Compact Data Structures for Shortest Unique Substring Queries

Takuya Mieno[1], Dominik Köppl[1,2][0000−0002−8721−4444], Yuto Nakashima[1], Shunsuke Inenaga[1], Hideo Bannai[1][0000−0002−6856−5185], and Masayuki Takeda[1]

[1] Department of Informatics, Kyushu University, Japan
[2] Japan Society for Promotion of Science, Japan
{takuya.mieno,dominik.koeppl,
yuto.nakashima,inenaga,bannai,takeda}@inf.kyushu-u.ac.jp

Abstract. Given a string T of length n, a substring $u = T[i..j]$ of T is called a shortest unique substring (SUS) for an interval $[s, t]$ if (a) u occurs exactly once in T, (b) u contains the interval $[s, t]$ (i.e. $i \leq s \leq t \leq j$), and (c) every substring v of T with $|v| < |u|$ containing $[s, t]$ occurs at least twice in T. Given a query interval $[s, t] \subset [1, n]$, the *interval SUS problem* is to output all the SUSs for the interval $[s, t]$. In this article, we propose a $4n+o(n)$ bits data structure answering an interval SUS query in output-sensitive $O(occ)$ time, where occ is the number of returned SUSs. Additionally, we focus on the *point SUS problem*, which is the interval SUS problem for $s = t$. Here, we propose a $\lceil (\log_2 3 + 1)n \rceil + o(n)$ bits data structure answering a point SUS query in the same output-sensitive time.

Keywords: string processing algorithm · shortest unique substring · compact data structure.

1 Introduction

A substring $u = T[i..j]$ of a string T is called a *shortest unique substring (SUS)* for an interval $[s, t]$ if (a) u occurs exactly once in T, (b) u contains the interval $[s, t]$ (i.e., $i \leq s \leq t \leq j$), and (c) every substring v of T with $|v| < |u|$ containing $[s, t]$ occurs at least twice in T. Given a query interval $[s, t] \subset [1, n]$, the *interval SUS problem* is to output all the SUSs for $[s, t]$. When a query interval consists of a single position (i.e., $s = t$), the SUS problem becomes a so-called *point* SUS problem.

Point SUS Problem. The point SUS problem was introduced by Pei et al. [12]. This problem is motivated by applications in bioinformatics like genome comparisons [4] or PCR primer design [12]. Pei et al. tackled this problem with an $O(n)$ words data structure that can return one SUS for a given query position in constant time. They can compute this data structure in $O(n^2)$ time with $O(n)$ space. Based on that result, Tsuruta et al. [13] provided an $O(n)$ words data structure answering the same query (returning one SUS) in constant time. Their

data structure can be constructed in $O(n)$ time. İleri et al. [7] independently showed another data structure with the same time complexities. For the general point SUS problem, Tsuruta et al. [13] can also resort to their proposed data structure returning all SUSs for a query position in optimal $O(occ)$ time, where occ is the number of returned SUSs.

The aforementioned data structures all take $\Theta(n)$ words. This space can become problematic for large n. This problem was perceived by Hon et al. [5], who proposed a data structure consisting of the input string T and two integer arrays, each of length n. Both arrays store, respectively, the beginning and the ending position of a SUS for each position i with $1 \leq i \leq n$. Hon et al. provided an algorithm that can construct these two arrays in linear time with $O(\log n)$ bits of additional working space, given that both arrays are stored in $2n \log n$ bits and that $\sigma \leq n$. Instead of building a data structure, Ganguly et al. [3] proposed a time-space trade-off algorithm using $O(n/\tau)$ words of additional working space, answering a given query in $O(n\tau^2 \log \frac{n}{\tau})$ time directly, for a trade-off parameter $\tau \geq 1$. They also proposed the first *compact* data structure of size $4n + o(n)$ bits that can answer a query in constant time. They can construct this data structure in $O(n \log n)$ time using $O(n \log \sigma)$ bits of additional working space.

Interval SUS Problem. Hu et al. [6] were the first to consider the interval SUS problem. They proposed a data structure answering a query returning all SUSs for the respective query interval in $O(occ)$ optimal time after $O(n)$ time preprocessing. In the compressed setting, Mieno et al. [11] considered the interval SUS problem when the input string T is given *run-length encoded (RLE)*, and proposed a data structure of size $O(r)$ words answering a query by returning all SUSs for the respective query interval in $O(\sqrt{\log r / \log \log r} + occ)$ time, where r is the number of single character runs in T.

Our Contribution. In this paper, we propose the following two data structures:

(A) A data structure of size $2n + 2m + o(n)$ bits answering an interval SUS query in $O(occ)$ time, where m is the number of *minimal unique substrings* of the input string[3], and occ is the number of SUSs of T for the respective query interval (Theorem 1).
(B) A data structure of size $\lceil (\log_2 3 + 1)n \rceil + o(n)$ bits answering a point SUS query in $O(occ)$ time, where occ is the number of SUSs of T for the respective query point (Theorem 2).

Instead of outputting the answer as a list of substrings of T, it is sometimes sufficient to output only the intervals corresponding to the respective substrings. In such a case, both data structures can answer a query *without* the need of the input string. The data structure (A) is the first data structure of size $O(n)$ bits for the interval SUS problem. Also, the data structure (B) is the first data

---

[3] We show later in Lemma 1 that the number of minimal unique substrings m is at most n.

structure of size $O(n)$ bits for the point SUS problem, returning *all* SUSs for a given query position. Notice that the data structure of Ganguly et al. [3] uses $4n + o(n)$ bits of space, but returns only one SUS for a point SUS query.

2 Preliminaries

Our model of computation is the word RAM with machine word size $\Omega(\log n)$.

2.1 Strings

Let Σ be an alphabet. An element of Σ^* is called a *string*. For $|\Sigma| = 2$, we call a string also a bit array. The length of a string T is denoted by $|T|$. The empty string ε is the string of length 0. Given a string T, the i-th character of T is denoted by $T[i]$, for an integer i with $1 \le i \le |T|$. For two integers i and j with $1 \le i \le j \le |T|$, a substring of T starting at position i and ending at position j is denoted by $T[i..j]$. Namely, $T[i..j] = T[i]T[i+1]\cdots T[j]$. For two strings T and w, the number of occurrences of w in T is denoted by $\#T(w) := |\{i \mid T[i..i + |w| - 1] = w\}|$. For two intervals $[i, j]$ and $[x, y]$, let $cover([i, j], [x, y]) := [\min\{i, x\}, \max\{j, y\}]$ denote the shortest interval that contains the text positions i, j, x, and y. If the interval $[x, y]$ consists of a single point, i.e., $x = y$, $cover([i, j], [x, y])$ is denoted by $cover([i, j], x)$ when we want to emphasize on the fact that $x = y$.

In what follows, we fix a string T of length $n \ge 1$ whose characters are drawn from an integer alphabet Σ of size $\sigma = n^{O(1)}$.

2.2 MUSs and SUSs

Let u be a non-empty substring of T. u is called a *repeating substring* of T if $\#T(u) \ge 2$, and u is called a *unique substring* of T if $\#T(u) = 1$. Since every unique substring $u = T[i..j]$ of T occurs exactly once in T, we identify u with its corresponding interval $[i, j]$. We also say that the interval $[i, j]$ is unique iff the corresponding substring $T[i..j]$ is a unique substring of T.

A unique substring $u = T[i..j]$ of T is said to be a *minimal unique substring (MUS)* of T iff every proper substring of u is a repeating substring, i.e., $\#T(T[i'..j']) \ge 2$ for every integer i' and every integer j' with $[i', j'] \subset [i, j]$ and $j' - i' < j - i$. Let $\mathsf{MUS}_T := \{[i, j] \mid T[i..j] \text{ is a MUS of } T\}$ be the set of all intervals corresponding to the MUSs of T. From the definition of MUSs, the next lemma follows:

Lemma 1 ([13, Lemma 2]). *No element of MUS_T is nested in another element of MUS_T, i.e., two different MUSs $[i, j], [k, l] \in \mathsf{MUS}_T$ satisfy $[i, j] \not\subset [k, l]$ and $[k, l] \not\subset [i, j]$. Therefore, $0 < |\mathsf{MUS}_T| \le |T|$.*

We use the following two sets containing interval and point SUSs, which were defined at the beginning of the introduction: Given an interval $[s, t] \subset [1, n]$, $\mathsf{SUS}_T([s, t])$ denotes the set of the interval SUSs of T for the interval $[s, t]$. Given
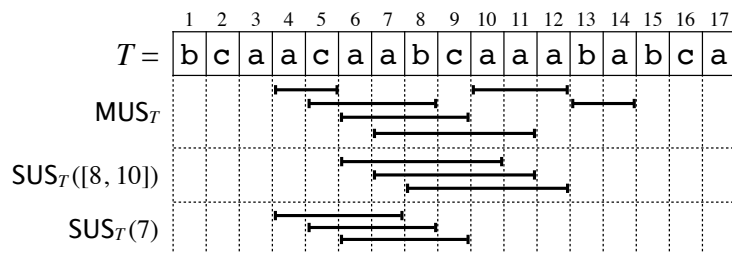
Fig. 1. The string $T = \texttt{bcaacaabcaaababca}$, and its set $\mathsf{MUS}_T = \{[4,5], [5,8], [6,9],$ $[7,11], [10,12], [13,14]\}$. MUS_T corresponds to the set $\{\texttt{ac}, \texttt{caab}, \texttt{aabc}, \texttt{abcaa}, \texttt{aaa}, \texttt{ba}\}$ of all MUSs of T. The substrings $T[6..10] = \texttt{aabca}$, $T[7..11] = \texttt{abcaa}$, and $T[8..12] = \texttt{bcaaa}$ are SUSs for the query interval $[8, 10]$. Also, the substrings $T[4..7] = \texttt{acaa}$, $T[5..8] = \texttt{caab}$, and $T[6..9] = \texttt{aabc}$ are SUSs for the query position 7. The later defined leftmost/rightmost SUS and MUS (cf. Sect. 6) for $p = 7$ are $\mathsf{lmSUS}_T^p = [4,7]$, $\mathsf{lmMUS}_T^p = [4,5]$, $\mathsf{rmSUS}_T^p = [6,9]$, and $\mathsf{rmMUS}_T^p = [6,9]$.

a text position $p \in [1, n]$, $\mathsf{SUS}_T(p)$ denotes the set of the point SUSs of T for the point p.

Given a query position $p \in [1, n]$ (resp. a query interval $[s, t] \subset [1, n]$), the *point* (resp. *interval*) *SUS problem* is to compute $\mathsf{SUS}_T(p)$ (resp. $\mathsf{SUS}_T([s, t])$). See Fig. 1 for an example depicting MUSs and SUSs.

3   Tools

In this section, we introduce the data structures needed for our approach solving both SUS problems.

3.1   Rank and Select

Given a string X of length n over the alphabet $[1, \sigma]$. For an integer i with $1 \le i \le n$ and a character $c \in [1, \sigma]$, the rank query $rank_X(c, i)$ returns the number of the character c in the prefix $X[1..i]$ of X. Also, the select query $select_X(c, i)$ returns the position of X containing the i-th occurrence of the character c (or returns the invalid symbol *nil* if such a position does not exist). For $\sigma = 2$ (i.e., X is a bit array), we can make use of the following lemma:

Lemma 2 ([8, 1]). *We can endow a bit array X of length n with a data structure answering $rank_X$ and $select_X$ in constant time. This data structure takes $o(n)$ bits of space, and can be built on X in $O(n)$ time with $O(\log n)$ bits of additional working space.*

3.2   Predecessor and Successor

Let Y be an array of length k whose entries are positive integers in strictly increasing order. Further suppose that these integers are less than or equal to n.

Given an integer d with $1 \leq d \leq n$, the *predecessor* and the *successor query* on Y with d are defined as $\mathsf{Pred}_Y(d) := \max\{i \mid Y[i] \leq d\}$ and $\mathsf{Succ}_Y(d) := \min\{i \mid Y[i] \geq d\}$, where we stipulate that $\min\{\} = \max\{\} = nil$.

Let BIT_Y be a bit array of length n marking all integers present in Y, i.e., $BIT_Y[i] = \mathtt{1}$ iff there is an integer j with $1 \leq j \leq k$ and $Y[j] = i$, for every i with $1 \leq i \leq n$. By endowing BIT_Y with a rank/select data structure, we yield an $n + o(n)$ bits data structure answering $\mathsf{Pred}_Y(d) = select_{BIT_Y}(\mathtt{1}, rank_{BIT_Y}(\mathtt{1}, d))$ and $\mathsf{Succ}_Y(d)$[4] in constant time for each d with $1 \leq d \leq n$.

3.3   RmQ and RMQ

Given an integer array Z of length n and an interval $[i, j] \subset [1, n]$, the range minimum query $\mathsf{RmQ}_Z(i, j)$ (resp. the range maximum query $\mathsf{RMQ}_Z(i, j)$) asks for the index p of a minimum element (resp. a maximum element) of the subarray $Z[i..j]$, i.e., $p \in \arg\min_{i \leq k \leq j} Z[k]$, or respectively $p \in \arg\max_{i \leq k \leq j} Z[k]$. We use the following well-known data structure to handle these kind of queries:

Lemma 3 ([2]). *Given an integer array Z of length n, there is an RmQ (resp. RMQ) data structure taking $2n + o(n)$ bits of space that can answer an RmQ (resp. RMQ) query on Z in constant time. This data structure can be constructed in $O(n)$ time with $o(n)$ bits of additional working space.*

3.4   Suffix Array, Inverse Suffix Array and LCP Array

We define the three integer arrays $\mathsf{SA}_T[1..n]$, $\mathsf{ISA}_T[1..n]$, and $\mathsf{LCP}_T[1..n + 1]$. The suffix array SA_T of T is the array with the property that $T[\mathsf{SA}_T[i]..n]$ is lexicographically smaller than $T[\mathsf{SA}_T[i + 1]..n]$ for every i with $1 \leq i \leq n - 1$ [10]. The inverse suffix array ISA_T of T is the inverse of SA_T, i.e., $\mathsf{SA}_T[\mathsf{ISA}_T[i]] = i$ for every i with $1 \leq i \leq n$. The LCP array LCP_T of T is the array with the property that $\mathsf{LCP}_T[1] = \mathsf{LCP}_T[n+1] = 0$ and $\mathsf{LCP}_T[i] = lcp(T[\mathsf{SA}_T[i]..n], T[\mathsf{SA}_T[i-1]..n])$ for every i with $2 \leq i \leq n$, where $lcp(P, Q)$ denotes the length of the longest common prefix of P and Q for two given strings P and Q.

4   Computing MUSs in Compact Space

For computing SUSs efficiently, it is advantageous to have a data structure available that can retrieve MUSs starting or ending at specific positions, as the following lemma gives a crucial connection between MUSs and SUSs:

Lemma 4 ([13, Lemma 2]). *Every point SUS contains exactly one MUS.*

Fig. 2 gives an overview of our introduced data structure and shows the connections between this section and the following sections that focus on our two SUS problems. For our data structure retrieving MUSs, we propose a compact

---

[4] $\mathsf{Succ}_Y(d)$ can be computed similarly by considering the case whether $BIT_Y[d] = \mathtt{1}$.

representation and an algorithm to compute this representation space-efficiently. Our data structure is based on the following two bit arrays MB_T and ME_T of length n with the properties that

- $\mathsf{MB}_T[i] = \mathtt{1}$ iff i is the beginning position of a MUS, and
- $\mathsf{ME}_T[i] = \mathtt{1}$ iff i is the ending position of a MUS.

For the rest of this paper, let m be the number of MUSs in T. We rank the MUSs by their starting positions in the text, such that the j-th MUS starts before the $(j+1)$-th MUS, for every integer j with $1 \leq j \leq m-1$.

Since MUSs are not nested (see Lemma 1), the number of $\mathtt{1}$'s in MB_T and ME_T is exactly m. Hence, the starting position, the ending position, and the length of the j-th MUS can be computed with rank/select queries for every integer j with $1 \leq j \leq m$. How MB_T and ME_T can be computed is shown in the following lemma:

Lemma 5. *Let $\mathcal{D}_T$ be a data structure that can access $\mathsf{ISA}_T[i]$ and $\mathsf{LCP}_T[i]$ in $\pi_a(n)$ time for every position i with $1 \leq i \leq n$. Suppose that we can construct it in $\pi_c(n)$ time with $\pi_s(n)$ bits of working space including the space for $\mathcal{D}_T$. Then MB_T and ME_T can be computed in $O(\pi_c(n) + n \cdot \pi_a(n))$ total time while using $2n + \pi_s(n)$ bits of total working space including the space for MB_T and ME_T.*

Proof. Given a text position i with $1 \leq i \leq n$, $T[i..i+\ell_i-1]$ with $\ell_i = \max\{\mathsf{LCP}_T[\mathsf{ISA}_T[i]], \mathsf{LCP}_T[\mathsf{ISA}_T[i]+1]\}$ is the longest repeating substring starting at i. If we extend this substring by the character to its right, it becomes unique. Thus, $T[i..i+\ell_i]$ is the shortest unique substring starting at i, except for the case that $i + \ell_i - 1 = n$ as we cannot extend it to the right (hence, there is no unique substring starting at i in this case). Additionally, the substring $T[i..i+\ell_i]$ is a MUS iff $T[i+1..i+\ell_i]$ is not unique (we already checked that $T[i..i+\ell_i-1]$ is not unique). $T[i+1..i+\ell_i]$ is not unique iff $\ell_i \leq \ell_{i+1}$ since $T[i+1..i+1+\ell_{i+1}]$ is the smallest unique substring starting at $i+1$. Since each ℓ_i can be computed in $O(\pi_a(n))$ time for every $1 \leq i \leq n$, the starting and ending positions of all MUSs (and hence, MB_T and ME_T) can be computed in $O(n \cdot \pi_a(n))$ time by a linear scan of the text. Therefore, the total computing time is $O(\pi_c(n) + n \cdot \pi_a(n))$ and the total working space is $2n + \pi_s(n)$ bits including the space for MB_T and ME_T.
□

5   Compact Data Structure for the Interval SUS Problem

In this section, we propose a compact data structure for the interval SUS problem. It is based on the data structure of Mieno et al. [11], which we review in the following. We subsequently provide a compact representation of this data structure.
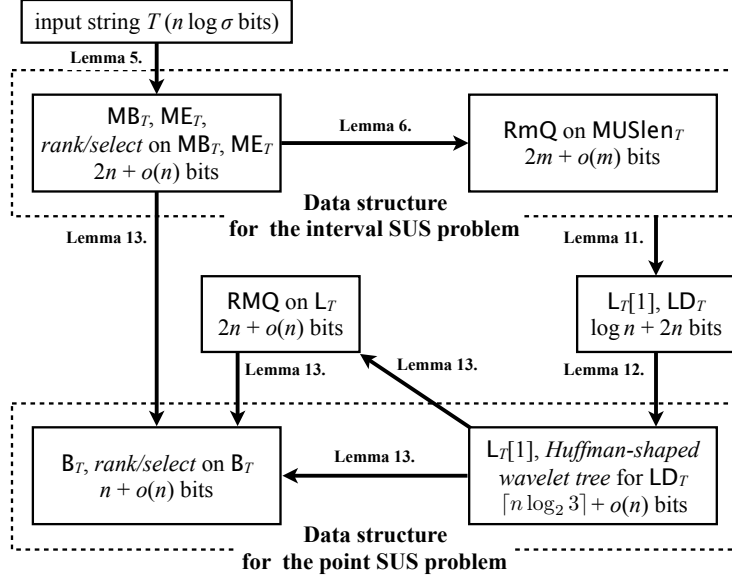
Fig. 2. Overview of the data structures proposed for solving the interval SUS and point SUS problem. Nodes are data structures. Edges of the same label (labeled by a certain lemma) describe an algorithm taking a set of input data structures to produce a data structure.

Data structures. The data structure proposed by Mieno et al. [11] consists of three arrays, each of length m: X_T, Y_T, and MUSlen_T. The arrays X_T and Y_T store, respectively, the beginning positions and ending positions of all MUSs sorted by their beginning positions such that the interval $[X_T[i], Y_T[i]]$ is the i-th MUS, for every integer i with $1 \leq i \leq m$. Further, $\mathsf{MUSlen}_T[i] = Y_T[i] - X_T[i] + 1$ stores the length of i-th MUS. During a preprocessing phase, X_T and Y_T are endowed with a successor and a predecessor data structure, respectively. Further, MUSlen_T is endowed with an RmQ data structure.

Answering queries. Given a query interval $[s, t]$, let $\ell = \mathsf{Pred}_{Y_T}(t)$ be the index in Y_T of the largest ending position of a MUS that is at most t, and $r = \mathsf{Succ}_{X_T}(s)$ be the index in X_T of the smallest starting position of a MUS that is at least s. Then, $\mathsf{SUS}_T([s, t]) \subset \{cover([s, t], [X_T[i], Y_T[i]]) \mid \ell \leq i \leq r\}$. That is because the shortest intervals in $\{cover([s, t], [X_T[i], Y_T[i]]) \mid \ell \leq i \leq r\}$ correspond to the shortest unique substrings (SUSs) among all substrings covering the interval $[s, t]$. Thus, one of the SUSs for $[s, t]$ can be detected by considering $cover([s, t], [X_T[\ell], Y_T[\ell]])$ (as a candidate for the leftmost SUS), $cover([s, t], [X_T[r], Y_T[r]])$ (as a candidate for the rightmost SUS), and $\mathsf{RmQ}_{\mathsf{MUSlen}_T}(\ell + 1, r - 1)$. To output all SUSs, it is sufficient to answer RmQ queries on subintervals of $\mathsf{MUSlen}_T[\ell+1..r-1]$ recursively. In detail, suppose that there is a MUS in $\mathsf{MUSlen}_T[\ell+1..r-1]$ that is a SUS for $[s, t]$. Further suppose

that this is the j-th MUS having length k. Then we query $\mathsf{MUSlen}_T[\ell + 1..j - 1]$ and $\mathsf{MUSlen}_T[j + 1..r - 1]$ for all other MUSs of minimal length k.

Compact Representation. Having the two bit arrays MB_T and ME_T of Section 4, we can simulate the three arrays X_T, Y_T, and MUSlen_T. By endowing these two bit arrays with rank/select data structures of Lemma 2, we can compute rank/select in constant time, which allows us to compute the value of $\mathsf{X}_T[p]$, $\mathsf{Y}_T[p]$, $\mathsf{MUSlen}_T[p]$, $\mathsf{Pred}_{Y_T}(q)$ and $\mathsf{Succ}_{X_T}(q)$ for every index p with $1 \leq p \leq m$ and every text position q with $1 \leq q \leq n$ in constant time while using only $2n + o(n)$ bits of total space. By endowing MUSlen_T with the RmQ data structure of Lemma 3, we can answer an RmQ query on MUSlen_T in constant time. This data structure takes $2m + o(m)$ bits of space. Altogether, with these data structures we yield the following theorem:

Theorem 1. *For the interval SUS problem, there exists a data structure of size $2n + 2m + o(n)$ bits that can answer an interval SUS query in $O(occ)$ time, where occ is the number of SUSs of T for the respective query interval.*

Also, the data structure can be constructed space-efficiently:

Lemma 6. *Given MB_T and ME_T, the data structure proposed in Theorem 1 can be constructed in $O(n)$ time using $2m + o(n)$ bits of total working space, which includes the space for this data structure.*

Proof. The data structure proposed in Theorem 1 consists of the two bit arrays MB_T, ME_T, and an RmQ data structure on MUSlen_T, which is simulated by rank/select data structures on MB_T and ME_T. Since MB_T and ME_T are already given, it is left to endow MB_T and ME_T with rank/select data structures (using Lemma 2), and to compute the RmQ data structure on MUSlen_T (using Lemma 3). □

6   Compact Data Structure for the Point SUS Problem

Before solving the point SUS problem, we borrow some additional notations from Tsuruta et al. [13] to deal with point SUS queries. This is necessary since some of the MUSs never take part in finding a SUS such that there is no meaning to compute and store them. Since we want to provide an output-sensitive algorithm answering a query in optimal time, we only want to store MUSs that are candidates for being a SUS.

We say that the interval $[x, y] \in \mathsf{MUS}_T$ is a *meaningful* MUS if $T[x..y]$ is a substring of (or equal to) a point SUS, i.e., $cover([x, y], p) \in \mathsf{SUS}_T(p)$ for a position p. Also, we say that the interval $[x, y] \in \mathsf{MUS}_T$ is a *meaningless* MUS if $[x, y]$ is not a meaningful MUS. Let

$$\mathsf{MMUS}_T := \{[i, j] \in \mathsf{MUS}_T \mid \text{there exists a } p \text{ with } 1 \leq p \leq n$$
$$\text{such that } cover([i, j], p) \in \mathsf{SUS}_T(p)\}$$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$T =$	b	c	a	a	c	a	a	b	c	a	a	a	b	a	b	c	a
B_T	0	0	0	1	1	1	0	0	0	1	0	0	1	0	0	0	0
L_T	5	4	3	2	2	3	4	4	4	3	3	3	2	2	3	4	5
$pred1pos_{\mathsf{B}_T}$	nil	nil	nil	4	5	6	6	6	6	10	10	10	13	13	13	13	13
$succ1pos_{\mathsf{B}_T}$	4	4	4	4	5	6	10	10	10	10	13	13	13	nil	nil	nil	nil
$predneq_{\mathsf{L}_T}$	nil	1	2	3	3	5	6	6	6	9	9	9	12	12	14	15	16
$succneq_{\mathsf{L}_T}$	2	3	4	6	6	7	10	10	10	13	13	13	15	15	16	17	nil

(The MUS_T row between T and B_T shows the MUS intervals; the interval $T[7..11] =$ *meaningless*.)

Fig. 3. $\mathsf{MUS}_T, \mathsf{B}_T, \mathsf{L}_T$, and the four functions defined in at the beginning of Section 6 for the string $T = \texttt{bcaacaabcaaababca}$. $\mathsf{B}_T[7] = 0$ because the MUS $T[7..11] = \texttt{abcaa}$ is meaningless.

denote the set of all meaningful MUSs of T.

Let lmSUS_T^p denote the interval in $\mathsf{SUS}_T(p)$ with the leftmost starting position, and let lmMUS_T^p denote the MUS contained in lmSUS_T^p. We say that lmSUS_T^p is the *leftmost SUS* for p, and lmMUS_T^p is the *leftmost MUS* for p. Similarly, we define the *rightmost SUS* rmSUS_T^p and the *rightmost MUS* rmMUS_T^p for p by symmetry. See Fig. 1 for an example for the leftmost/rightmost SUS and MUS.

Let L_T be an array of length n such that $\mathsf{L}_T[i]$ is the length of a SUS[5] of T containing i for each position i with $1 \le i \le n$. Let B_T be a bit array of length n such that $\mathsf{B}_T[i] = 1$ iff i is the beginning position of a meaningful MUS of T.

From the definition of L_T, we yield the following observation:

Observation 1. For every position p with $1 \le p \le n$ and every interval $[x, y] \in \mathsf{SUS}_T(p)$, $p - \mathsf{L}_T[p] + 1 \le x \le p \le y \le p + \mathsf{L}_T[p] - 1$.

Next, we define the following four functions related to L_T and B_T. For a position q with $1 \le q \le n$ let

- $pred1pos_{\mathsf{B}_T}(q) := \max\{i \mid i \le q \text{ and } \mathsf{B}_T[i] = 1\}$,
- $succ1pos_{\mathsf{B}_T}(q) := \min\{i \mid i \ge q \text{ and } \mathsf{B}_T[i] = 1\}$,
- $predneq_{\mathsf{L}_T}(q) := \max\{i \mid i < q \text{ and } \mathsf{L}_T[i] \ne \mathsf{L}_T[q]\}$, and
- $succneq_{\mathsf{L}_T}(q) := \min\{i \mid i > q \text{ and } \mathsf{L}_T[i] \ne \mathsf{L}_T[q]\}$.

For all four functions, we stipulate that $\min\{\} = \max\{\} = nil$. See Fig. 3 for an example of the arrays and functions defined above.

---

[5] Although there can be multiple SUSs containing i, their lengths are all equal.

6.1   Finding SUSs with **L** and **B**

Our idea is to answer point SUS queries with L_T and B_T. For that, we first think about how to find the leftmost and rightmost SUS for a given query (Observation 1 gives us the range in which to search). Having this leftmost and the rightmost SUS, we can find all other SUSs with B_T marking the beginning positions of the meaningful MUSs that correspond to the SUSs we want to output. Before that, we need some properties of L_T that help us to prove the following lemmas in this section: Lemma 7 gives us a hint on the shape of L_T, while Lemma 8 shows us how to find SUSs based on two consecutive values of L_T with a connection to MUSs.

Lemma 7. $|\mathsf{L}_T[p] - \mathsf{L}_T[p+1]| \leq 1$ *for every position p with* $1 \leq p \leq n-1$.

Proof. Let $\ell = \mathsf{L}_T[p]$ and $\ell' = \mathsf{L}_T[p+1]$. From the definition of L_T, there exists a unique substring of length ℓ containing the position p. If $\ell < \ell'$, there is no unique substring of length ℓ containing $p+1$. Thus, $T[p-\ell+1..p]$ is unique, and consequently $T[p-\ell+1..p+1]$ is also unique. Hence, $\ell' = \ell+1$. Similarly, in the case of $\ell > \ell'$, it can be proven that $\ell' = \ell-1$.   □

Lemma 8. *Let p be a position with* $1 \leq p \leq |T|-1$, *and let* $\ell := \mathsf{L}_T[p]$. *If* $\mathsf{L}_T[p+1] = \ell+1$, *then*

 − $T[p-\ell+1..p] \in \mathsf{SUS}_T(p)$,
 − $T[p-\ell+1..p+1] \in \mathsf{SUS}_T(p+1)$, *and*
 − $p-\ell+1$ *is the starting position of a MUS of T*.

If $\mathsf{L}[p+1] = \ell-1$ *then*

 − $T[p..p+\ell-1] \in \mathsf{SUS}_T(p)$,
 − $T[p+1..p+\ell-1] \in \mathsf{SUS}_T(p+1)$, *and*
 − $p+\ell-1$ *is the ending position of a MUS of T*.

Proof. First, we consider the case that $\mathsf{L}_T[p+1] = \ell+1$. From the proof of Lemma 7, $T[p-\ell+1..p]$ and $T[p-\ell+1..p+1]$ are unique substrings in T. Thus, $T[p-\ell+1..p] \in \mathsf{SUS}_T(p)$ and $T[p-\ell+1..p+1] \in \mathsf{SUS}_T(p+1)$. Since every point SUS contains exactly one MUS (cf. Lemma 4), there exists a MUS $[b, e] \subset [p-\ell+1, p]$. Assume that $b > p-\ell+1$, then $T[b..p]$ is the shortest unique substring among all substrings containing the text position p. Its length is $p-b+1 < \ell$. This contradicts that $T[p-\ell+1..p] \in SUS_T(p)$, and therefore $b = p-\ell+1$ must hold. The remaining case $\mathsf{L}_T[p+1] = \ell-1$ can be proven analogously by symmetry.   □

In the following two lemmas (Lemmas 9 and 10), we focus on finding the leftmost SUS and the rightmost SUS for a given query point. That is because the leftmost SUS and the rightmost SUS give us an interval containing the starting positions of the remaining SUSs we want to report[6].

---

[6] The actual reporting of those SUSs is done in Lemma 14
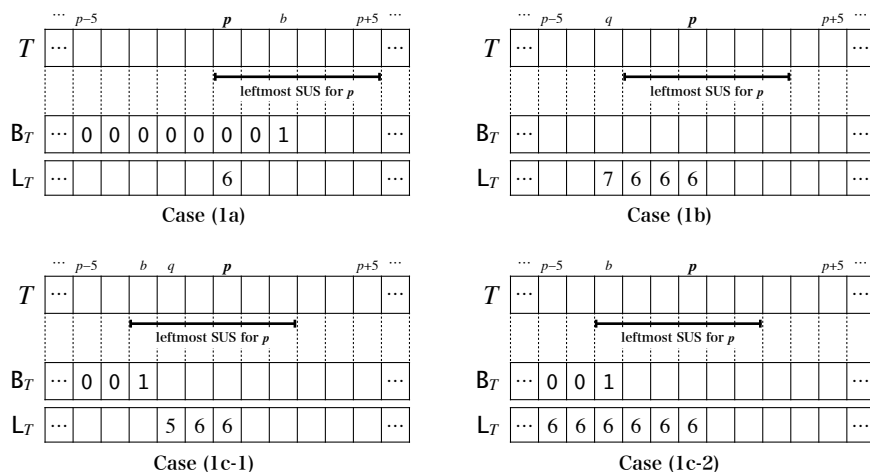
Fig. 4. Example of the proof of Lemma 9 with $\mathsf{L}_T[p] = \ell = 6$. The example (as well as all later examples in this section) still works when replacing the number $\ell = 6$ with another number as long as the relative differences to the other entries in L_T and the search range in T is kept.

Lemma 9. *Let p be a position with $1 \leq p \leq n$, and let $\ell = \mathsf{L}_T[p]$, $q = predneq_{\mathsf{L}_T}(p)$, and $b = succ1pos_{\mathsf{B}_T}(\max\{1, p-\ell+1\})$. Then, $b \leq \min\{p+\ell-1, n\}$ and*

$$\mathsf{lmSUS}_T^p = \begin{cases} [p, p+\ell-1] & \text{if } b \geq p, & (1a) \\ [q+1, q+\ell] & \text{if } b < p \text{ and } q \geq p - \ell + 1 \text{ and } \mathsf{L}_T[q] > \ell, & (1b) \\ [b, b+\ell-1] & \text{otherwise.} & (1c) \end{cases}$$

Proof. If $\ell = 1$, it is clear that the interval $[p, p]$ of length 1 is a MUS of T, thus $b = p$ and $\mathsf{lmSUS}_T^p = [p, p]$. For the rest of the proof, we focus on the case that $\ell \geq 2$. Since $\mathsf{L}_T[p] = \ell$, there exists a unique substring of length ℓ containing the position p, and there exists at least one MUS that is a subinterval of $[p-\ell+1, p+\ell-1]$. Thus, $b \leq \min\{p+\ell-1, n\}$. See Fig. 4 for an illustration of each of the above cases we consider in the following:

(1a) Assume that there exists a unique substring $T[p'..p'+\ell-1]$ containing the position p with $p' < p$. Since $b \geq p > p'$, $T[p'+1..p'+\ell-1]$ is also unique and contains position p. It contradicts $\mathsf{L}_T[p] = \ell$; therefore, $\mathsf{lmSUS}_T^p = [p, p+\ell-1]$.

(1b) From the definition of q and Lemma 7, $\mathsf{L}_T[q] = \ell + 1$ and $\mathsf{L}_T[q+1] = \ell$. From Lemma 8, $[q+1, q+\ell]$ is unique. Also, $[q+1, q+\ell] \in \mathsf{SUS}_T(p)$ because $p \in [q+1, q+\ell]$. Since $\mathsf{L}_T[q] = \ell+1$, there is no unique substring that contains the position q and is shorter than $\ell + 1$. Therefore, $\mathsf{lmSUS}_T^p = [q+1, q+\ell]$.

(1c) We divide this case into two subcases:
  (1c-1) $b < p$ and $q \geq p - \ell + 1$ and $\mathsf{L}_T[q] < \ell$, or
  (1c-2) $b < p$ and $q < p - \ell + 1$.

In Subcase (1c-1), from the definition of q and Lemma 7, $\mathsf{L}_T[q] = \ell - 1$ and $\mathsf{L}_T[i] = \ell$ for all $i \in [q+1, p]$. From Lemma 8, the interval $[q - \ell + 2, q]$ of length $\ell - 1$ is unique. Since $[p - \ell + 1, q] \subset [q - \ell + 2, q]$, $\mathsf{L}_T[i] \leq \ell - 1$ for all $i \in [p - \ell + 1, q]$. In Subcase (1c-2), it is clear that $\mathsf{L}_T[i] = \ell$ for all $i \in [p - \ell + 1, p]$. Therefore, $\mathsf{L}_T[i] \leq \ell$ for all $i \in [p - \ell + 1, p]$ in both subcases. Let e be the ending position of the meaningful MUS $[b, e]$ starting at the position b, and $\ell' = e - b + 1$ be the length of this MUS. We assume $\ell' > \ell$ for the sake of contradiction (and thus $[b, e]$ cannot be lmMUS_T^p whose length is at most ℓ). Since $b \geq p - \ell + 1$ and $\ell' > \ell$, $e > p$ must hold. Let $[b', e'] = \mathsf{lmMUS}_T^p$. Since (a) there is no interval $[x, y] \in \mathsf{SUS}_T(p)$ such that $x < \min\{b, p\}$, and (b) MUSs cannot be nested, we follow that $b' > b$ and $e' > e$. Thus, $\mathsf{lmSUS}_T^p = cover([b', e'], p) = [e' - \ell + 1, e']$ and $\mathsf{L}_T[i] \leq \ell$ for all $p \leq i \leq e'$. Since $[b, e] \subset [p - \ell + 1, e']$, $\mathsf{L}_T[i] \leq \ell$ for all $b \leq i \leq e$. This contradicts that the MUS $[b, e]$ of length $\ell' > \ell$ is a meaningful MUS. Therefore, $\ell' \leq \ell$ and $\mathsf{lmSUS}_T^p = cover([b, e], p) = [b, b + \ell - 1]$.

□

From Lemma 9 we yield the following corollary:

Corollary 1. *If we can compute $\mathsf{L}_T[i]$, $predneq_{\mathsf{L}_T}(i)$ and $succ1pos_{\mathsf{B}_T}(i)$ in constant time for each i with $1 \leq i \leq n$, we can compute lmSUS_T^p in constant time for each position p with $1 \leq p \leq n$.*
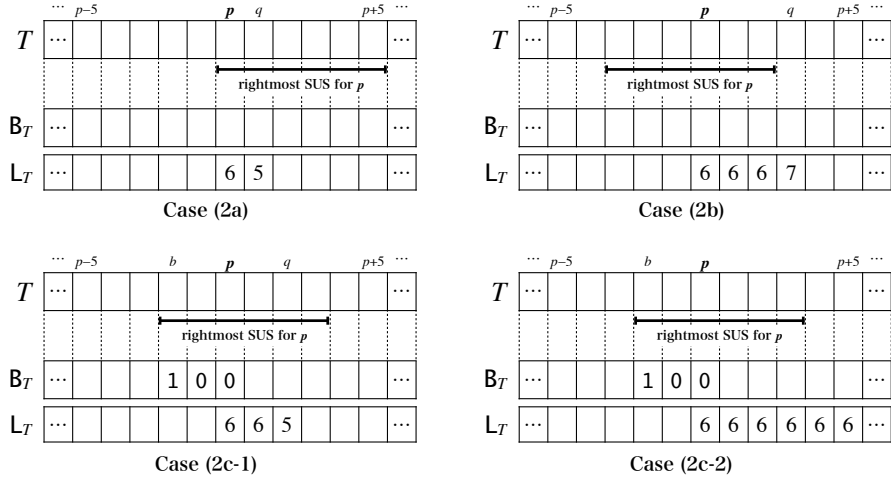

Fig. 5. Example of the proof of Lemma 10 with $\mathsf{L}_T[p] = \ell = 6$.

Lemma 10. *Let p be a position with $1 \leq p \leq n$, and let $\ell = \mathsf{L}_T[p]$, $q = succneq_{\mathsf{L}_T}(p)$, and $b = pred1pos_{\mathsf{B}_T}(p)$. Then,*

$$\mathrm{rmSUS}_T^p = \begin{cases} [p, p + \ell - 1] & \textit{if } q = p + 1 \textit{ and } \mathsf{L}_T[q] < \ell, & \text{(2a)} \\ [q - \ell, q - 1] & \textit{if } q \leq p + \ell - 1 \textit{ and } \mathsf{L}_T[q] > \ell, & \text{(2b)} \\ [b, b + \ell - 1] & \textit{otherwise.} & \text{(2c)} \end{cases}$$

Proof. If $\ell = 1$, it is clear that the interval $[p, p]$ of length 1 is a MUS of T, thus $b = p$ and $\mathrm{rmSUS}_T^p = [p, p]$. We consider the condition of $\ell \geq 2$. See Fig. 5 for an illustration of each of the above cases we consider in the following:

(2a) From Lemma 8, $[p, p + \ell - 1]$ is a SUS for p, which is by definition the rightmost one.

(2b) In this case, $\mathsf{L}_T[q] = \ell + 1$ and $\mathsf{L}_T[q - 1] = \ell$. From Lemma 8, $[q - \ell, q - 1]$ is unique. Since $p \in [q - \ell, q + 1]$, $[q - \ell, q - 1]$ is a SUS for p. Additionally, there is no unique interval $[x, y] \in \mathsf{SUS}_T(p)$ such that $y \geq q$ because $\mathsf{L}_T[q] = \ell + 1$. Thus, $\mathrm{rmSUS}_T^p = [q - \ell, q - 1]$.

(2c) We divide this case into two subcases:

(2c-1) $p + 1 < q \leq p + \ell - 1$ and $\mathsf{L}_T[q] < \ell$, or

(2c-2) $q > p + \ell - 1$.

In Subcase (2c-1), $\mathsf{L}_T[p + 1] = \ell$ and $\mathsf{L}_T[q] = \ell - 1$ and $\mathsf{L}_T[i] = \ell$ for all $p + 2 \leq i \leq q - 1$. From Lemma 8, $[q, q + \ell - 2]$ (of length $\ell - 1$) is unique. Since $[q, p + \ell - 1] \subset [q, q + \ell - 2]$, $\mathsf{L}_T[i] \leq \ell - 1$ for all $q \leq i \leq p + \ell - 1$. In Subcase (2c-2), from the definition of q, $\mathsf{L}_T[i] = \ell$ for all $p \leq i \leq p + \ell - 1$. Therefore, $\mathsf{L}_T[p + 1] = \ell$ and $\mathsf{L}_T[i] \leq \ell$ for all integers i with $p + 2 \leq i \leq p + \ell - 1$ in both subcases.

For the sake of contradiction, assume that there is a MUS $[b', e']$ such that $b' > p$ and $cover([b', e'], p) = [p, e'] \in \mathsf{SUS}_T(p)$. Since $\mathsf{L}_T[p] = \ell$, $[p, e']$ is a unique substring of length ℓ. Hence, $cover([b', e'], p + 1) = [p + 1, e']$ is a unique substring of length $\ell - 1$. It contradicts $\mathsf{L}_T[p + 1] = \ell$; therefore, the beginning position of the rightmost MUS for p is at most p. Next, we show that the MUS starting at b is the rightmost meaningful MUS for p. Let e be its ending position, and $\ell' = e - b + 1$ be its length. We assume that $\ell' > \ell$ for the sake of contradiction (and thus, $[b, e]$ is not rmMUS_T^p whose length is at most ℓ). Since $\mathsf{L}_T[p] = \ell$, $b \geq p - \ell + 1$ and $e > p$. Let $[b'', e''] = \mathrm{rmMUS}_T^p$. Since MUSs cannot be nested, $b'' < b$. Since $e'' - b'' + 1 \leq \ell$, $\mathsf{L}_T[i] \leq \ell$ for all i with $b'' \leq i \leq p + \ell - 1$. We consider two cases to obtain a contradiction:

  − If $e \leq p + \ell - 1$ then it is clear that $\mathsf{L}_T[i] \leq \ell$ for all i with $b \leq i \leq e$. This contradicts that the MUS $[b, e]$ of length ℓ' is a meaningful MUS.

  − If $e > p + \ell - 1$, it is clear that $|cover([b, e], p + \ell - 1)| = |[b, e]| = \ell'$. Since $\mathsf{L}_T[p + \ell - 1] \leq \ell$, there exists a unique substring $[s, t]$ such that $s \leq p + \ell - 1 \leq t$ and $t - s + 1 \leq \ell$. Hence, $\mathsf{L}_T[i] \leq \ell$ for all i with $s \leq i \leq t$. Since $[b, e]$ is a MUS and $p \leq s$, $b < s < e < t$. Consequently, $\mathsf{L}_T[i] \leq \ell$ for all i with $b \leq i \leq e$ and this contradicts that the MUS $[b, e]$ of length ℓ' is a meaningful MUS.

Therefore, $\ell' \leq \ell$ and $\mathrm{rmSUS}_T^p = cover([b, e], p) = [b, b + \ell - 1]$.

$\square$

Corollary 2. *If we can compute* $\mathsf{L}_T[i]$, $succneq_{\mathsf{L}_T}(i)$ *and* $pred1pos_{\mathsf{B}_T}(i)$ *in constant time for each* i *with* $1 \le i \le n$, *we can compute* rmSUS_T^p *in constant time for each position* p *with* $1 \le p \le n$.

6.2   Compact Representations of L

We now propose a succinct representation of the array L_T consisting of the integer array LD_T of length n defined as $\mathsf{LD}_T[1] = 0$ and $\mathsf{LD}_T[i] = \mathsf{L}_T[i] - \mathsf{L}_T[i-1] \in \{-1, 0, 1\}$ for every i with $2 \le i \le n$.

Lemma 11. *The data structure of Theorem 1 can compute* $\mathsf{L}_T[p]$ *in constant time with* $O(\log n)$ *bits of additional working space for each* p *with* $1 \le p \le n$.

Proof. Suppose that we have the data structure D of Theorem 1 and want to know $\mathsf{L}_T[p]$. We query D with the interval $[p, p]$ to retrieve *one* SUS for the query interval $[p, p]$ in constant time. This can be achieved by stopping the retrieval after the first SUS $[i, j] \in \mathsf{SUS}_T([p, p])$ has been reported. Since all SUSs for $[p, p]$ have the same length, $\mathsf{L}_T[p] = j - i + 1$. The additional working space is $O(\log n)$ bits. $\square$

No.	Process	Total working space in bits (excluding MB_T and ME_T)
1	input MB_T, ME_T	-
2	construct RmQ on MUSlen_T	$2m + o(n)$ Lemma 6
3	construct LD_T, $\mathsf{L}_T[1]$	$2n + 2m + o(n)$ Lemma 11
4	free RmQ on MUSlen_T	$2n + o(n)$
5	construct *Huffman-shaped Wavelet Tree* for LD_T	$2n + \lceil n \log_2 3 \rceil + o(n)$ Lemma 12
6	free LD_T	$\lceil n \log_2 3 \rceil + o(n)$
7	construct RMQ on L_T	$2n + \lceil n \log_2 3 \rceil + o(n)$ Lemma 13
8	construct B_T	$3n + \lceil n \log_2 3 \rceil + o(n)$ Lemma 13

Table 1. Working space used during the construction of the data structure proposed in Theorem 2. We can free up space of no longer needed data structures between several steps. See also Fig. 2 for the dependencies of the execution, and other possible ways to build the final data structure. However, these other ways need more maximum working space (at some step) than the way listed in this table.

Lemma 11 allows us to compute LD_T in $O(n)$ time, which we represent as an integer array with bit width two, thus using $2n$ bits of space. In the following, we build a compressed rank/select data structure on LD_T. This data structure is a self-index such that we no longer need to keep LD_T in memory. With LD_T we can access L_T, as can be seen by the following lemma:

Lemma 12. *There exists a data structure of size* $\lceil n \log_2 3 \rceil + o(n)$ *bits that can access* $\mathsf{L}_T[i]$, *and can compute* $predneq_{\mathsf{L}_T}(i)$ *and* $succneq_{\mathsf{L}_T}(i)$ *in constant time for each position* i *with* $1 \le i \le n$. *Given* MB_T *and* ME_T, *the data structure can be constructed in* $O(n)$ *time using* $2n + \max\{\lceil n \log_2 3 \rceil, 2m\} + o(n)$ *bits of total working space, which includes the space for this data structure.*

Proof. The following equations hold for every text position i with $1 \leq i \leq n$:

$$\mathsf{L}_T[i] = \mathsf{L}_T[1] + rank_{\mathsf{LD}_T}(1, i) - rank_{\mathsf{LD}_T}(-1, i),$$
$$predneq_{\mathsf{L}_T}(i) = \max\{select_{\mathsf{LD}_T}(c, rank_{\mathsf{LD}_T}(c, i)) - 1 \mid c \in \{-1, 1\}\},$$
$$succneq_{\mathsf{L}_T}(i) = \min\{select_{\mathsf{LD}_T}(c, rank_{\mathsf{LD}_T}(c, i) + 1) \mid c \in \{-1, 1\}\}.$$

We can compute the value of $\mathsf{L}_T[1]$ and LD_T with Lemma 11. With a rank/select data structure on LD_T we can compute the above functions. Such a data structure is the Huffman-shaped wavelet tree [9]. This data structure can be constructed in linear time and takes $\lceil n \log_2 3 \rceil + o(n)$ bits of space, since the possible number of different values in LD_T is three. Therefore, it can also provide answers to rank/select queries in constant time. □

Finally, we show how to compute B_T:

Lemma 13. *There exists a data structure of size $n + o(n)$ bits that can compute $succ1pos_{\mathsf{B}_T}(i)$ and $pred1pos_{\mathsf{B}_T}(i)$ in constant time for each position $1 \leq i \leq n$. Given MB_T and ME_T, this data structure can be constructed in $O(n)$ time using $3n + \lceil n \log_2 3 \rceil + o(n)$ bits of total working space including the space for this data structure.*

Proof. Our idea is to compute B_T since the following equations hold for every text position i with $1 \leq i \leq n$ (cf. BIT_Y in Section 3.2):

$$pred1pos_{\mathsf{B}_T}(i) = \begin{cases} i & \text{if } \mathsf{B}_T[i] = 1, \\ select_{\mathsf{B}_T}(1, rank_{\mathsf{B}_T}(1, i)) & \text{if } \mathsf{B}_T[i] = 0. \end{cases}$$

$$succ1pos_{\mathsf{B}_T}(i) = \begin{cases} i & \text{if } \mathsf{B}_T[i] = 1, \\ select_{\mathsf{B}_T}(1, rank_{\mathsf{B}_T}(1, i) + 1) & \text{if } \mathsf{B}_T[i] = 0. \end{cases}$$

In the following we show how to compute B_T from MB_T and ME_T in linear time with linear number of bits of working space. Let $b_i = select_{\mathsf{MB}_T}(1, i)$ and $e_i = select_{\mathsf{ME}_T}(1, i)$ be the starting position and the ending position of the i-th MUS respectively, for each $1 \leq i \leq m$. Given $x_i = \mathsf{RMQ}_{\mathsf{L}_T}(b_i, e_i)$, $\mathsf{L}_T[x_i] \leq e_i - b_i + 1$ since $b_i \leq x_i \leq e_i$ and $[b_i, e_i]$ is unique. If $\mathsf{L}_T[x_i] < e_i - b_i + 1$, there is no position p with $cover([b_i, e_i], p) \in \mathsf{SUS}_T(p)$, i.e., $[b_i, e_i]$ is a meaningless MUS. Otherwise $(\mathsf{L}_T[x_i] = e_i - b_i + 1)$, $cover([b_i, e_i], x_i) = [b_i, e_i] \in \mathsf{SUS}_T(x_i)$, i.e., $[b_i, e_i]$ is a meaningful MUS. Hence, it can be detected in constant time whether a MUS is meaningful by an RMQ query on L_T. We can compute the compact representation of L_T described in Lemma 12. The data structure takes $\lceil n \log_2 3 \rceil + o(n)$ bits and can be constructed with $2n + \max\{\lceil n \log_2 3 \rceil, 2m\} + o(n)$ bits of total working space. Subsequently, we endow it with the RMQ data structure of Lemma 3 in $O(n)$ time using $2n + o(n)$ bits of space. Therefore, the computing time of B_T is $O(n)$ and the working space is, asides from the space for MB_T and ME_T, $3n + \lceil n \log_2 3 \rceil + o(n)$ bits, including the space for B_T. Finally, we can endow B_T with rank/select data structures, which allows us to compute each of the above two functions $pred1pos_{\mathsf{B}_T}$ and $succ1pos_{\mathsf{B}_T}$ in constant time. □

Actually, having MB_T and ME_T available, we can simulate an access to $\mathsf{L}_T[i]$ in constant time with Lemma 11 by using the RmQ data structure on MUSlen_T. This allows us to compute the RMQ data structure on L_T directly without the need for computing B_T at first place, i.e., we can replace the working space of Lemma 13 with $2n + 2m + o(n)$ additional bits of working space. However, since our final data structure needs LD_T, computing B_T before LD_T would require more working space in the end than in the other way around, since we no longer need the RmQ data structure on MUSlen_T after having built the rank/select data structure of Lemma 12.

Before stating our final theorem, we need a property for meaningful MUSs:

Lemma 14. *On the one hand, $cover([s_i, e_i], p) \in \mathsf{SUS}_T(p)$ for every meaningful MUS $[s_i, e_i]$ starting with or after the leftmost MUS for p and starting before or with the rightmost MUS for p. On the other hand, each element (i.e., an interval) of $\mathsf{SUS}_T(p)$ starting with or after the leftmost MUS for p and starting before or with the rightmost MUS for p contains exactly one distinct MUS.*

Proof. The first part is shown by Tsuruta et al. [13, Lemma 3]. The second part is due to Lemma 4. □

Theorem 2. *For the point SUS problem, there exists a data structure of size $n + \lceil n \log_2 3 \rceil + o(n)$ bits that can answer a point SUS query in $O(occ)$ time, where occ is the number of SUSs of T for the respective query point. Given MB_T and ME_T, the data structure can be constructed in $O(n)$ time using $3n + \lceil n \log_2 3 \rceil + o(n)$ bits of total working space, which includes the space for this data structure.*

Proof. Let p be a query position, and suppose that the number of SUSs for p is occ. Like the MUSs in Section 4, we rank the SUSs for p by their starting positions. Let $[s_j, e_j]$ be the j-th SUS for p with $1 \le j \le occ$ such that $[s_1, e_1]$ and $[s_{occ}, e_{occ}]$ are the leftmost SUS and the rightmost SUS for p, respectively. If $s_1 = p$ then $[s_1, e_1] = [s_{occ}, e_{occ}]$, and thus the output consists of this single interval. Otherwise ($s_1 \ne p$), we can compute s_i iteratively from s_{i-1} by $s_i = select_{\mathsf{B}_T}(1, rank_{\mathsf{B}_T}(1, s_{i-1}) + 1)$ in constant time for each i with $2 \le i \le occ - 1$, allowing us to answer the query in time linear to the number of SUSs. As occ is not known in advance, we stop the iteration whenever we computed an s_i that is larger than the starting position of the rightmost SUS for p. A detailed analysis of the claimed working space is given in Table 1. □

Corollary 3. *The data structure of Theorem 2 can compute the number of SUSs for a query position in constant time.*

Proof. Let $[s_l, e_l]$ and $[s_r, e_r]$ be the leftmost and the rightmost SUS for a given query position, respectively. All MUSs starting between s_l and s_r (excluding s_l and s_r) are SUSs for this query position. Let occ' be their number. Therefore, the number we want to output is $occ = occ' + 2$. With Lemmas 9 and 10, we can find $[s_l, e_l]$ and $[s_r, e_r]$ in constant time. Further, we can compute occ' in constant time since $occ' = rank_{\mathsf{B}_T}(1, s_r - 1) - rank_{\mathsf{B}_T}(1, s_l)$. □

References

1. Clark, D.R.: Compact Pat Trees. Ph.D. thesis (1998), uMI Order No. GAXNQ-21335
2. Davoodi, P., Raman, R., Satti, S.R.: Succinct representations of binary trees for range minimum queries. In: Proceedings of the 18th Annual International Computing and Combinatorics Conference (COCOON 2012). pp. 396–407 (2012)
3. Ganguly, A., Hon, W.K., Shah, R., Thankachan, S.V.: Space-time trade-offs for finding shortest unique substrings and maximal unique matches. Theoretical Computer Science **700**, 75–88 (2017)
4. Haubold, B., Pierstorff, N., Möller, F., Wiehe, T.: Genome comparison without alignment using shortest unique substrings. BMC Bioinformatics **6**(1), 123 (2005)
5. Hon, W.K., Thankachan, S.V., Xu, B.: An in-place framework for exact and approximate shortest unique substring queries. In: Proceedings of International Symposium on Algorithms and Computation (ISAAC). pp. 755–767 (2015)
6. Hu, X., Pei, J., Tao, Y.: Shortest unique queries on strings. In: Proceedings of String Processing and Information Retrieval (SPIRE). pp. 161–172 (2014)
7. İleri, A.M., Külekci, M.O., Xu, B.: A simple yet time-optimal and linear-space algorithm for shortest unique substring queries. Theoretical Computer Science **562**, 621–633 (2015)
8. Jacobson, G.: Space-efficient static trees and graphs. In: Proceedings of 30th Annual Symposium on Foundations of Computer Science (FOCS). pp. 549–554 (1989)
9. Mäkinen, V., Navarro, G.: Succinct suffix arrays based on run-length encoding. Nordic Journal of Computing **12**(1), 40–66 (2005)
10. Manber, U., Myers, E.W.: Suffix arrays: A new method for on-line string searches. SIAM Journal on Computing **22**(5), 935–948 (1993)
11. Mieno, T., Inenaga, S., Bannai, H., Takeda, M.: Shortest Unique Substring Queries on Run-Length Encoded Strings. In: Proceedings of 41st International Symposium on Mathematical Foundations of Computer Science (MFCS). pp. 69:1–69:11 (2016)
12. Pei, J., Wu, W.C., Yeh, M.: On shortest unique substring queries. In: Proceedings of IEEE 29th International Conference on Data Engineering (ICDE). pp. 937–948 (2013)
13. Tsuruta, K., Inenaga, S., Bannai, H., Takeda, M.: Shortest unique substrings queries in optimal time. In: Proceedings of SOFSEM 2014: Theory and Practice of Computer Science. pp. 503–513 (2014)