Enumerating Range Modes

Kentaro Sumigawa
Graduate School of Information Technology and Science, The University of Tokyo, Japan
kentaro_sumigawa@me2.mist.i.u-tokyo.ac.jp

Sankardeep Chakraborty
National Institute of Informatics, Tokyo, Japan
sankardeep.chakraborty@gmail.com

Kunihiko Sadakane
Graduate School of Information Technology and Science, The University of Tokyo, Japan
sada@mist.i.u-tokyo.ac.jp

Srinivasa Rao Satti
Department of Computer Science and Engineering, Seoul National University, South Korea
ssrao@snu.ac.kr

Abstract

Given a sequence of elements, we consider the problem of indexing the sequence to support range mode queries – given a query range, find the element with maximum frequency in the range. We give indexing data structures for this problem; given a sequence, we construct a data structure that can be used later to process arbitrary queries. Our algorithms are efficient for small maximum frequency cases. We also consider a natural generalization of the problem: the range mode enumeration problem, for which there has been no known efficient algorithms. Our algorithms have query time complexities which are linear in the output size plus small terms.

2012 ACM Subject Classification Theory of computation → Data structures design and analysis

Keywords and phrases range mode, space-efficient data structure, enumeration algorithm

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2020.29

Related Version A full version of the paper can be found at [12], `http://arxiv.org/abs/1907.10984`.

Funding *Sankardeep Chakraborty*: Supported by JSPS KAKENHI Grant Number 18H05291, Japan.
Kunihiko Sadakane: Supported by JST CREST Grant Number JPMJCR1402, Japan.

1 Introduction

We consider the range mode problem, defined as follows.

▶ **Definition 1.1** (Mode). *Given a non-empty multiset S, $x \in S$ is said to be a* mode *of S if its frequency (multiplicity) is no smaller than those of any other elements.*

▶ **Definition 1.2** (Range mode problem). *For a sequence $A[0 \dots n-1]$ and a range $[l, r]$ of A $(0 \le l \le r < n)$, output any one of the modes of the multiset $\{A[l], A[l+1], \dots, A[r-1], A[r]\}$.*

The problem has many applications in data mining and data analysis [2, 4, 15]. Moreover, this problem is of interest to the theory community in general as it is related to the famous Boolean matrix multiplication and set intersection problem [1, 15].

Table 1 Complexities of data structures for the range mode problem where n is the number of terms of a string and m is the maximum frequency of an item. The space complexities do not include one for the input string and are measured in *bits*. The space complexities in the references are measured in *words*, and therefore these are multiplied by $\log n$ in this table.

Data structure	Space complexity (bits)	Query time complexity	conditions
[9, 3]	$\mathrm{O}\left(n^{2-2\epsilon}\log n\right)$	$\mathrm{O}(n^{\epsilon})$	$0 \leq \epsilon \leq 1/2$
[1]	$\mathrm{O}\left(n^{2-2\epsilon}\right)$	$\mathrm{O}(n^{\epsilon})$	$0 \leq \epsilon \leq 1/2$
[10]	$\mathrm{O}\left(\frac{n^2\log\log n}{\log n}\right)$	$\mathrm{O}(1)$	
[6]	$\mathrm{O}(nm\log n)$	$\mathrm{O}(\log m)$	
[3]	$\mathrm{O}((n^{1-\epsilon}m+n)\log n)$	$\mathrm{O}(n^{\epsilon}+\log\log n)$	$0 \leq \epsilon \leq 1/2$
Theorem 3.15	$\mathrm{O}\left(4^k nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}\right)$	$\mathrm{O}(2^k)$	k is any positive integer
Corollary 3.16	$\mathrm{O}\left(nm\left(\log\log\frac{n}{m}\right)^2\right)$	$\mathrm{O}\left(\log\log\frac{n}{m}\right)$	
Theorem 3.19	$\mathrm{O}(nm)$	$\mathrm{O}(\min\{\log m, \log\log n\})$	

In this paper, we consider the indexing version of the range mode problem. That is, given a sequence of length n, we first construct a data structure, called an *index*. Then given a query range $[l, r]$, we solve the query using the index as well as the input. The algorithm is measured by the index size (in bits) and query time complexity. There are many existing work [9, 1, 10, 6, 3][1] and some of them are summarized in Table 1.

Our first contribution is space-efficient indexes for the range mode problem, for the case in which the maximum frequency m of an item in the set is small. Table 1 summarizes our results. The data structure in Corollary 3.16 has better time and space complexities than that of [6] and that of [3] with $\epsilon = 0$, which are also specialized for small m. Also, for $k = O(1)$, the space complexity of Theorem 3.15 is better than that of [10] for all $m \leq n/\log^2 n$ (while the query time is $O(1)$ for both). All the data structures can be constructed in time proportional to the size.

Our second contribution is an efficient index for the range mode enumeration problem, defined as follows.

▶ **Definition 1.3** (Range Mode Enumeration Problem). *Given a sequence* $A[0...n-1]$ *and a query range* $[l, r]$ $(0 \leq l \leq r < n)$*, output* all *modes of the multiset* $\{A[l], A[l+1], \ldots, A[r-1], A[r]\}$.

Though the problem seems to be a natural generalization of the range mode problem, there has been no existing work. A simple modification of an existing algorithm [3] works, but it takes $\mathrm{O}(n^{\epsilon})$ time to output each element of the result (see Theorem 4.4). A related and important problem, the set intersection problem [1], has been considered. However, the set intersection problem can be reduced to the range mode enumeration problem, whereas the converse is not true. We cannot use existing algorithms for the set intersection problem to solve the range mode enumeration problem.

We give faster solutions whose query time complexity is linear in the output size plus some small additive terms. Table 2 summarizes the results. The data structures can be also constructed in time proportional to the size.

[1] The papers [1] and [3] have the same title, but some of the results in [3] do not appear in [1].

Table 2 Complexities of data structures for the range mode enumeration problem where t denotes the number of solutions, n the length of the input sequence, m the maximum frequency of symbols, and ϵ is a parameter between 0 and 1/2 users can choose. Note that the space complexities do not include the space needed to store the input sequence S.

Data structure	Space (bits)	Query time
Theorem 4.4	$O(n^{2-2\epsilon} \log n)$	$O(n^{\epsilon} t)$
Theorem 4.14	$O\left(nm \left(\log\log \frac{n}{m}\right)^2 + n \log n\right)$	$O\left(\log\log \frac{n}{m} + t\right)$
Theorem 4.15	$O(nm + n \log n)$	$O(\log m + t)$
Theorem 4.16	$O(n^{1+\epsilon} \log n + n^{2-\epsilon})$	$O(n^{1-\epsilon} + t)$

The paper is organized as follows. In Section 2, we review basic properties of the range mode problem and existing algorithms for the range mode problem. We also explain fundamental data structures for storing integer sequences. In Section 3, we give our improved algorithms for the range mode problem. In Section 4, we give algorithms for the range mode enumeration problem. Section 5 summarizes the paper.

2 Preliminaries

2.1 Basic properties

We define the following.

- S: input string
- n: the length of string S
- Σ: the set of characters (alphabet) of S
- $f(l, r)$: the frequency of the modes of the substring $S[l, r]$
- m: the frequency of a character with maximum frequency in S, that is, $m = f(0, n-1)$

We assume that Σ is the set of all characters in S. Without loss of generality, we assume that $\Sigma = \{0, 1, \dots, |\Sigma| - 1\}$ and $|\Sigma| \le n$.

▶ **Lemma 2.1** ([7]). *If non-empty multisets M, M_1 and a multiset M_2 satisfies $M = M_1 \cup M_2$ and if x is a mode of M, at least one of the following holds.*

- *x is a mode of M_1.*
- *x belongs to M_2.*

Proof. We prove by contradiction. Let x be a mode of M, and assume x is not a mode of M_1 and $x \notin M_2$. Let $y \in M_1$ be a mode of M_1. From the definition the frequency of y in M_1 is strictly larger than that of x in M_1. Because $x \notin M_2$, the frequency of x in M is equal to that of x in M_1, and it is smaller than that of y in M. This contradicts the assumption that x is a mode of M. ◀

▶ **Lemma 2.2** ([7]). *For indices $l_2 < l_1 \le r_1 < r_2$ of a string S, if $f(l_1, r_1) = f(l_2, r_2)$, modes of range $[l_1, r_1]$ are also modes of range $[l_2, r_2]$.*

Proof. Let c be any mode in range $[l_1, r_1]$ and m be its frequency. Because range $[l_2, r_2]$ contains range $[l_1, r_1]$, the frequency of c in range $[l_2, r_2]$ is at least m. On the other hand, because $f(l_1, r_1) = f(l_2, r_2) = m$, the frequency of c in range $[l_2, r_2]$ is at most m. Therefore the frequency of c in range $[l_2, r_2]$ becomes m and c is also a mode in range $[l_2, r_2]$. ◀

2.2 Algorithms for the range mode problem

We review the data structure with $O(n^{2-2\epsilon})$-word space and $O(n^{\epsilon})$ query time [3]. The input string S of length n is partitioned into $n/s = n^{1-\epsilon}$ blocks of length $s = n^{\epsilon}$ each. In addition to S, the data structure has the following four components.

Two-dimensional array A : For each character in the alphabet, an array for storing positions of its occurrences.

Array B : For each position i of S, $B[i]$ stores the number of times that the character $S[i]$ occurs in the substring $S[0,\dots,i-1]$.

Two-dimensional array C : The (i,j)-th entry of C stores the frequency of modes of the substring from the i-th block to the j-th block. That is, $C[i][j] = f(i\cdot s, (j+1)\cdot s - 1)$.

Two-dimensional array D : The (i,j)-th entry of D stores one of the modes of the substring from the i-th block to the j-th block.

The space complexity is $O(n^{2-2\epsilon})$ words, for any fixed $0 \le \epsilon \le 1/2$. Using these arrays, any query $[l,r]$ is solved in $O(n^{\epsilon})$ time as follows. If a query is contained inside a block, we scan the range $[l,r]$ and for each character in the alphabet, we count its number of occurrences. This takes $O(s) = O(n^{\epsilon})$ time. If a query range $[l,r]$ lies on more than one block, we partition the query range into prefix $[l,(b_l+1)s-1]$, span $[(b_l+1)s, b_r s - 1]$, and suffix $[b_r s, r]$ where $b_l = \lfloor l/s \rfloor, b_r = \lfloor r/s \rfloor$. Note that the span may be empty.

From Lemma 2.1, modes of range $[l,r]$ are either (a) modes of the span, (b) a character in the prefix, or (c) a character in the suffix. For (b) and (c), we scan the prefix and the suffix, and for each character in them, we compute its frequency using the arrays A and B (for details refer to [3]). For (a), one of the modes of the span and its frequency is obtained from $D[b_l][b_r]$ and $C[b_l][b_r]$, respectively. This also takes $O(s) = O(n^{\epsilon})$ time.

There exist improved data structures which are summarized in Table 1.

2.3 Representations of integer sequences

We define *IMS*(n,u) (increasing monotone sequences) and *DMS*(n,u) (decreasing monotone sequences) as follows.

▶ **Definition 2.3.** *We define IMS(n,u) as the set of all integer sequences A of length n such that $0 \le A[0] \le A[1] \le \cdots \le A[n-1] < u$. We also define DMS$(n,u)$ as the set of all integer sequences A of length n such that $u > A[0] \ge A[1] \ge \cdots \ge A[n-1] \ge 0$.*

▶ **Theorem 2.4** ([13]). *Given a sequence $A \in$ IMS(n,u) $(n > u)$ and an integer $k \ge 0$, there exists data structure $Z(n,u)$ using $O(2^k n^{1/2^k} u^{1-1/2^k})$ bits which can answer the queries*

- *access(i,A): return $A[i]$, and*
- *bound(i,A): return $|\{j \mid A[j] > i\}|$*

in $O(2^k)$ time.

▶ **Theorem 2.5** (FID [11]). *For a bit-vector B of length n which contains u ones, consider the following operations.*

- *access(i,B): return the i-th bit of B.*
- *rank$_c(i,B)$: return $|\{j \mid j \le i, B[j] = c\}|$.*
- *select$_c(i,B)$: return $\min\{j \mid \text{rank}_c(j,B) = i\}$.*

Here $c \in \{0,1\}$. There exists a data structure which performs the operations in constant time using $\log \binom{n}{u} + \Theta\left(\frac{n \log\log n}{\log n}\right)$ bits of space.

3 Improved Data Structures for the Range Mode Problem

We propose efficient data structures for the range mode problem using m, the largest frequency of characters, as a parameter.

Consider the data structure of Section 2.2 with $\epsilon = 0$. For simplicity we define $C[i][j] = 1$ for any i, j with $i > j$. Then the $n \times n$ array C satisfies the following property.

▶ **Property 1.** *For any two adjacent entries in the two-dimensional array C, it holds*

$$\begin{aligned} C[i][j] \le C[i][j+1] \le C[i][j]+1 \quad & (0 \le i < n,\ 0 \le j < n-1), \\ C[i][j] \le C[i-1][j] \le C[i][j]+1 \quad & (1 \le i < n,\ 0 \le j < n). \end{aligned}$$

From the definition, C also satisfies:

▶ **Property 2.** *Any entry of C is an integer between* 1 *and m.*

We propose a data structure to store C in compressed form and support constant time access.

3.1 An efficient representation of doubly monotonic arrays

We define the set of two-dimensional arrays which have both column-wise and row-wise monotonicity as follows.

▶ **Definition 3.1.** *We define IMS2(n, m) to be the set of two-dimensional arrays $A[0 \dots n-1][0 \dots n-1]$ which satisfy all the following inequalities.*

$$\begin{aligned} A[i][j] \le A[i][j+1] \quad & (0 \le i < n,\ 0 \le j < n-1), \\ A[i][j] \le A[i+1][j] \quad & (1 \le i < n,\ 0 \le j < n), \\ 0 \le A[i][j] < m \quad & (0 \le i < n,\ 0 \le j < n). \end{aligned}$$

The main result of this subsection is the following.

▶ **Lemma 3.2.** *Let A be a two-dimensional array in IMS2(n, m) $(n \ge m)$ and k be a non-negative integer. There exists a data structure $S_k(n, m)$ which can output an entry of A in $\mathrm{O}\left(2^k\right)$ time using $\mathrm{O}\left(4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}\right)$ bits of space.*

Its proof is given in Section 3.2.

The data structure is recursive. We partition the two-dimensional array A into $u \times u$ blocks where $u = n/t$, each of which has $t = \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k-1}}}$ columns and rows. The block corresponding to $A[it, \dots, (i+1)t-1][jt, \dots, (j+1)t-1]$ is a $t \times t$ two-dimensional array and denoted by $B_{i,j}$. We define flatness of a block as follows.

▶ **Definition 3.3.** *A block is called* flat *if all the entries in the block are identical.*

We also define the height of a block.

▶ **Definition 3.4.** *The* height *of block $B_{i,j}$, denoted by $d_{i,j}$, is defined as*

$$d_{i,j} = B_{i,j}[t-1][t-1] - B_{i,j}[0][0] + 1 \quad (= A[(i+1)t-1][(j+1)t-1] - A[it][jt] + 1).$$

That is, the height of a block is the difference between the maximum and the minimum values in the block, plus one.

We prove the following:

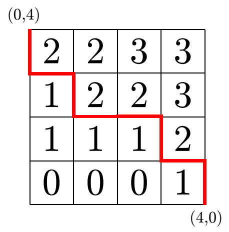


Figure 1 An *IMS2*$(4,4)$ array. The second boundary of its grid graph is shown in a red bold line. We can see that the boundary is a shortest path from vertex $(0,4)$ to vertex $(4,0)$ of the grid graph.

▶ **Lemma 3.5.** *Suppose an $n \times n$ array A in IMS2(n,m) $(n \geq m)$ is partitioned into u^2 blocks of dimension $(n/u) \times (n/u)$ each, for some parameter $u < n$. Then there are at most $2um$ non-flat blocks.*

To prove it, we define the k-th *boundary* in a block for $k = 0, 1, \ldots, m-1$ as follows.

▶ **Definition 3.6.** *For a two-dimensional array $A \in$ IMS2(n,m), consider the $(n+1)\times(n+1)$ grid graph G. The k-th boundary of A is defined as the edge set of G satisfying:*

$$\{((i,j),(i+1,j))|A[i][j-1] < k \text{ and } A[i][j] \geq k\} \\ \cup \{((i,j),(i,j+1))|A[i-1][j] < k \text{ and } A[i][j] \geq k\},$$

where we assume $A[-1][\cdot] = A[\cdot][-1] = -\infty$ and $A[n][\cdot] = A[\cdot][n] = \infty$.

See Figure 1 for an example.

Then the following holds.

▶ **Property 3.** *The k-th boundary is a shortest path from vertex $(0,n)$ to vertex $(n,0)$ of the grid graph G. That is, if we regard the path as a directed path from $(0,n)$ to $(n,0)$, the edges in the path are of the form of either $(i,j) \to (i+1,j)$ or $(i,j) \to (i,j-1)$.*

Proof of Lemma 3.5. A block is flat if and only if no boundary passes through the block. For each of the m boundaries, the number of blocks through which the boundary passes is $2u$. Therefore the number of blocks containing at least one boundary in it is at most $2um$. ◀

Based on Property 3, we use the following data structures for $S_k(n,m)$.

E: to store *IMS2*(u,m)

We define $E[i][j] = A[it][jt]$ $(0 \leq i < u, 0 \leq j < u)$. It is clear that $E \in$ *IMS2*(u,m).

$F_{i,j}$: to store differences inside block $B_{i,j}$

For non-flat block $B_{i,j}$, we define $F_{i,j}[x][y] = B_{i,j}[x][y] - B_{i,j}[0][0]$. Then it holds $F_{i,j} \in$ *IMS2*$(t, d_{i,j})$.

First we show how to compute an entry of A using the above data structures. For the original array A, $A[i][j] = E[i/t][j/t] + F_{i/t,j/t}[i\%t][j\%t]$, where / and % denotes integer division and modulo respectively, and for flat block $B_{i,j}$, the two-dimensional array $F_{i,j}$ is the zero-value array. It is necessary to decide if a block is flat or not. Because a naive data structure using a $u \times u$ Boolean array is space-consuming, we develop a space-efficient solution. To do so, we define the following mapping.

▶ **Definition 3.7** (Mapping to decide if a block is flat or not). *We define a mapping from a block number to a pair of integers $\Phi : \{0,\ldots,u-1\}^2 \to \{0,\ldots,m-1\} \times \{0,\ldots,2u-2\}$ as*

$$(i,j) \mapsto (E[i][j], i-j+u-1).$$

We obtain the following.

▶ **Lemma 3.8.** *$\Phi(i_1, j_1) \neq \Phi(i_2, j_2)$ for any two distinct non-flat blocks B_{i_1,j_1} and B_{i_2,j_2}.*

Proof. Let B_{i_1,j_1} and B_{i_2,j_2} be two distinct non-flat blocks. From the definition of Φ the claim holds if $E[i_1][j_1] \neq E[i_2][j_2]$. If $E[i_1][j_1] = E[i_2][j_2]$, the $E[i_2][j_2]$-th boundary must pass both B_{i_1,j_1} and B_{i_2,j_2}. It is however not possible to pass both of them if $i_1 - j_1 = i_2 - j_2$ from Property 3. Thus it holds $\Phi(i_1, j_1) \neq \Phi(i_2, j_2)$. ◀

We also define a mapping, which is something like an inverse of Φ.

▶ **Definition 3.9.** *We define a mapping $\Psi : \{0, \ldots, m-1\} \times \{0, \ldots, 2u-2\} \to \{0, \ldots, u-1\}^2 \cup \{\perp\}$ as $\Psi(x, y) = (i, j)$ if there exists a non-flat block $B_{i,j}$ with $\Phi(i, j) = (x, y)$, and $\Psi(x, y) = \perp$ otherwise.*

Then it is easy to see that block $b_{i,j}$ is non-flat if and only if $\Psi(\Phi(i, j)) = (i, j)$. To use this fact, we have to compute both Ψ and Φ. We can compute Φ from Definition 3.7. To compute Ψ, we use the following.

▶ **Proposition 3.10.** *Assume that $\Psi(x, y_1) = (i_1, j_1)$ and $\Psi(x, y_2) = (i_2, j_2)$. Then $y_1 \leq y_2$ implies $i_1 \leq i_2$ and $j_1 \geq j_2$.*

Finally we obtain the following:

▶ **Lemma 3.11.** *$\Psi(x, y)$ is computed in constant time using a data structure of size $(2\sqrt{2}c^{1/3} + 2)\dfrac{nm}{t}$ bits.*

Proof. We use a two-dimensional Boolean array K of $2um$ bits storing for each member (x, y) of $\{0, \ldots, m-1\} \times \{0, \ldots, 2u-2\}$, True if $\Psi(x, y) \neq \perp$ and False if $\Psi(x, y) = \perp$. In addition to this, for each x we create two integer sequences I_x and J_x of length $2u - 1$ each, as follows. For each $y \in \{0, \ldots, 2u-2\}$, we define $(I_x[y], J_x[y]) = \Psi(x, y)$ if $\Psi(x, y) \neq \perp$. If $\Psi(x, y) = \perp$, we choose arbitrary values for $I_x[y]$ and $J_x[y]$ satisfying $I_x[y] \in \mathit{IMS}(2u-1, u), J_x[y] \in \mathit{DMS}(2u-1, u)$. From Proposition 3.10, such sequences I_x and J_x must exist. By using the data structure $Z(2u-1, u)$ of Theorem 2.4, we can store each sequence in at most $\sqrt{2}c^{1/3}u$ bits and access in constant time. The total space for these $2m$ sequences is at most $2um + 2\sqrt{2}c^{1/3}um = (2\sqrt{2}c^{1/3} + 2)\dfrac{nm}{t}$ bits. ◀

3.2 Proof of Lemma 3.2

We first analyze the size of the data structure S_k for $k > 0$. We prove by induction on k that there exists a data structure $S_k(n, m)$ using at most $c4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2k}}}$ bits of space, where $c \geq 1$ is some constant satisfying:

- There exists data structure $Z(n, m)$ using at most $c^{1/3}\sqrt{nm}$ bits of space which can read an entry of $\mathit{IMS}(n, m)$ in constant time.

Next, we show such a constant c exists, if we use the data structure of Theorem 2.4. For $k = 0$, we use the data structure $Z(n, m)$ of Theorem 2.4 for storing each column. Then the space usage is at most $c^{1/3}n^{3/2}m^{1/2}$ bits, which is at most $cn^{3/2}m^{1/2}$ bits and the claim holds. For $k > 0$, the size of the data structure S_k satisfies the following lemma.

▶ **Lemma 3.12.** *The size of the data structure S_k is $c4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2k}}}$ for $k > 0$.*

Proof. If $u > m$, the space complexity of the two-dimensional array E is, from the assumption of $S_{k-1}(u,m)$,

$$c4^{k-1}um\left(\frac{u}{m}\right)^{\frac{1}{2^{2^{k-1}}}} = c4^{k-1}nm\left(\frac{m}{n}\right)^{\frac{1}{2^{2^{k-1}}}}\left(\frac{u}{m}\right)^{\frac{1}{2^{2^{k-1}}}} = c4^{k-1}nm\left(\frac{1}{t}\right)^{\frac{1}{2^{2^{k-1}}}}$$
$$= c4^{k-1}nm\left(\frac{m}{n}\right)^{\frac{1}{2^{2^{k-1}}}\frac{1}{2^{2^{k-1}}}} \le c4^{k-1}nm.$$

If $u \le m$, it can be stored in $c^{1/3}u^{3/2}m^{1/2}$ bits by using the data structure $Z(u,m)$ for each row. Therefore for any case E can be stored in at most $c4^{k-1}nm$ bits.

Next we consider the space complexity of storing differences inside non-flat blocks. For the summation of all $d_{i,j}$, it holds $\sum_{\substack{0\le i<u \\ 0\le j<u}} d_{i,j} \le \frac{2mn}{t}$ because from the column-wise and row-wise monotonicity, for each $l = -u+1, \dots, u-1$, it holds $\sum_{i-j=l} d_{i,j} \le m$.

Consider the space complexity of the data structure storing $F_{i,j} \in \mathit{IMS2}(t, d_{i,j})$ for non-flat blocks $B_{i,j}$. If $t > d_{i,j}$, by using $S_{k-1}(t, d_{i,j})$, the space becomes $c4^{k-1}td_{i,j}\left(\frac{t}{d_{i,j}}\right)^{\frac{1}{2^{(2^{k-1})}}}$ bits. If $t \le d_{i,j}$, we store each row of the two-dimensional array in $t \cdot c^{1/3}\sqrt{td_{i,j}}$ bits by using $Z(t, d_{i,j})$ which support constant access. For both time and space, the former case has worse complexities. Therefore we analyze the space by assuming every block is stored in $c4^{k-1}td_{i,j}\left(\frac{t}{d_{i,j}}\right)^{\frac{1}{2^{(2^{k-1})}}}$ bits.

$$\sum_{\substack{i,j \\ B_{i,j}\text{ not flat}}} c4^{k-1}td_{i,j}\left(\frac{t}{d_{i,j}}\right)^{\frac{1}{2^{2^{k-1}}}} \le \sum_{i,j} c4^{k-1}td_{i,j}\left(\frac{t}{d_{i,j}}\right)^{\frac{1}{2^{2^{k-1}}}} \le \sum_{i,j} c4^{k-1}td_{i,j}t^{\frac{1}{2^{2^{k-1}}}}$$
$$\le c4^{k-1}t^{1+\frac{1}{2^{2^{k-1}}}}\sum_{i,j} d_{i,j} \le c4^{k-1}t^{1+\frac{1}{2^{2^{k-1}}}} \cdot \frac{2nm}{t}$$
$$= 2c\cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^{k-1}}}\frac{1}{2^{2^{k-1}}}} = 2c\cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}.$$

We also need to store pointers to the data structures $F_{i,j}$ because their size varies depending on (i,j). As a bijection between $\{0,\dots,m-1\}\times\{0,\dots,2u-2\}$ and $\{0,1,\dots,m(2u-1)\}$, we define $(i,j)\mapsto i(2u-1)+j$. By using this, we can regard the pointers to the data structures as a monotone increasing sequence P with $2um$ terms and range $2c\cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}$. By representing P by the data structure $Z(2um, 2c\cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}})$, it holds

$$c^{1/3}\sqrt{2um\cdot c\cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}} \le c^{5/6}2^k nm.$$

Therefore the space is upper-bounded by $c^{5/6}2^k nm$ bits.

The total space of the data structures for S_k is:

$$\underbrace{c4^{k-1}nm}_{\text{array } E} + \underbrace{(c^{1/3}\cdot 2\sqrt{2}+2)\frac{nm}{t}}_{\Psi} + \underbrace{2c\cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}}_{\text{total space for } F} + \underbrace{c^{5/6}2^k nm}_{P} \text{ bits.}$$

By letting $c \geq 10^6$, for any positive integer k, it holds

$$
\begin{aligned}
& c4^{k-1}nm + (2\sqrt{2}c^{1/3} + 2)\frac{nm}{t} + 2c \cdot 4^{k-1}nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}} + c^{5/6}2^k nm \\
\leq\ & (c4^{k-1} + c^{1/3}2\sqrt{2} + 2 + 2c \cdot 4^{k-1} + c^{5/6}2^k)nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}} \\
\leq\ & c4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}} .
\end{aligned}
$$

This proves there exists a data structure of $c4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}$ bits for $S_k(n, m)$. ◀

Next we consider the time T_k to access an entry of S_k. It holds $T_k = 2T_{k-1} + \mathrm{O}(1)$ and we obtain $T_k = \mathrm{O}(2^k)$.

This completes the proof of Lemma 3.2. By setting $k = \log\log\log \frac{n}{m}$ in Lemma 3.2, we obtain:

▶ **Corollary 3.13.** *There exists a data structure of* $\mathrm{O}\left(nm \left(\log\log \frac{n}{m}\right)^2\right)$ *bits supporting an access to IMS2*(n, m) $(n > m)$ *in* $\mathrm{O}\left(\log\log \frac{n}{m}\right)$ *time.*

We add an auxiliary data structure:

▶ **Lemma 3.14.** *Given an* $n \times n$ *array* $A \in$ *IMS2*(n, m)*, there exists a data structure of* $\mathrm{O}(nm)$ *bits, such that given a column number* r *of* $A \in$ *IMS2*(n, m) *and a value* h*, one can compute* $\min\{x \mid A[r][x] \geq h\}$ *in constant time.*

Proof. For the array A, consider the boundaries of Definition 3.6. Note that $\min\{x \mid A[r][x] \geq h\}$ is the minimum row number of elements in r-th column which are above the h-th boundary. Recall that boundaries are shortest paths in the grid graph from vertex $(0, n)$ to vertex $(n, 0)$. We give a bit-vector representation of a boundary as follows. Initially the bit-vector is set empty. We traverse the graph from vertex $(0, n)$ to vertex $(n, 0)$ along the boundary, and append 0 when we go down, and 1 when we go right, to the end of the bit-vector. Thus the bit-vector for a boundary has length $2n$, with n zeros and n ones in it. Hence we need $O(mn)$ bits to store the bit-vectors for all the m boundaries. Let B_k denote the bit-vector for the k-th boundary. From definition, it holds $\min\{x \mid A[r][x] \geq h\} = n - \mathsf{rank}_0(\mathsf{select}_1(r, B_h), B_h)$. This can be computed in constant time by using FID (Theorem 2.5). ◀

3.3 An efficient representation of the array C

By using the data structure of Lemma 3.2 for a two-dimensional array C satisfying Property 1, we can compute the frequency $c = f(l, r)$ of the modes of a query range $[l, r]$.

▶ **Theorem 3.15.** *In addition to the string* S*, by using a data structure of* $\mathrm{O}\left(4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}\right)$ *bits, we can solve the range mode problem in* $\mathrm{O}\left(2^k\right)$ *time.*

Proof. We can use Algorithm 1, where the two-dimensional array C satisfies the following:

$$
C[i][j] = \begin{cases} f(i, j) & (i \leq j), \\ 1 & (\text{otherwise}). \end{cases}
$$

By permuting the columns of C in reverse order and subtracting one from all values, C belongs to *IMS2*(n, m).

A mode is obtained by computing $S[\min\{x > l \mid C[l][x] = c\}]$ where $c = f(l,r)$ because $C[l][x-1] = c-1$ and $S[x]$ is a mode. To compute this, we use the data structure of Lemma 3.14. In Algorithm 1, the data structures of Lemma 3.2 and Lemma 3.14 are used. The space complexity includes $\mathrm{O}\left(4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}\right)$ bits for Lemma 3.2 and $\mathrm{O}(nm)$ bits for Lemma 3.14, and therefore the total space complexity is $\mathrm{O}\left(4^k nm \left(\frac{n}{m}\right)^{\frac{1}{2^{2^k}}}\right)$ bits. ◀

Algorithm 1 Algorithm for Theorem 3.15.

```
Require: a query range $[l, r]$
Ensure: a mode in range $[l, r]$ of string $S$
1: $f \leftarrow C[l][r]$                              ▷ using Lemma 3.2
2: if $f = 1$ then
3:     $i \leftarrow l$
4: else
5:     $i \leftarrow \min\{x \mid C[l][x] = f\}$       ▷ using Lemma 3.14
6: end if
7: return $S[i]$
```

Algorithm 2 Algorithm for finding the mode index set.

```
Require: a query range $[l, r]$
Ensure: the mode index set $ans$
1: $g \leftarrow f(l, r)$
2: $b \leftarrow \min\{t \mid f(l, t) \geq g\}$
3: $x \leftarrow r$
4: $ans \leftarrow \{\}$
5: while $x \geq b$ do
6:     $ans \leftarrow ans \cup \{x\}$                  ▷ add to the mode index set
7:     $x \leftarrow \mathsf{select}_1(\mathsf{rank}_1(x-1, B[l]), B[l])$   ▷ update $x$
8: end while
9: return $ans$
```

By letting $k = \log\log\log \frac{n}{m}$ in Theorem 3.15, we obtain:

▶ **Corollary 3.16.** *In addition to the string S, using a data structure of $\mathrm{O}\left(nm \left(\log\log \frac{n}{m}\right)^2\right)$ bits, we can answer range mode queries in $\mathrm{O}\left(\log\log \frac{n}{m}\right)$ time.*

This data structure is superior to the data structure of [3] with $\epsilon = 0$, which has space complexity $\mathrm{O}(nm \log n)$ bits and query time complexity $\mathrm{O}(\log\log n)$, in terms of both time and space.

3.4 Efficient data structure for small m

Instead of using the two-dimensional array C storing frequencies of all the ranges, we can compute the frequency of modes using only the bit-vector representation of the boundaries of Lemma 3.14.

▶ **Lemma 3.17.** *For a two-dimensional array $A \in IMS2(n, m)$, there exists a data structure of $\mathrm{O}(nm)$ bits to decide if $A[i][j] \geq k$ in constant time.*

Proof. We store all the bit-vectors $B_0, \dots, B_{m-1}$ representing the boundaries. It is enough to decide if the k-th boundary of A is either in the $(0, 0)$'s side or (n, n)'s side with respect to (i, j). This is done by Algorithm 3, which runs in $O(1)$ time. This is done in $O(1)$ time. ◀

Algorithm 3 A function to compare $A[i][j]$ with k.

```
Require: an index $(i, j)$ of $A$ and an index $k$ of a boundary
Ensure: if $A[i][j] \geq k$ or not
1: if $n - \mathsf{rank}_0(\mathsf{select}_1(i, B_k), B_k) \leq j$ then
2:     return YES
3: else
4:     return NO
5: end if
```

From Lemma 3.17, we can compute $C[i][j] = \max\{k | C[i][j] \geq k\}$ in $\mathrm{O}(\log m)$ time by a binary search on k.

▶ **Lemma 3.18.** *There exists a data structure of $\mathrm{O}(nm)$ bits that for a given (r, c) of $A \in \mathit{IMS2}(n, m)$, computes the value h such that $h - \log n < A[r][c] \leq h$ in $O(\log \log n)$ time.*

Proof. For every column r of A, we take the set of (at most $m/\log n$) row indexes c such that $A[r][c] > A[r][c-1]$ and $A[r][c]$ is a multiple of $\log n$, and store the set using a predecessor data structure [14]. The space usage for each column is $O((m/\log n)\log n) = O(m)$ bits, and hence overall $O(nm)$ bits to represent A. The query is supported by finding the predecessor of c in the predecessor data structure corresponding to the column r. ◀

Now using the structure of Lemma 3.17, we can compute $C[i][j] = \max\{k | C[i][j] \geq k\}$ in $\mathrm{O}(\log \log n)$ time by a binary search on k, after narrowing down the length of the range of k to $O(\log n)$ using the structure of Lemma 3.18. Furthermore, from Lemma 3.14, we can compute an index for modes in constant time. Now we obtain the following theorem.

▶ **Theorem 3.19.** *In addition to the input string S, by using a data structure of $\mathrm{O}(nm)$ bits, the range mode problem is solved in $\mathrm{O}(\min\{\log m, \log \log n\})$ time.*

Table 1 summarizes the proposed and known data structures.

4 Range Mode Enumeration Problem

Below we consider range modes of a string S with alphabet size σ instead of a sequence A. We evaluate algorithms with their space complexity and query time complexity using the size of the output t as a parameter.

4.1 Algorithms using existing data structures

Data structures for the range mode problem return an arbitrary item among all range modes. Instead here we consider a data structure for the problem which returns the leftmost index and the frequency of range modes, where the leftmost index is defined as follows.

▶ **Definition 4.1.** *For a string S and query range $[l, r]$, the leftmost index of range modes is defined as $\min\{x \mid S[x]$ is an item with the largest frequency in the query range $[l, r]\}$.*

▶ **Lemma 4.2.** *Let D be a data structure which returns the leftmost index of range modes for a query range $[l, r]$ in time T using s space, there exists a data structure which solves the range mode enumeration problem in time $(T + O(1))t$ using s space.*

Proof. We solve the problem using the data structure D. The algorithm narrows the query range gradually. Because the data structure D returns the leftmost index i of range modes, the number of range modes for the new query range $[i + 1, r]$ is exactly one smaller than that of the current query range. Therefore the total time complexity is $(T + \mathrm{O}(1))t$. ◀

▶ **Lemma 4.3.** *There exists a data structure for finding the leftmost index of range modes and their frequency in time $\mathrm{O}(n^{\epsilon})$ using a data structure with space complexity $\mathrm{O}(n^{2-2\epsilon})$ words in addition to the input string S.*

Proof. We slightly change the data structure of [3] described in Section 2.2. Instead of the two-dimensional array D storing modes of block ranges, we create another two-dimensional array D' storing leftmost indices of block ranges. Then we can find the leftmost index of span in constant time. For items appearing in the prefix and the suffix, we can find the leftmost index and its frequency using the same algorithm. Algorithm 4 gives a pseudo code. ◀

Algorithm 4 Find the leftmost index of range modes (assuming l, r belong to different blocks).

```
Require: a query range $[l, r]$ $(b_l := \lfloor l/n^{\epsilon} \rfloor \neq b_r := \lfloor r/n^{\epsilon} \rfloor)$
Ensure: (leftmost index$li$, frequency$f$)
 1: $f \leftarrow C[b_l][b_r]$                          ▷ obtain the frequency of modes of span
 2: $li \leftarrow D'[b_l][b_r]$
 3: for $i = l, \dots, (b_l + 1)s - 1$ do                ▷ check symbols in the prefix
 4:     $cnt \leftarrow 0$
 5:     while (the number of terms of$A[S[i]]$) $> B[i] + f - 1$ and $A[S[i]][B[i] + f - 1] \leq r$ do
 6:         $li \leftarrow \min(li, i)$
 7:         $f \leftarrow f + 1, cnt \leftarrow cnt + 1$
 8:     end while
 9:     if $cnt > 0$ then
10:         $f \leftarrow f - 1$
11:     end if
12: end for
13: for $i = b_r s, \dots, r$ do                          ▷ check symbols in the suffix
14:     $cnt \leftarrow 0$
15:     while $0 \leq B[i] - freq + 1$ and $A[S[i]][B[i] - freq + 1] \geq l$ do
16:         $f \leftarrow f + 1, cnt \leftarrow cnt + 1$
17:         $li \leftarrow \min(li, A[S[i]][B[i] - freq + 1])$
18:     end while
19:     if $cnt > 0$ then
20:         $f \leftarrow f - 1$
21:     end if
22: end for
23: return $(li, f)$
```

From Lemmas 4.2 and 4.3, we obtain the following.

▶ **Theorem 4.4.** *There exists a data structure for the range mode enumeration problem solving a query in $\mathrm{O}(n^{\epsilon}t)$ time using $\mathrm{O}(n^{2-2\epsilon} \log n)$ bits.*

4.2 More efficient data structures for enumeration

▶ **Definition 4.5.** *The* mode index set *for a query range $[l,r]$ of the range mode enumeration problem is the set of the leftmost position of each mode in the query range.*

Because the set of all range modes can be obtained from the mode index set, below we focus on finding the mode index set.

Define n bit-vectors $B[0],\ldots,B[n-1]$ of length n each as $B[i][j]=1 \Leftrightarrow i \le j$ and $S[j]$ is a mode of range $[i,j]$. Using these bit-vectors, we obtain:

▶ **Lemma 4.6.** *The set of modes for a query range $[l,r]$ is $\{x \mid f(l,x)=f(l,r) \wedge B[l][x]=1\}$.*

Proof. From the definition of $B[l][x]$, $S[x]$ is a mode of range $[l,x]$. From Lemma 2.2, $S[x]$ is also a mode of range $[l,r]$. Conversely, for any index x contained in the index set for range $[l,r]$, it holds $f(l,x)=f(l,r)$ and $B[l][x]=1$. Therefore these two sets coincide. ◀

Therefore we can enumerate items in the mode index set.

We now analyse the time and space complexities of the algorithm. For the data structure B, which consists of n bit-vectors of length n, we use $\mathrm{O}(n^2)$ bits. We also use $\mathrm{O}(n^2)$ bits for the array C storing frequencies using bit-vectors, which is used to obtain frequency of modes of a query range. Therefore the total space is $\mathrm{O}(n^2)$ bits. The total time complexity is $\mathrm{O}(t)$. We obtain the following basic data structure.

▶ **Lemma 4.7.** *There exists a data structure for the range mode enumeration problem which computes the mode index set in $\mathrm{O}(t)$ time using a data structure of $\mathrm{O}(n^2)$ bit space in addition to the input string S.*

Now we improve the space using a parameter m, the frequency of modes of the entire range. The following lemma holds for the two-dimensional bit-array B.

▶ **Lemma 4.8.** *There is the following relation between function f and bit-array B.*

If $f(i,j)=f(i+1,j)$ and $B[i+1][j]=1$, then $B[i][j]=1$.

Proof. Because $B[i+1][j]=1$, $S[j]$ is a mode of range $[i+1,j]$. Using Lemma 2.2, it holds $S[j]$ is also a mode of range $[i,j]$. From the definition of B, we obtain $B[i][j]=1$. ◀

Using this property, we define m integer sequences $H[1],\ldots,H[m]$ of length n each.

▶ **Definition 4.9.** *Define integer sequences $H[1],\ldots,H[m]$ as follows*[2].

$$H[i][j]=\max\{\{k \mid f(k,j)=i \text{ and } B[k][j]=1\} \cup \{-1\}\} \quad (1 \le i \le m, 1 \le j \le n).$$

That is, $H[i][j]$ is the index k of the shortest range $[k,j]$ such that the frequency of the modes in $[k,j]$ is i, and $S[j]$ is a mode.

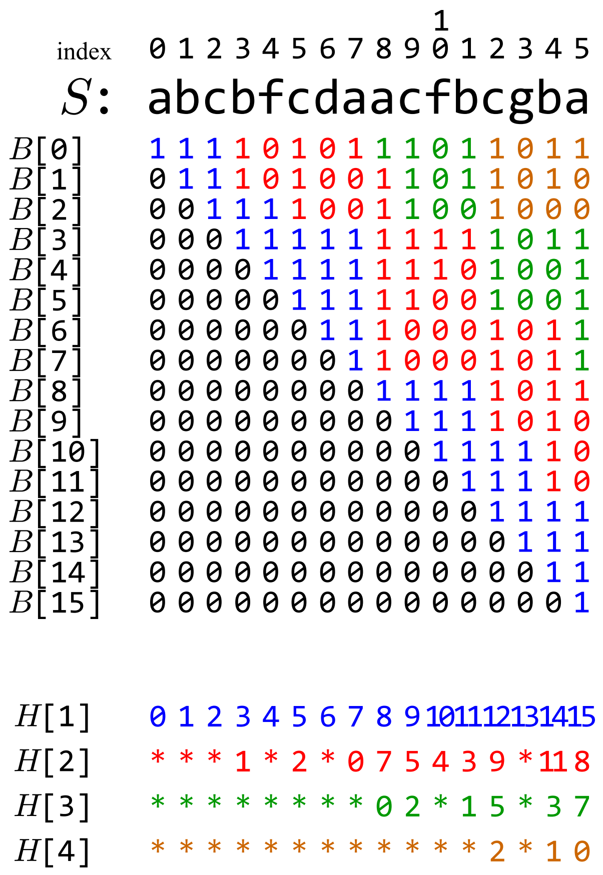
The bit-array B and the sequences H have the following relation.

▶ **Lemma 4.10.** *$B[i][j]=1 \Leftrightarrow H[f(i,j)][j] \ge i$.*

Figure 2 shows an example of bit-array B and sequences H for string $S=$ "abcbfcdaacfbcgba".

Algorithm 2 enumerates indices with bits being set in range $[b,r]$ of bit-vector $B[l]$. Here for any t with $b \le t \le r$, the value of $f(l,t)$ is always $g=f(l,r)$. Therefore this operation is identical to enumerate all indices in range $[b,r]$ of sequence $H[g]$ whose value is at least l. This problem can be regarded as the range maximum problem.

[2] $H[i][j]$ denotes the j-th entry of sequence $H[i]$



index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S:	a	b	c	b	f	c	d	a	a	c	f	b	c	g	b	a
B[0]	1	1	1	1	0	1	0	1	1	1	0	1	1	0	1	1
B[1]	0	1	1	1	0	1	0	0	1	1	0	1	1	0	1	0
B[2]	0	0	1	1	1	1	0	0	1	1	0	0	1	0	0	0
B[3]	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1
B[4]	0	0	0	0	1	1	1	1	1	1	1	0	1	0	0	1
B[5]	0	0	0	0	0	1	1	1	1	1	0	0	1	0	0	1
B[6]	0	0	0	0	0	0	1	1	1	0	0	0	1	0	1	1
B[7]	0	0	0	0	0	0	0	1	1	0	0	0	1	0	1	1
B[8]	0	0	0	0	0	0	0	0	1	1	1	1	1	0	1	1
B[9]	0	0	0	0	0	0	0	0	0	1	1	1	1	0	1	0
B[10]	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	0
B[11]	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	0
B[12]	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1
B[13]	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
B[14]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1
B[15]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
H[1]	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
H[2]	*	*	*	1	*	2	*	0	7	5	4	3	9	*	11	8
H[3]	*	*	*	*	*	*	*	*	0	2	*	1	5	*	3	7
H[4]	*	*	*	*	*	*	*	*	*	*	*	*	2	*	1	0

Figure 2 Bit-vectors $B[0], \ldots, B[15]$ and sequences $H[1], \ldots, H[4]$ for string S of length $n = 16$. The marks * stand for -1. Colors of numbers for B represent frequencies of modes of the corresponding ranges. Blue, red, green, and brown colors represent frequencies 1, 2, 3, and 4, respectively.

▶ **Definition 4.11** (Range Maximum Problem (RMQ)). *Given a sequence A and query range $[l, r]$, the range maximum problem asks an index of the maximum value in the sub-sequence $A[l, \ldots, r]$.*

For range maximum queries, there is an efficient data structure.

▶ **Theorem 4.12** ([5]). *For the range maximum problem of size n, there exists a data structure with space complexity $2n + \mathrm{o}(n)$ bits and query time complexity $\mathrm{O}(1)$.*

▶ **Theorem 4.13** ([8]). *Consider the following problem: Given a sequence A, a query range $[l, r]$ and a threshold t, compute $\{l \leq k \leq r \mid A[k] \geq t\}$. If there exists an oracle to check if $A[k] \geq t$ for some k in constant time, then the query is answered in time proportional to the output size using the range maximum data structure for A.*

Consider a data structure to decide if $H[f(l, r)][k] \geq l$ or not for finding the index set. This can be done by using the arrays A, B in Section 2.2 because it is equivalent that $H[f(l, r) = g][k] \geq l$ and the frequency of $S[k]$ in range $[l, r]$ is at least g.

Now we obtain the main theorems.

▶ **Theorem 4.14.** *There exists a data structure with space complexity* $\mathrm{O}\left(nm\left(\log\log\frac{n}{m}\right)^2+n\log n\right)$ *bits in addition to the input string* S*, which answers a range mode enumeration query in time* $\mathrm{O}\left(\log\log\frac{n}{m}+t\right)$ *where* t *is the number of modes.*

Proof. It is enough to use the following data structures to enumerate the solutions.

Two-dimensional array A storing positions of occurrences of symbols
An array to store positions of occurrences in ascending order for each symbol of the alphabet

Array B to store ranks for strings
An array storing the rank for each index of S, that is, $B[i]$ stores the number of times that the symbol $S[i]$ appears in the substring $S[0,\dots,i-1]$.

Two-dimensional array C storing frequencies of modes for all ranges
The (i,j) entry of the array C stores the frequency of the modes for range $[i,j]$.

m bit-vectors D storing boundaries of the array C
The array stores m bit-vectors of Theorem 3.14.

Two-dimensional array H storing m RMQ data structures for arrays of length n each
An array storing m sequences of Definition 4.9 as RMQ data structures. The sequences themselves are not stored.

The space complexity of the two-dimensional array C varies depending on which data structure is used. For example, we can use ones in Corollary 3.13 and Theorem 3.19. The space complexities of A, B, D, H are $\mathrm{O}(n\log n)$ bits, $\mathrm{O}(n\log n)$ bits, $\mathrm{O}(nm)$ bits, $\mathrm{O}(nm)$ bits, respectively.

The time complexity is $\mathrm{O}(t)$.

To recap, the complexities of the algorithms become $\mathrm{O}(S+nm+n\log n)$ bit space and $\mathrm{O}(T+t)$ query time, where S is the space complexity of the two-dimensional array C, and T is the time complexity to access an entry of C. ◀

Using Corollary 3.13 and Theorem 3.19, we obtain the following.

▶ **Theorem 4.15.** *There exists a data structure with space complexity* $\mathrm{O}(nm+n\log n)$ *bits in addition to the input string* S*, which answers a range mode enumeration query in time* $\mathrm{O}(\log m+t)$*.*

By combining the data structure of [3], we can further reduce the space complexity. Consider a string S_1 which stores symbols of S whose frequencies are at least $n^{1-\epsilon}$, and a string S_2 which stores the rest of the symbols. The string S_1 stores at most n^ϵ distinct symbols. Using the data structures of [3] and Theorem 4.15 for S_1 and S_2 respectively, the following holds.

▶ **Theorem 4.16.** *There exists a data structure with space complexity* $\mathrm{O}(n^{1+\epsilon}\log n+n^{2-\epsilon})$ *bits in addition to the input string* S*, which answers a range mode enumeration query in time* $\mathrm{O}(n^{1-\epsilon}+t)$*.*

The proposed data structures for the range mode enumeration problem are summarized in Table 2.

5 Concluding Remarks

In this paper, we have given more efficient algorithms for the indexing version of the range mode problem. Our algorithms are more space- and time-efficient for small maximum frequency case than existing ones. We have also considered a natural extension of the range mode problem, called the range mode enumeration problem, and given fast algorithms for it.

There are other related problems like Boolean matrix multiplication problem. As future work, we plan to give efficient algorithms for these problems.

References

1 Timothy M. Chan, Stephane Durocher, Kasper Green Larsen, Jason Morrison, and Bryan T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory of Computing Systems*, 55(4):719–741, November 2014. doi:10.1007/s00224-013-9455-2.

2 Erik D. Demaine, Alejandro López-Ortiz, and J. Ian Munro. Frequency estimation of internet packet streams with limited space. In *Algorithms - ESA 2002, 10th Annual European Symposium, Rome, Italy, September 17-21, 2002, Proceedings*, pages 348–360, 2002.

3 Stephane Durocher and Jason Morrison. Linear-space data structures for range mode query in arrays. *CoRR*, abs/1101.4068, 2011. arXiv:1101.4068.

4 Min Fang, Narayanan Shivakumar, Hector Garcia-Molina, Rajeev Motwani, and Jeffrey D. Ullman. Computing iceberg queries efficiently. In *VLDB'98, Proceedings of 24rd International Conference on Very Large Data Bases, August 24-27, 1998, New York City, New York, USA*, pages 299–310, 1998. URL: http://www.vldb.org/conf/1998/p299.pdf.

5 J. Fischer and V. Heun. Space-Efficient Preprocessing Schemes for Range Minimum Queries on Static Arrays. *SIAM Journal on Computing*, 40(2):465–492, 2011.

6 Mark Greve, Allan Grønlund Jørgensen, Kasper Dalgaard Larsen, and Jakob Truelsen. Cell probe lower bounds and approximations for range mode. In *Automata, Languages and Programming, 37th International Colloquium, ICALP 2010, Bordeaux, France, July 6-10, 2010, Proceedings, Part I*, pages 605–616, 2010.

7 Danny Krizanc, Pat Morin, and Michiel Smid. Range mode and range median queries on lists and trees. *Nordic J. of Computing*, 12(1):1–17, March 2005. URL: http://dl.acm.org/citation.cfm?id=1195881.1195882.

8 S. Muthukrishnan. Efficient algorithms for document retrieval problems. In *Proceedings of ACM-SIAM SODA*, pages 657–666, 2002.

9 Holger Petersen. Improved bounds for range mode and range median queries. In Viliam Geffert, Juhani Karhumäki, Alberto Bertoni, Bart Preneel, Pavol Návrat, and Mária Bieliková, editors, *SOFSEM 2008: Theory and Practice of Computer Science*, pages 418–423, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

10 Holger Petersen and Szymon Grabowski. Range mode and range median queries in constant time and sub-quadratic space. *Information Processing Letters*, 109(4):225–228, 2009. doi:10.1016/j.ipl.2008.10.007.

11 R. Raman, V. Raman, and S. R. Satti. Succinct indexable dictionaries with applications to encoding k-ary trees, prefix sums and multisets. *ACM Transactions on Algorithms (TALG)*, 3(4), 2007. doi:10.1145/1290672.1290680.

12 Kentaro Sumigawa, Sankardeep Chakraborty, Kunihiko Sadakane, and Srinivasa Rao Satti. Enumerating range modes. *CoRR*, abs/1907.10984, 2019. arXiv:1907.10984.

13 Kentaro Sumigawa and Kunihiko Sadakane. Storing partitions of integers in sublinear space. *The Review of Socionetwork Strategies*, 13(2):237–252, 2019. doi:10.1007/s12626-019-00044-2.

14 Dan E. Willard. Log-logarithmic worst-case range queries are possible in space $\theta(n)$. *Information Processing Letters*, 17(2):81–84, 1983. doi:10.1016/0020-0190(83)90075-3.

15 Virginia Vassilevska Williams and Yinzhan Xu. Truly subcubic min-plus product for less structured matrices, with applications. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 12–29. SIAM, 2020. doi:10.1137/1.9781611975994.2.