`TA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGA`	A	16	5
37	3	2	`TACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GAT`	T	42	6
38	12	6	`TACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGA`	A	17	7
39	20	9	`TACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GA`	A	18	8
40	29	2	`TAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GAT`	T	43	9
41	38	5	`TAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GAT`	T	44	10
42	2	1	`TTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GA`	A	19	37
43	28	3	`TTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GA`	A	20	40
44	37	6	`TTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GA`	A	21	41

Table 2 The values we extract from Table 1, with the last two columns sorted.

j	head(j)	SA[head(j)]	tail(j)	SA[tail(j)]	BWT[head(j)]	$\mu(j)$	finger(j)
0	0	44	0	44	`A`	0	0
1	1	8	10	39	`T`	1	1
2	11	9	11	9	`#`	2	1
3	12	6	14	23	`C`	3	1
4	15	32	21	36	`G`	5	1
5	22	5	27	10	`A`	6	1
6	28	18	28	18	`#`	12	3
7	29	0	29	0	`$`	17	4
8	30	26	31	35	`#`	19	4
9	32	7	36	42	`A`	22	5
10	37	3	37	3	`T`	25	5
11	38	12	39	20	`A`	32	9
12	40	29	41	38	`T`	42	13
13	42	2	44	37	`A`	43	13

In practice we can compute the values directly without building Table 1, using prefix-free parsing [3].

We build Table 2 with these values but we sort the last two columns, which we refer to as $\mu(j)$ and finger(j). An equivalent way to define $\mu(j)$ and finger(j), illustrated in Table 1, is to draw boxes corresponding to the runs in the BWT, permute those boxes according to LF, and write their starting positions in order as the $\mu(j)$ values and the numbers of the runs in the BWT covering their starting positions as the finger(j) values. Storing Table 2 takes about

$$2r\lg(n/r) + 2r\lg n + r\lg\sigma + 2r$$

bits, where σ is the size of the alphabet. (We do not actually need to store $\text{tail}(j) = \text{head}(j+1) - 1$, of course, but we include it in Table 2 to simplify our explanation.) It is within a reasonable constant factor of the most space-efficient implementation and simple to build.

For each suffix $P[i..m-1]$ of P from shortest to longest, MONI finds the length ℓ_i of the longest prefix $P[i..i+\ell_i-1]$ of $P[i..m-1]$ that occurs in T, the lexicographic rank q_i of a suffix of T starting with $P[i..i+\ell_i-1]$, the starting position $\text{SA}[q_i]$ of that suffix in T, and the row j_i of Table 2 such that head(j_i) is the predecessor of q_i in that column. We note that the (pos, len) pairs $(\text{SA}[q_0], \ell_0), \ldots, (\text{SA}[q_{m-1}], \ell_{m-1})$ are the matching statistics $\text{MS}[0..m-1]$ of P with respect to T.

Suppose we know i, ℓ_i, q_i, $\text{SA}[q_i]$ and j_i, and we want to find ℓ_{i-1}, q_{i-1}, $\text{SA}[q_{i-1}]$ and j_{i-1}. If $\text{BWT}[\text{head}(j_i)] = P[i-1]$ then we perform an LF step, as we describe in a moment. If $\text{BWT}[\text{head}(j_i)] \neq P[i-1]$ then we find the last row j_i' above row j_i with $\text{BWT}[\text{head}(j_i')] = P[i-1]$, and the first row j_i'' below row j_i with $\text{BWT}[\text{head}(j_i'')] = P[i-1]$, using rank and select queries on column BWT[head(j)] in Table 2. We use LCE queries to check whether $T[\text{SA}[q_i]..n-1]$ has a longer common suffix with $T[\text{SA}[\text{tail}(j_i')]..n-1]$ or with $T[\text{SA}[\text{head}(j_i'')]..n-1]$ and, depending on that comparison, either reset

$$\begin{aligned}
\ell_i &= \text{LCE}(\text{SA}[q_i], \text{SA}[\text{tail}(j_i')]) \\
q_i &= \text{tail}(j_i') \\
\text{SA}[q_i] &= \text{SA}[\text{tail}(j_i')] \\
j_i &= j_i'
\end{aligned}$$

or reset

$$\begin{aligned}\ell_i &= \mathrm{LCE}(\mathrm{SA}[q_i], \mathrm{SA}[\mathrm{head}(j_i'')]) \\ q_i &= \mathrm{head}(j_i'') \\ \mathrm{SA}[q_i] &= \mathrm{SA}[\mathrm{head}(j_i'')] \\ j_i &= j_i''\,.\end{aligned}$$

Now $\mathrm{BWT}[\mathrm{head}(j_i)] = P[i-1]$, so we can proceed with the LF step.

For example, suppose $P[0..11] = \texttt{TAGATTACATTA}$, $i = 2$ and we have already found $\ell_2 = 8$ (because `GATTACAT` occurs in T but `GATTACATT` does not), $q_2 = 29$, $\mathrm{SA}[q_2] = 0$ and $j_2 = 7$. Since $\mathrm{BWT}[\mathrm{head}(7)] = \$ \neq P[1] = \texttt{A}$, we find $j_2' = 5$ and $j_2'' = 9$ and compare

$$\mathrm{LCE}(0, \mathrm{SA}[\mathrm{tail}(5)]) = \mathrm{LCE}(0, 10) = 3$$

against

$$\mathrm{LCE}(0, \mathrm{SA}[\mathrm{head}(9)]) = \mathrm{LCE}(0, 7) = 0\,.$$

Since the former LCE is longer, we set $\ell_2 = 3$, $q_2 = 27$, $\mathrm{SA}[q_2] = 10$ and $j_2 = 5$.

To perform an LF step with Table 2 when we know i, ℓ_i, q_i, $\mathrm{SA}[q_i]$ and j_i, we first set

$$\begin{aligned}\ell_{i-1} &= \ell_i + 1 \\ q_{i-1} &= \mu(\pi(j_i)) + q_i - \mathrm{head}(j_i) \\ \mathrm{SA}[q_{i-1}] &= \mathrm{SA}[q_i] - 1\,,\end{aligned}$$

where π is the permutation on $\{0, \ldots, r-1\}$ that stably sorts the column $\mathrm{BWT}[\mathrm{head}(j)]$. If we keep $\mathrm{BWT}[\mathrm{head}(j)]$ in a wavelet tree then we have fast access to π.

For our example, consider $i = 2$, $\ell_2 = 3$, $q_2 = 27$, $\mathrm{SA}[q_2] = 10$ and $j_2 = 5$. Since $\mathrm{BWT}[\mathrm{head}(5)]$ is the second `A` in the column $\mathrm{BWT}[\mathrm{head}(j)]$ and there are 4 characters in the column lexicographically strictly less than `A`, $\pi(5) = 5$ and $\mu(5) = 6$, so we set $\ell_1 = 4$, $q_1 = 6 + 27 - 22 = 11$ and $\mathrm{SA}[11] = 9$. Notice π is similar to an LF mapping for the sequence obtained by sampling one character from each run of the BWT (but in our example π has a fixed point at 5); in fact, it permutes the coloured boxes in Table 1 according to LF. It follows that $\mu(\pi(j_i)) = \mathrm{LF}(\mathrm{head}(j_i))$. Since LF maintains the relationship between elements in the same box,

$$\mathrm{LF}(q_i) - \mathrm{LF}(\mathrm{head}(j_i)) = q_i - \mathrm{head}(j_i)\,;$$

substituting and rearranging, we obtain our formula for q_{i-1}.

The last thing left for us to do during an LF step is find j_{i-1}. For this, we use the $\mathrm{finger}(j)$ column. By construction, $\mathrm{head}(\mathrm{finger}(\pi(j_i)))$ is the predecessor of $\mathrm{LF}(\mathrm{head}(j_i))$ in the set

$$\{\mathrm{head}[0], \ldots, \mathrm{head}[r-1]\}\,.$$

Therefore, since $q_{i-1} = \mathrm{LF}(q_i) \geq \mathrm{LF}(\mathrm{head}(j_i))$, we can find the row j_{i-1} of Table 2 such that $\mathrm{head}(j_{i-1})$ is the predecessor of q_{i-1} in that column, by starting an exponential search at row $\mathrm{finger}(\pi(j_i))$. This takes $O(\log r)$ time in the worst case and in practice it takes constant time. Nishimoto and Tabei showed how to guarantee it takes constant time at the cost of increasing the size of Table 2 slightly.

For more formal discussions, we refer readers to previous papers on MONI [13] and the r-index [5, 12, 4, 11].

Table 3 The table we use for ϕ queries and access to the LCP.

j	SA[head(j)]	SA[tail(j)]	LCP[head(j)]	finger(j)
0	0	18	3	9
1	2	38	1	12
2	3	42	2	12
3	5	36	0	12
4	6	9	1	7
5	7	35	0	12
6	8	44	0	13
7	9	39	5	12
8	12	3	6	2
9	18	10	11	7
10	26	0	5	0
11	29	20	2	9
12	32	23	11	9
13	44	37	0	12

3 LCP access

We can support ϕ queries with table lookup as well: for each run BWT[$i..j$] in the BWT, we store SA[i] and SA[$(i-1) \bmod n$] as a row; we sort the rows by their first components; and we add to each row the number of the row containing the predecessor of the second component in the first column. Abusing notation slightly, we refer to the columns of the resulting table as SA[head(j)], SA[tail(j)] and finger(j). Table 3 is for our running example, augmented with a column LCP[head(j)] that stores the length of the longest common prefix of T[SA[head(j)]..$n-1$] and T[SA[tail(j)]..$n-1$]. Since we are storing the row containing the predecessor of each entry in SA[tail(j)] in the column SA[head(j)], we can encode each entry in SA[tail(j)] as the difference between it and its predecessor in SA[head(j)].

Analysis shows the table then takes about $3r\lg(n/r) + r\lg r$ bits: we essentially gap-code the interleaving of column SA[head(j)] and the sorted column SA[tail(j)], which consists of $2r$ sorted numbers between 0 and $n-1$ and thus takes about $2r\lg(n/r)$ bits; Kärkkäinen, Kempa and Piątkowski [6] showed that the entries in LCP[head(j)] sum to $O(n\log r)$ so, by Jensen's Inequality, we can store them in a total of about $r\lg\frac{O(n\log n)}{r} = r\lg(n/r) + r\lg\lg r + O(r)$ bits; and finger(j) takes about $r\lg r$ bits.

To see how we use Table 3 to answer ϕ queries, suppose we know that the predecessor of 24 in SA[head(j)] is in row 9. Then we have

$$\phi(24) = \mathrm{SA}[\mathrm{tail}(9)] + 24 - \mathrm{SA}[\mathrm{head}(9)] = 10 + 24 - 18 = 16\,.$$

We know that the predecessor of 10 in SA[head(j)] is in row finger(9) = 7, but the predecessor of 16 could be in a later row. Again, we perform an exponential search starting in row finger(9) = 7 and find the predecessor 12 of 16 in row 8. Then we have

$$\phi(16) = \mathrm{SA}[\mathrm{tail}(8)] + 16 - \mathrm{SA}[\mathrm{head}(8)] = 3 + 16 - 12 = 7\,.$$

Looking at rows 32 to 34 in Table 1, we see that indeed $\phi(24) = 16$ and $\phi(16) = 7$. This works because, similar to the equation for LF, if BWT[$j-1$] = BWT[j] then $\phi(\mathrm{SA}[j]-1) = \phi(\mathrm{SA}[j]) - 1$. Again, for more formal discussions, we refer readers to previous papers on the r-index [5, 12, 4, 11].

We do not know how to support random access to the LCP array quickly in $O(r \log n)$ bits, but we can use Table 3 to provide a kind of sequential access to it. Specifically, as we use ϕ to enumerate the values in the SA – without necessarily knowing the positions of the cells of the SA those values appear in – we can use similar computations to enumerate the corresponding values in the LCP array. In our example, since the predecessor 18 of 24 in $\mathrm{SA}[\mathrm{head}(j)]$ is in row 9, we can compute the LCP value corresponding to the SA value 24 as

$$\mathrm{LCP}[\mathrm{SA}^{-1}[24]] = \mathrm{LCP}[\mathrm{head}(9)] + \mathrm{SA}[\mathrm{head}(9)] - 24 = 11 + 18 - 24 = 5\,.$$

Checking this, we see that $\mathrm{LCP}[\mathrm{SA}^{-1}[24]] = \mathrm{LCP}[34] = 5$. Since the predecessor 12 of $\phi(24) = 16$ in $\mathrm{SA}[\mathrm{head}(j)]$ is in row 8 of Table 3,

$$\mathrm{LCP}[\mathrm{SA}^{-1}[16]] = \mathrm{LCP}[\mathrm{head}(8)] + \mathrm{SA}[\mathrm{head}(8)] - 16 = 6 + 12 - 16 = 2\,.$$

Checking this, we see that $\mathrm{LCP}[\mathrm{SA}^{-1}[16]] = \mathrm{LCP}[33] = 2$.

Notice we do not use the SA row numbers 34 and 33 to compute the LCP value, as the SA value 24 is sufficient. We could avoid using the inverse suffix array SA^{-1} in our formula by writing $\mathrm{LCP}[\mathrm{SA}^{-1}[24]]$ as $\mathrm{PLCP}[24]$, for example, where $\mathrm{PLCP}[0..n-1]$ denotes the permuted LCP array [7] of T. The kind of sequential access we obtain to the LCP is actually random access to the PLCP array, and it is easier to explain why it works from that perspective – because if $\mathrm{BWT}[j-1] = \mathrm{BWT}[j]$ then $\mathrm{PLCP}[\mathrm{SA}[j]-1] = \mathrm{PLCP}[\mathrm{SA}[j]] + 1$.[2] Nevertheless, we present our results in terms of the LCP and SA^{-1} because we will use them later in conjunction with ϕ queries to enumerate the values in LCP intervals.

Symmetric to using Table 3 to support ϕ queries, we can use a table to support ϕ^{-1} queries. In fact, the $(\mathrm{SA}[\mathrm{head}(j)], \mathrm{SA}[\mathrm{tail}(j)])$ pairs in the table are the same, but sorted by their second components; now we add to each row the number of the row containing the predecessor in the second column of the first component. Since we are storing the row containing the predecessor of each entry in $\mathrm{SA}[\mathrm{head}(j)]$ in the column $\mathrm{SA}[\mathrm{tail}(j)]$, we can encode each entry in $\mathrm{SA}[\mathrm{head}(j)]$ as the difference between it and its predecessor in $\mathrm{SA}[\mathrm{tail}(j)]$. Analysis then shows the table takes about $2r \lg(n/r) + r \lg r$ bits. Table 4 is for supporting ϕ^{-1} queries on our running example. For example, if we know that the predecessor of 7 in $\mathrm{SA}[\mathrm{tail}(j)]$ is in row 1, then we can compute

$$\phi^{-1}(7) = \mathrm{SA}[\mathrm{head}(1)] + 7 - \mathrm{SA}[\mathrm{tail}(1)] = 12 + 7 - 3 = 16$$

and we can find the row containing the predecessor of 16 in $\mathrm{SA}[\mathrm{tail}(j)]$ with an exponential search starting at row $\mathrm{finger}(1) = 3$ (and ending in the same row). We can then compute

$$\phi^{-1}(16) = \mathrm{SA}[\mathrm{head}(3)] + 16 - \mathrm{SA}[\mathrm{tail}(3)] = 18 + 16 - 10 = 24$$

and we can find the row 6 containing the predecessor of 24 in $\mathrm{SA}[\mathrm{tail}(j)]$ with an exponential search starting at row $\mathrm{finger}(3) = 4$.

[2] The formula for PLCP has a +1 where the formula for ϕ has a −1,

$$\begin{aligned}\phi(\mathrm{SA}[j]-1) &= \phi(\mathrm{SA}[j]) - 1\\ \mathrm{PLCP}[\mathrm{SA}[j]-1] &= \mathrm{PLCP}[\mathrm{SA}[j]] + 1\,,\end{aligned}$$

because if $\mathrm{BWT}[j-1] = \mathrm{BWT}[j]$ then moving from j to $LF(j)$ decrements the SA entry but increments the LCP entry.

Table 4 The table we use for ϕ^{-1} queries.

j	SA[head(j)]	SA[tail(j)]	finger(j)
0	26	0	6
1	12	3	3
2	6	9	1
3	18	10	4
4	0	18	0
5	29	20	6
6	32	23	6
7	7	35	11
8	5	36	1
9	44	37	13
10	2	38	0
11	9	39	2
12	3	42	1
13	8	44	1

With these two $O(r \log n)$-bit tables, given k, j and SA[j], we can compute SA[$j - k + 1..j + k - 1$] and LCP[$j - k + 1..j + k - 1$] in $O(k \log r) \subseteq O(k \log n)$ time. (Actually, we can achieve that bound even without the finger(j) columns in the tables, but Brown et al.'s results suggest those will provide a significant speedup in practice.) With Nishimoto and Tabei's modification, we can reduce that to $O(k)$ time while keeping the tables in $O(r \log n)$ bits; this would slightly improve the time bound we give in the next section to $O(m(k + \log n))$.

▶ **Lemma 1.** *We can store two $O(r \log n)$-bit tables such that, given k, j and* SA[j], *we can compute* SA[$j - k + 1..j + k - 1$] *and* LCP[$j - k + 1..j + k - 1$] *in $O(k \log n)$ time.*

4 Finding k-MEMs with Lemma 1

We store the tables described in Sections 2 and 3 for T, which add $O(r \log n)$ bits to MONI. Given P and k, we find the MEMs of P with respect to T as before but then, from each SA[q_i], we use Lemma 1 to find LCP[$q_i - k + 2..q_i + k - 1$] in $O(k \log n)$ time.

For example, suppose that $P[0..11] =$ `TAGATTACATTA`, as in Section 2, and $k = 3$. Starting with $q_{12} = 22$, with MONI we compute the values shown in columns q_i, SA[q_i], ℓ_i and BWT[q_i] of Table 5. (It is important that we choose q_i to be one of the endpoints of a run, since we store SA entries only at those positions, but this is true also for MONI.) The crossed out values are the ones we replace because BWT[q_i] $\neq P[i]$. If we look at the original SA[q_i] and ℓ_i values, before any replacements, we obtain the matching statistics

$$\mathrm{MS}[0..11] = (38, 5), (9, 4), (0, 8), (1, 7), (2, 6), (20, 5), (21, 4), (22, 3), (1, 4), (2, 3), (3, 2), (4, 1)$$

of P with respect to T, with (pos, len) pair MS[i] indicating the starting position MS[i].pos in T of an occurrence of the longest prefix of $P[i..m - 1]$ that occurs in T, and the length MS[i].len of that prefix.

From the matching statistics, it is easy to compute the MEMs $P[0..4] =$ `TAGAT`, $P[2..9] =$ `GATTACAT` and $P[8..11] =$ `ATTA` of P with respect to T: a MEM starts at any position i such that $i = 0$ or MS[$i - 1$].len $\leq$ MS[i].len. For each i, after we compute q_i, SA[q_i] and ℓ_i (and before we replace them, if we do), we use Lemma 1 to compute the sub-interval LCP[$q_i - 1..q_i + 2$] of length $4 = 2k - 2$.

Table 5 With MONI we compute the values shown in columns q_i, SA$[q_i]$, ℓ_i and BWT$[q_i]$ on the left side of the table, and from those we can compute the matching statistics and MEMs of $P[0..11]$ = `TAGATTACATTA` with respect to $T[0..44]$ = `GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$`. After we have computed the values on the left side of the table, we can also compute the values in columns LCP$[q_i - k + 2..q_i + k - 1]$, L_i and $\min(\ell_i, L_i)$ on the right side of the table, and from those we can compute the 3-MEMs of P with respect to T.

i	$P[i]$	q_i	SA$[q_i]$	ℓ_i	BWT$[q_i]$	LCP$[q_i - 1..q_i + 2]$	L_i	$\min(\ell_i, L_i)$
12		22	5	0	`A`			
11	`A`	6	4	1	`T`	[0, 1, 5, 8]	5	1
10	`T`	37	3	2	`T`	[1, 2, 6, 9]	6	2
9	`T`	42	2	3	`A`	[5, 1, 3, 6]	3	3
8	`A`	14 ~~19~~	23 ~~1~~	2 ~~4~~	`C` ~~`G`~~	[10, 2, 4, 7]	4	4
7	`C`	24	22	3	`A`	[4, 7, 0, 3]	4	3
6	`A`	8	21	4	`T`	[5, 8, 1, 4]	5	4
5	`T`	37 ~~39~~	3 ~~20~~	5	`T` ~~`A`~~	[6, 9, 2, 5]	6	5
4	`T`	42	2	6	`A`	[5, 1, 3, 6]	3	3
3	`A`	19	1	7	`G`	[10, 2, 4, 7]	4	4
2	`G`	27 ~~29~~	10 ~~0~~	3 ~~8~~	`A` ~~`$`~~	[11, 3, 5, 8]	5	5
1	`A`	10 ~~11~~	39 ~~9~~	4	`T` ~~`#`~~	[4, 5, 1, 3]	4	4
0	`T`	41	38	5	`T`	[2, 5, 1, 3]	2	2

We scan each interval LCP$[q_i - k + 2..q_i + k - 1]$ in $O(k)$ time and find a sub-interval of length $k - 1$ such that the minimum LCP value L_i in that sub-interval is maximized. This LCP sub-interval corresponds to a sub-interval of length k in SA$[q_i - k + 1..q_i + k - 1]$ containing the starting positions of k suffixes of T – including $T[\mathrm{SA}[q_i]..n - 1]$ itself – whose common prefix with $T[\mathrm{SA}[q_i]..n - 1]$ has the maximum possible length L_i.

In our example, we scan each interval in column LCP$[q_i - 1..q_i + 2]$ of Table 5 and find the sub-interval of length 2 such that the minimum LCP value L_i is maximized. If we check Table 1, we find that the longest prefix of $T[4..44]$ = `ACAT#AGATA`... that occurs at least 3 times in T indeed has length $L_{11} = 5$, the longest prefix of $T[3..44]$ = `TACAT#AGATA`... that occurs at least 3 times in T indeed has length $L_{10} = 6$, and so on.

Since the common prefix of $P[i..m - 1]$ and $T[\mathrm{SA}[q_i]]$ has the maximum possible length ℓ_i, the longest prefix of $P[i..m - 1]$ that occurs at least k times in T has length $\min(\ell_i, L_i)$. Computing $\min(\ell_i, L_i)$ for each i takes a total of $O(km \log n)$ time. The values $\min(\ell_i, L_i)$ are something like a parameterized version of the lengths in the matching statistics: $\min(\ell_i, L_i)$ is the length of the longest prefix of $P[i..m - 1]$ that occurs at least k times in T.

We can compute the k-MEMs of P with respect to T from the $\min(\ell_i, L_i)$ values in the same way we compute MEMs from the lengths in the matching statistics: a k-MEM starts at any position i such that $i = 0$ or $\min(\ell_{i-1}, L_{i-1}) \leq \min(\ell_i, L_i)$. In our example,

$$\begin{aligned}
\min(\ell_0, \ell'_0) &\leq \min(\ell_1, \ell'_1) = 4 \\
\min(\ell_1, \ell'_1) &\leq \min(\ell_2, \ell'_2) = 5 \\
\min(\ell_4, \ell'_4) &\leq \min(\ell_5, \ell'_5) = 5 \\
\min(\ell_7, \ell'_7) &\leq \min(\ell_8, \ell'_8) = 4
\end{aligned}$$

and so the k-MEMs are $P[0..1]$ = `TA`, $P[1..4]$ = `AGAT`, $P[2..6]$ = `GATTA`, $P[5..9]$ = `TACAT` and $P[8..11]$ = `ATTA`.

We can compute $\min(\ell_i, L_i)$ as soon as we have computed SA$[q_i]$ and ℓ_i, so we can compute the k-MEMs of P with respect to T online.

▶ **Theorem 2.** *Suppose we have MONI for a text $T[0..n-1]$ whose BWT consists of r runs. We can add $O(r \log n)$ bits to MONI such that, given $P[0..m-1]$ and k, we can find the k-MEMs of P with respect to T online in $O(k \log n)$ time per character of P.*

5 Finding k-MEMs with precomputed values

Suppose the interval of length k that we find in SA for $P[i..m-1]$, following the procedures in Section 4, is $\mathrm{SA}[s_i..s_i+k-1]$ and $\mathrm{BWT}[s_i] = \cdots = \mathrm{BWT}[s_i+k-1] = P[i-1]$. Then $\min(\ell_{i-1}, L_{i-1}) = \min(\ell_i, L_i) + 1$ and we can find the interval for $P[i-1..m-1]$ with an LF query for s_i, in $O(\log n)$ time. This means we need the results of Section 3 only when at least one character in $\mathrm{BWT}[s_i..s_i+k-1]$ is not equal to $P[i-1]$.

First, suppose $\mathrm{BWT}[q_i] \neq P[i-1]$. Following the procedures in Section 2, MONI resets q_i to the endpoint b of a run in the BWT, resets ℓ_i, and then computes $q_{i-1} = \mathrm{LF}(b)$. Following the procedures in Section 4, we compute $\mathrm{LCP}[q_{i-1}-k+2..q_{i-1}+k-1]$ and scan it to compute the interval $\mathrm{SA}[s_{i-1}..s_{i-1}+k-1]$ for $P[i-1..m-1]$.

If we are given k at construction time, however, then for every endpoint b of a run in the BWT, we can precompute

- the sub-interval of length $k-1$ of $\mathrm{LCP}[\mathrm{LF}(b)-k+2..\mathrm{LF}(b)+k-1]$ that maximizes the minimum value $L(b)$ in the sub-interval,
- that value $L(b)$.

With this information, we do not need the results of Section 3 for this case either, and can handle it in $O(\log n)$ time as well. Since the sub-interval we store for b starts between $\mathrm{LF}(b)-k+2$ and $\mathrm{LF}(b)+k-1$, we can store it in $O(\log k)$ bits as an offset. This means we store $O(r \log k)$ bits on top of at most $2r$ LCP values, or $O(r \log n)$ bits in total.

The remaining case is when $\mathrm{BWT}[q_i] = P[i-1]$ but some of the other characters in $\mathrm{BWT}[s_i..s_i+k-1]$ are not equal to $P[i-1]$. If $\mathrm{BWT}[q_i]$ is the end of a run, then we can proceed as in the previous case in $O(\log n)$ time, using our precomputed values for q_i (but without resetting q_i and ℓ_i). Otherwise, we claim we can choose such a character $\mathrm{BWT}[b] = P[i-1]$ at the end of a run, set

$$\ell_i = \min(\mathrm{LCE}(\mathrm{SA}[q_i], \mathrm{SA}(b)), \ell_i)$$

and $q_i = b$, and then proceed as in the previous case in $O(\log n)$ time, and still be sure of obtaining the correct k-MEMs of P with respect to T. (Continuing to run MONI with the new values of q_i and ℓ_i may not give us the correct MEMs, however.) To be able to change q_i and ℓ_i this way, it is important that we now work online, instead of running MONI on P and then using the results to find the k-MEMs.

To see why our claim holds, assume our query has worked correctly so far, so

$$T[\mathrm{SA}[q_i]..\mathrm{SA}[q_i] + \min(\ell_i, L_i) - 1] = T[\mathrm{SA}[b]..\mathrm{SA}[b] + \min(\ell_i, L_i) - 1]$$

is the longest prefix of $P[i..m-1]$ that occurs at least k times in T. Therefore, the k-MEMs starting in $P[0..i-1]$ are all completely contained in $P[0..i+\min(\ell_i, L_i)-1]$. It follows that resetting

$$\ell_i = \min(\mathrm{LCE}(\mathrm{SA}[q_i], \mathrm{SA}(b)), \ell_i)$$

and $q_i = b$ does not affect the set of k-MEMs we find that start in $P[0..i-1]$.

```
if $\mathrm{BWT}[s_i] = \cdots = \mathrm{BWT}[s_i + k - 1] = P[i-1]$ then
  $q_{i-1} \leftarrow \mathrm{LF}(q_i)$
  $\ell_{i-1} \leftarrow \ell_i + 1$
  $L_{i-1} \leftarrow L_i + 1$
  $s_{i-1} = \mathrm{LF}(s_i)$
else
  if $\mathrm{BWT}[q_i] \neq P[i-1]$ then
    reset $q_i$ and $\ell_i$ as MONI does
  else if $\mathrm{BWT}[q_i]$ is not at the end of a run
    choose $b$ in $[s_i..s_i + k - 1]$ with $\mathrm{BWT}[b] = P[i-1]$ at the end of a run
    $\ell_i \leftarrow \min(\mathrm{LCE}(\mathrm{SA}[q_i], \mathrm{SA}[b]), \ell_i)$
    $q_i \leftarrow b$
  end if
  $q_{i-1} \leftarrow \mathrm{LF}(q_i)$
  $\ell_{i-1} \leftarrow \ell_i + 1$
  $L_{i-1} \leftarrow L(q_i)$
  $s_{i-1} \leftarrow \mathrm{LF}(q_i) - \mathrm{offset}(q_i)$
end if
```

Figure 1 Pseudo-code for how we find k-MEMs with precomputed values.

Figure 1 shows pseudo-code for how we find k-MEMs with precomputed values. Table 6 shows the offsets and $L(b)$ values for our example, surrounded by coloured boxes on the right, with each offset indicating how far above $\mathrm{LF}(b)$ the sub-interval starts. The coloured boxes on the left indicate the sub-interval itself and the longest common prefix of the suffixes starting in the sub-interval of the SA.

For our example, suppose we again start with $q_{12} = 22$ and $\ell_{12} = 0$. Since $\mathrm{BWT}[q_{12}] = P[11] = \mathtt{A}$, we set $q_{11} = \mathrm{LF}(22) = 6$ and $\ell_{11} = \ell_{12} + 1 = 1$. The values $\mathrm{offset}(22) = 0$ and $L(22) = 5$ in the black rectangle in Table 6 tell us to set $s_{11} = \mathrm{LF}(22) - 0 = 6$ and $L_{11} = 5$. This means the suffixes of T with starting points in

$$\mathrm{SA}[6..8] = [4, 13, 21]$$

have a longest common prefix of length 5, which starts with the longest prefix of $P[11]$ that occurs at least 3 times in T. This longest prefix has length $\min(\ell_{11}, L_{11}) = 1$ – so it is just $P[11] = \mathtt{A}$. After this initial setup, we can fill in Table 7 according to the pseudo-code in Figure 1, with crossed out values again indicating those that are replaced.

▶ **Theorem 3.** *Suppose we have MONI for a text $T[0..n-1]$ whose BWT consists of r runs. Given k, we can add $O(r \log n)$ bits to MONI such that, given $P[0..m-1]$, we can find the k-MEMs of P with respect to T online in $O(\log n)$ time per character of P.*

6 Conclusion

We have shown, first, how we can add $O(r \log n)$ bits to MONI for a text $T[0..n-1]$, where r is the number of runs in the BWT of T, such that if we are given k at query time with $P[0..m-1]$, then we can find the k-MEMs of P with respect to T online in $O(k \log n)$ time per character of P. We have then shown how, if we are given k at construction time, we can add $O(r \log k)$ bits and at most $2r$ LCP values – which are $O(r \log n)$ bits in total – such

Table 6 The table showing the precomputed values we use to find 3-MEMs with respect to our example $T =$ GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$.

i	SA[i]	LCP[i]	lexicographically ith cyclic shift of T	BWT[i]	LF(i)	offset(i)	$L(i)$
0	44	0	$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA	A	5	0	1
1	8	0	#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT	T	32	0	2
2	17	1	#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT	T	33		
3	25	4	#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT	T	34		
4	34	9	#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT	T	35		
5	43	0	A$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGAT	T	36		
6	4	1	ACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATT	T	37		
7	13	5	ACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGAT	T	38		
8	21	8	ACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GAT	T	39		
9	30	1	AGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATT	T	40		
10	39	4	AGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATT	T	41	2	2
11	9	5	AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#	#	1	0	1
12	6	1	AT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTAC	C	22	0	4
13	15	3	AT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATAC	C	23		
14	23	6	AT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATAC	C	24	2	4
15	32	11	AT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAG	G	25	0	3
16	41	2	ATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAG	G	26		
17	11	3	ATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AG	G	27		
18	19	10	ATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#G	G	28		
19	1	2	ATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$G	G	29		
20	27	4	ATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#G	G	30		
21	36	7	ATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#G	G	31	2	5
22	5	0	CAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTA	A	6	0	5
23	14	4	CAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATA	A	7		
24	22	7	CAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATA	A	8		
25	31	0	GAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTA	A	9		
26	40	3	GATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTA	A	10		
27	10	4	GATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#A	A	11	2	4
28	18	11	GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#	#	2	0	4
29	0	3	GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$	$	0	0	0
30	26	5	GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#	#	3	1	4
31	35	8	GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#	#	4	2	4
32	7	0	T#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACA	A	12	0	3
33	16	2	T#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACA	A	13		
34	24	5	T#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACA	A	14		
35	33	10	T#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGA	A	15		
36	42	1	TA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGA	A	16	0	3
37	3	2	TACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GAT	T	42	0	3
38	12	6	TACAT#GATACAT#GATTAGAT#GATTAGATA$GATTACAT#AGA	A	17	1	3
39	20	9	TACAT#GATTAGAT#GATTAGATA$GATTACAT#AGATACAT#GA	A	18	2	3
40	29	2	TAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GAT	T	43	1	3
41	38	5	TAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GAT	T	44	2	3
42	2	1	TTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$GA	A	19	0	4
43	28	3	TTAGAT#GATTAGATA$GATTACAT#AGATACAT#GATACAT#GA	A	20		
44	37	6	TTAGATA$GATTACAT#AGATACAT#GATACAT#GATTAGAT#GA	A	21	2	4

Table 7 The values we compute (except BWT$[s_i..s_i+2]$, which we include here only for clarity) while finding the 3-MEMs of $P[0..11]$ = `TAGATTACATTA` with respect to our example T = `GATTACAT#AGATACAT#GATACAT#GATTAGAT#GATTAGATA$`.

i	q_i	ℓ_i	L_i	$\min(\ell_i, L_i)$	s_i	$P[i-1]$	BWT$[q_i]$	BWT$[s_i..s_i+2]$
12	22	0				A	A	
11	6	1	5	1	6	T	T	TTT
10	37	2	6	2	37	T	T	TAA
9	42	3	3	3	42	A	A	AAA
8	14 ~~19~~	2 ~~4~~	4	4	19	C	C ~~G~~	GGG
7	24	3	4	3	22	A	A	AAA
6	8	4	5	4	6	T	T	TTT
5	37 ~~39~~	5	6	5	37	T	T ~~A~~	TAA
4	42	6	3	3	42	A	A	AAA
3	19	7	4	4	19	G	G	GGG
2	27 ~~29~~	3 ~~8~~	5	5	29	A	A ~~$~~	$##
1	10 ~~11~~	4	4	4	9	T	T ~~#~~	TT#
0	41	5	2	2	39		T	ATT

that we can find the k-MEMs of P with respect to T online in $O(\log n)$ time per character of P. Along the way, we have also shown how to extend ϕ queries to support sequential access to the LCP, which may be of independent interest.

Although we have not discussed construction, we expect it will not be difficult to modify prefix-free parsing [2] to build our tables for ϕ, LCP and ϕ^{-1} queries. Once we can support those queries, we can use them to compute k-MEMs in $O(km \log n)$ time, or to build in $O(kr)$ time the table of precomputed values that we need to compute k-MEMs in $O(m \log n)$ time. In fact, once we have built the tables for ϕ, LCP and ϕ^{-1} queries – which take $O(r)$ space but may be significantly larger than our table of precomputed values – then we can store them in external memory and recover them only when we want to build a table of precomputed values for a different choice of k.

We believe our approach is a practical extension of MONI and we are currently implementing it. One possible application might be to index two genomic databases (possibly with two different values of k), one of haplotypes from people with symptoms of a genetic disease and one of haplotypes from people without; then, as the first step in a bioinformatics pipeline, we could use those indexes to mine for substrings that are common in one database and not in the other. We think this application is interesting because, except for a remark in Bannai et al.'s paper about potentially applying MEM-finding to rare-disease diagnosis, the r-index and MONI have so far been considered only as tools for pangenomic *alignment*, and this is an application to pangenomic *analysis*. If the disease is recessive or multifactorial then variations associated with it are likely to be present in both databases, so MEM-finding is unlikely to detect them; those variations could be more frequent in the first database, however, so k-MEM-finding may still be useful.

References

1 Hideo Bannai, Travis Gagie, and Tomohiro I. Refining the r-index. *Theoretical Computer Science*, 812:96–108, 2020.

2 Christina Boucher, Travis Gagie, Tomohiro I, Dominik Köppl, Ben Langmead, Giovanni Manzini, Gonzalo Navarro, Alejandro Pacheco, and Massimiliano Rossi. PHONI: Streamed matching statistics with multi-genome references. In *2021 Data Compression Conference (DCC)*, pages 193–202. IEEE, 2021.

3 Christina Boucher, Travis Gagie, Alan Kuhnle, Ben Langmead, Giovanni Manzini, and Taher Mun. Prefix-free parsing for building big BWTs. *Algorithms for Molecular Biology*, 14(1):1–15, 2019.

4 Nathaniel K. Brown, Travis Gagie, and Massimiliano Rossi. RLBWT tricks. In *20th Symposium on Experimental Algorithms (SEA 2022)*, pages 16:1–16:16, 2022.

5 Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM (JACM)*, 67(1):1–54, 2020.

6 Juha Kärkkäinen, Dominik Kempa, and Marcin Piątkowski. Tighter bounds for the sum of irreducible LCP values. *Theoretical Computer Science*, 656:265–278, 2016.

7 Juha Kärkkäinen, Giovanni Manzini, and Simon J Puglisi. Permuted longest-common-prefix array. In *20th Symposium on Combinatorial Pattern Matching (CPM)*, pages 181–192. Springer, 2009.

8 Heng Li. Aligning sequence reads, clone sequences and assembly contigs with BWA-MEM. *arXiv preprint*, 2013. arXiv:1303.3997.

9 Gonzalo Navarro. *Compact Data Structures: A Practical Approach.* Cambridge University Press, 2016.

10 Gonzalo Navarro. Computing MEMs on repetitive text collections. *arXiv preprint v3*, 2022. Accepted to this conference. arXiv:2210.09914.

11 Takaaki Nishimoto, Shunsuke Kanda, and Yasuo Tabei. An optimal-time RLBWT construction in BWT-runs bounded space. In *49th International Colloquium on Automata, Languages, and Programming (ICALP 2022)*, pages 99:1–99:20, 2022.

12 Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on BWT-runs compressed indexes. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, 2021.

13 Massimiliano Rossi, Marco Oliva, Ben Langmead, Travis Gagie, and Christina Boucher. MONI: A pangenomic index for finding maximal exact matches. *Journal of Computational Biology*, 29(2):169–187, 2022.