MATHEMATICAL ENGINEERING
TECHNICAL REPORTS

Enumerating Range Modes

Kentaro SUMIGAWA, Sankardeep CHAKRABORTY,
and Kunihiko SADAKANE

METR 2019–01 February 2019

DEPARTMENT OF MATHEMATICAL INFORMATICS
GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
THE UNIVERSITY OF TOKYO
BUNKYO-KU, TOKYO 113-8656, JAPAN

WWW page: https://www.keisu.t.u-tokyo.ac.jp/research/techrep/

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
RIKEN Center for Advanced Intelligence Project, Japan
sankar.chakraborty@riken.jp

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

Abstract

We consider the range mode problem where given a sequence and a query range in it, we want to find items with maximum frequency in the range. We give time- and space- efficient algorithms for this problem. 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 is linear to the output size plus small terms. We also give an application of our algorithm for fast set intersection problem.

1 Introduction

We consider the range mode problem, defined as follows.

Definition 1 (Mode). *Given a non-empty multiset S, $x \in S$ is said to be a* mode *of S, if its multiplicity is no smaller than those of any other elements.*

Definition 2 (Range mode problem). *For a sequence $A[0...n-1]$ and a range $[l, r]$ of A $(0 \leq l \leq r < n)$, output any one of the modes of the multiset $\{A[l], A[l+1], \ldots, A[r-1], A[r]\}$.*

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

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 [7, 1, 8, 3] and some of them are summarized in Table 1.

Our first contribution is space-efficient indexes for the range mode problem, for the case the maximum multiplicity m of an item in the set is small. Table 1 summarizes our results. The one in Corollary 26 has better time and space complexities than that of [3] with $\epsilon = 0$, which is also specialized for small m and has space complexity $\mathrm{O}(nm \log n)$ bits and query time complexity $\mathrm{O}(\log \log n)$.

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

Definition 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 *items in the multiset $\{A[l], A[l+1], \ldots, A[r-1], A[r]\}$ with the largest number of occurrences.*

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.

Data structure	Space complexity (bits)	Query time complexity	conditions
[7]	$\mathrm{O}\left(n^{2-2\epsilon}\log n\right)$	$\mathrm{O}(n^{\epsilon})$	$0 \leq \epsilon \leq 1/2$
[1]	$\mathrm{O}\left(n^{2-2\epsilon}\right)$	$\mathrm{O}(n^{\epsilon})$	$0 \leq \epsilon \leq 1/2$
[8]	$\mathrm{O}\left(\frac{n^2 \log\log n}{\log n}\right)$	$\mathrm{O}(1)$	
[3]	$\mathrm{O}((n^{1-\epsilon}m+n)\log n)$	$\mathrm{O}(n^{\epsilon}+\log\log n)$	$0 \leq \epsilon \leq 1/2$
Theorem 25	$\mathrm{O}\left(4^k nm\left(\frac{n}{m}\right)^{\frac{1}{2^k}}\right)$	$\mathrm{O}(2^k)$	k is any positive integer
Theorem 28	$\mathrm{O}(nm)$	$\mathrm{O}(\log m)$	
Corollary 26	$\mathrm{O}\left(nm\left(\log\log\frac{n}{m}\right)^2\right)$	$\mathrm{O}\left(\log\log\frac{n}{m}\right)$	

Though the problem seems to be a natural generalization of the range mode problem, there has been no existing work. 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. A simple modification of an existing algorithm [3] works, but it takes $\mathrm{O}(n^{\epsilon})$ time for each output of an item (see Theorem 32). We give faster solutions whose query time complexity is linear to the output size plus some small term. Table 2 summarizes the results.

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 summaries the paper. Some of the proofs, algorithms, and figures are given in the appendix.

2 Preliminaries

2.1 Basic properties

To avoid confusion between modes and frequency of modes, from now on we consider range mode problems for not integer sequences but strings on an alphabet. We define the following.

- S: input string
- n: the length of string S

Table 2 Complexities of data structures for the range mode enumeration problem where $|output|$ 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 does not contain that for the input sequence S.

Data structure	Space (bits)	Query time
Theorem 32	$\mathrm{O}(n^{2-2\epsilon}\log n)$	$\mathrm{O}(n^{\epsilon}\lvert output\rvert)$
Theorem 43	$\mathrm{O}\left(nm\left(\log\log\frac{n}{m}\right)^2 + n\log n\right)$	$\mathrm{O}\left(\log\log\frac{n}{m} + \lvert output\rvert\right)$
Theorem 44	$\mathrm{O}(nm + n\log n)$	$\mathrm{O}(\log m + \lvert output\rvert)$
Theorem 45	$\mathrm{O}(n^{1+\epsilon}\log n + n^{2-\epsilon})$	$\mathrm{O}(\log m + n^{1-\epsilon} + \lvert output\rvert)$

- 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, that is, $m = f(0, n-1)$

We assume that there exists a bijection $\Sigma \to \{0, 1, \ldots, |\Sigma| - 1\}$ which can be computed in constant time. We sometimes identify characters in the alphabet and integers.

Lemma 4. *[[6]] 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.*

Lemma 5. *For $l_2 < l_1 \leq r_1 < r_2$, if $f(l_1, r_1) = f(l_2, r_2)$, modes of range $[l_1, r_1]$ are also modes of range $[l_2, r_2]$.*

2.2 Algorithms for the range mode problem

We review the data structure with $\mathrm{O}(n^{2-2\epsilon})$-word space and $\mathrm{O}(n^\epsilon)$ 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 is used.

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

Two-dimensional array C : The (i, j) 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) 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 $0 \leq \epsilon \leq 1/2$. Using these arrays, any query $[l, r]$ is solved in $\mathrm{O}(n^\epsilon)$ 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 $\mathrm{O}(s) = \mathrm{O}(n^\epsilon)$ 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 4, 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 $\mathrm{O}(s) = \mathrm{O}(n^\epsilon)$ time.

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

2.3 Representations of integer sequences

Definition 6. *We define $IMS(n, u)$ as the set of all integer sequences A of length n such that $0 \leq A[0] \leq A[1] \leq \cdots \leq A[n-1] < u$.*

Theorem 7 ([10])**.** *For a sequence $A \in IMS(n, u)$ $(n > u)$ and an integer $k \geq 0$, there exists a data structure using $\mathrm{O}(2^k n^{1/2^k} u^{1-1/2^k})$ bits which can compute*

- $\mathsf{access}(i, A) = A[i]$
- $\mathsf{bound}(i, A) = \#\{j \mid A[j] > i\}$

in $\mathrm{O}(2^k)$ time.

Theorem 8 (FID [9]). *For a bit-vector B of length n which contains u ones, consider the following operations.*

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

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 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 adjacent entries in the two-dimensional array C, it holds

$$C[i][j] \le C[i][j+1] \le C[i][j] + 1 \quad (0 \le i < n,\ 0 \le j < n-1),$$
$$C[i][j] \le C[i-1][j] \le C[i][j] + 1 \quad (1 \le i < n,\ 0 \le j < n).$$

From the definition, C also satisfies:

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

Below we propose a data structure for storing C in a compressed form and supporting constant time access.

3.1 An efficient representation of the array C

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

Definition 9. *We define the set of two-dimensional arrays $A[0 \dots n-1][0 \dots n-1]$ which satisfy all the following inequalities as IMS2(n, m).*

$$A[i][j] \le A[i][j+1] \quad (0 \le i < n,\ 0 \le j < n-1),$$
$$A[i][j] \le A[i+1][j] \quad (1 \le i < n,\ 0 \le j < n),$$
$$0 \le A[i][j] < m \quad (0 \le i < n,\ 0 \le j < n).$$

Theorem 10. *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.*

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

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

Below we show such a constant c exists, if we use the data structure of Theorem 7.

For $k = 0$, we use the data structure $Z(n, m)$ of Theorem 7 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.

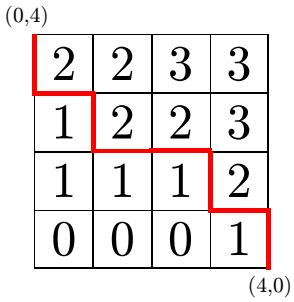

Figure 1 An *IMS2*$(4,4)$ array. The second boundary of its grid graph is shown in a red bold line. We can see that the boundary is a shortest path from vertex $(0,4)$ to vertex $(4,0)$ of the grid graph.

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

Definition 11. *A block is called* flat *if all the entries in the block are identical.*

We also define the height of a block.

Definition 12. *The* height *of block $B_{i,j}$, denoted by $d_{i,j}$, is defined as*

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

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:

Theorem 13. *Among u^2 blocks, there are at most $2um$ non-flat blocks.*

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

Definition 14. *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 $A[-1][\cdot] = A[\cdot][-1] = -1$.

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 $(i,j) \to (i,j-1)$.

Example 15. Figure 1 shows an *IMS2*$(4,4)$ array and its second boundary.

Proof of Theorem 13. It is equivalent that a block is flat, and that any boundary does not pass inside the block. For each of m boundaries, the number of blocks in which the boundary passes is $2u$. Therefore the number of blocks which contains at least one boundary in it is at most $2um$. □

Based on this property, we use the following data structure.

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 $F_{i,j} \in IMS2(t, d_{i,j})$.

It holds for the original array A, $A[i][j] = E[i/t][j/t] + F_{i/t,j/t}[i\%t][j\%t]$, and for flat block $B_{i,j}$, the two-dimensional array $F_{i,j}$ is the zero-value array. Then an access to the array A is done by Algorithm 1. At line 4 of the algorithm, 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 16 (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.

Theorem 17. *For any two distinct non-flat blocks* B_{i_1,j_1} *and* B_{i_2,j_2}*, it holds* $\Phi(i_1, j_1) \neq \Phi(i_2, j_2)$.

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 18. *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 holds block $b_{i,j}$ is not flat $\Leftrightarrow \Psi(\Phi(i, j)) = (i, j)$. To use this fact, we have to compute both Ψ and Φ. We can compute Φ at line 3 of Algorithm 1 from Definition 16. To compute Ψ, we use the following.

Lemma 19. *Assume that* $\Psi(x, y_1) = (i_1, j_1), \Psi(x, y_2) = (i_2, j_2)$*. Then if* $y_1 \leq y_2$*, it holds* $i_1 \leq i_2$ *and* $j_1 \geq j_2$.

Finally we obtain:

Theorem 20. $\Psi(x, y)$ *is computed in constant time using a data structure of* $(2\sqrt{2}c^{1/3}+2)\dfrac{nm}{t}$ *bits.*

Therefore the decision at line 4 of Algorithm 1 is done by using Algorithm 2.

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

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

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.

Lemma 21. *For the summation of all $d_{i,j}$, it holds* $\sum_{\substack{0\le i<u\\0\le j<u}} d_{i,j} \le \frac{2mn}{t}$.

Proof. From the column-wise and row-wise monotonicity, for each $l = -u+1, \ldots, u-1$, it holds $\sum_{i-j=l} d_{i,j} \le m$. By summing this for all l, we obtain the claim. □

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.

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

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 $\{0, \ldots, m-1\} \times \{0, \ldots, 2u-2\}$ and $\{0, 1, \ldots, m(2u-1)\}$, 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 $2c \cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2k}}}$. By representing P by the data structure $Z(2um, 2c \cdot 4^{k-1}nm\left(\frac{n}{m}\right)^{\frac{1}{2^{2k}}})$, it holds

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

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:

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

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

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

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

Next we consider the time T_k to access an entry of S_k. In Algorithm 1, lines 3 and 5 take T_{k-1} time. For other lines including the call to Algorithm 2 it takes constant time. Therefore it holds $T_k = 2T_{k-1} + \mathrm{O}(1)$ and we obtain $T_k = \mathrm{O}(2^k)$.

This completes the proof of Theorem 10. □

We also obtain:

Theorem 22. *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.*

Proof. We obtain the claim by letting $k = \log\log\log\frac{n}{m}$ in Theorem 10. □

By using this data structure for a two-dimensional array C satisfying Property 1, we can compute the frequency f of the modes of a query range $[l, r]$. From Lemma 5, a mode is obtained by computing $S[\min\{x \mid C[l][x] = f\}]$. To compute this, consider the following data structure.

Theorem 23. *There exists a data structure of* $\mathrm{O}(nm)$ *bits for given column number* r *of* $A \in$ *IMS2*(n, m) *and a value* h*, to compute* $\min\{x \mid A[r][x] \geq h\}$ *in constant time.*

Proof. For two-dimensional array A, consider the boundaries of Definition 14. We can say 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 $(n, 0)$. Consider to encode the m boundaries as follows.

Definition 24 (A bit-vector representation of a boundary)**.** *We encode a boundary by a bit-vector of* $2n$ *bits 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.*

The bit-vector for a boundary has n zeros and n ones. There are m such bit-vectors and the space complexity is $\mathrm{O}(nm)$ bits in total. Let B_k denote the bit-vector for the k-th boundary. From definition, it holds $\min\{x \mid A[r][x] \geq h\} = n - \mathsf{rank}_0(\mathsf{select}_1(r, B_k), B_k)$. This can be computed in constant time by using FID. □

Theorem 25. *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 4, 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 storing the rows or columns in reverse order and subtracting one from all values, C belongs to $\mathit{IMS2}(n, m)$.

In Algorithm 4, the data structures of Theorems 10 and 23 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 Theorem 10 and $\mathrm{O}(nm)$ bits for Theorem 23, 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. □

By letting $k = \log\log\log \frac{n}{m}$ in Theorem 25, we obtain:

Corollary 26. *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, the range mode problem is solved 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.

3.2 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 Definition 24.

Theorem 27. *For a two-dimensional array $A \in \mathit{IMS2}(n, m)$, there exists a data structure of* $\mathrm{O}(nm)$ *bits that given i, j, k, to decide if $A[i][j] \geq k$ in constant time.*

Proof. We store all the bit-vectors $B_0, \ldots, B_{m-1}$ 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)' size with respect to (i, j). This is done by Algorithm 3, which runs in constant time. □

From Theorem 27, 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. Furthermore, from Theorem 23, we can compute an index for modes in constant time. Now we obtain the following theorem.

Theorem 28. *In addition to the input string S, by using a data structure of* $\mathrm{O}(nm)$ *bits, the range mode problem is solved in* $\mathrm{O}(\log m)$ *time.*

This data structure is effective if $m \ll \log n$. 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 $|output|$ as a parameter.

4.1 Algorithms using existing data structures

Data structures for the range mode problem return only arbitrary one 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 29. *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 30. *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 $(t + O(1))|output|$ using s space.*

Proof. Algorithm 5 solves 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 while loop at line 4 of the algorithm is executed $|output| + 1$ times, and the total time complexity is $(t + \mathrm{O}(1))|output|$. □

Lemma 31. *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 6 gives a pseudo code. □

From Lemmas 30 and 31, we obtain the following.

Theorem 32. *There exists a data structure for the range mode enumeration problem solving a query in $\mathrm{O}(n^\epsilon|output|)$ time using $\mathrm{O}(n^{2-2\epsilon})$.*

4.2 More efficient data structures for enumeration

Definition 33. *The* mode index set *for a query range $[l, r]$ of the range mode enumeration problem is the set of the rightmost position of each mode in the query range. That is,*

$\{i \mid S[i]$ *is a mode and* $S[i] \neq S[j]$ *for any* $j = i + 1, i + 2, \ldots, l\}$.

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 follows.

$B[i][j] = 1 \Leftrightarrow i \leq j$and $S[j]$ is a mode of range $[i, j]$

Using these bit-vectors, we obtain:

Theorem 34. *The set of modes for a query range* $[l, r]$ *is* $\{x \mid f(l, x) = f(l, r)$ *and* $B[l][x] = 1\}$.

Proof. From the definition of $B[l][x]$, $S[x]$ is a mode of range $[l, r]$. From Lemma 5, $S[x]$ 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 by using Algorithm 10.

Consider 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. As for the time complexity, the algorithm executes Line 5 for $\mathrm{O}(|output|)$ times. Lines 1 and 2 takes constant time if we use data structures for bit-vectors. Therefore the total time complexity is $\mathrm{O}(|output|)$. We obtain the following basic data structure.

Theorem 35. *There exists a data structure for the range mode enumeration problem which computes the mode index set in* $\mathrm{O}(|output|)$ *time using a data structure of* $\mathrm{O}(n^2)$ *bit space in addition to the input string* S.

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

Lemma 36. *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 5, 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 37. *Define integer sequences* $H[1], \ldots, H[m]$ *as follows.*

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

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

Lemma 38. $B[i][j] = 1 \Leftrightarrow H[f(i, j)][j] \geq i$.

Example 39. Figure 2 shows an example of bit-array B and sequences H for string $S =$ "abcbfcdaacfbcgba".

Algorithm 10 enumerates indices with bits being set in range $[b, r]$ of bit-vector $B[l]$. Here for any t with $b \leq t \leq r$, the value of $f(l, t)$ is always g. 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.

Definition 40 (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* $A[l, \ldots, r]$.

Theorem 41 ([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 42. *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 it holds $A[k] \geq t$ for some k in constant time, there exists a data structure for the problem with $\mathrm{O}(n)$ bits and $\mathrm{O}(|output|)$ query time.*

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.

From the observation above, 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 $S[0, \ldots, i-1]$.

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 Definition 24.

Two-dimensional array H storing m RMQ data structures for arrays of length n each
: An array storing m sequences of Definition 37 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 Theorems 22 and 28. 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 pseudo code is given in Algorithm 9. Only Line 1 cannot be done in constant time. For other lines, the time complexity is proportional to the number of times the function *range* is executed, and it is $\mathrm{O}(|output|)$.

To recap, the complexities of the algorithms become $\mathrm{O}(S + nm + n \log n)$ bit space and $\mathrm{O}(T + |output|)$ 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 Theorems 22 and 28, we obtain the following.

Theorem 43. *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 solves a query in time* $\mathrm{O}\left(\log\log\frac{n}{m} + |output|\right)$.

Theorem 44. *There exists a data structure with space complexity* $\mathrm{O}(nm + n \log n)$ *bits in addition to the input string S, which solves a query in time* $\mathrm{O}(\log m + |output|)$.

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 44 for S_1 and S_2 respectively, the following holds.

Theorem 45. *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 solves a query in time* $\mathrm{O}(\log m + n^{1-\epsilon} + |output|)$.

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: range mode enumeration problem and given fast algorithms.

There are other related problems like Boolean matrix multiplication problem. A future work, we will 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, Nov 2014. URL: https://doi.org/10.1007/s00224-013-9455-2, 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. URL: https://doi.org/10.1007/3-540-45749-6_33, doi:10.1007/3-540-45749-6_33.

3 Stephane Durocher and Jason Morrison. Linear-space data structures for range mode query in arrays. *CoRR*, abs/1101.4068, 2011. URL: http://arxiv.org/abs/1101.4068, 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. URL: http://epubs.siam.org/doi/10.1137/090779759, doi:10.1137/090779759.

6 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.

7 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.

8 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. URL: http://www.sciencedirect.com/science/article/pii/S0020019008003116, doi:https://doi.org/10.1016/j.ipl.2008.10.007.

9 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. URL: http://doi.acm.org/10.1145/1290672.1290680, doi:10.1145/1290672.1290680.

10 Kentaro Sumigawa and Kunihiko Sadakane. An efficient representation of partitions of integers. In *Combinatorial Algorithms - 29th International Workshop, IWOCA 2018, Singapore, July 16-19, 2018, Proceedings*, pages 361–373, 2018. URL: https://doi.org/10.1007/978-3-319-94667-2_30, doi:10.1007/978-3-319-94667-2_30.

A Omitted Proofs

A.1 Proof of Lemma 4

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 multiplicity of y in M_1 is strictly larger than that of x in M_1. Because $x \notin M_2$, the multiplicity 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 that x is a mode of M. □

A.2 Proof of Lemma 5

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 $[l_2, r_2]$. □

A.3 Proof of Theorem 20

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, 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 $I_x[y] \in IMS(2u-1, u), J_x[y] \in DMS(2u-1, u)$. From Lemma 19, such sequences I_x, J_x must exist. By using the data structure $Z(2u-1, u)$ 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 $2um + 2\sqrt{2}c^{1/3}um = (2\sqrt{2}c^{1/3} + 2)\frac{nm}{t}$ bits. □

A.4 Proof of Theorem 42

Proof. We recursively find range maximum values as in Algorithm 7. Consider the number of times that the function *range* of Algorithm 8 is called. The number of times that an item is added to the set *ans* in the function is at most $|output|$. On the other hand, if $ans = \{x_1, \ldots, x_{|output|}\}$, the number of times that any item is not added to the set *ans* is at most $|output|+1$, for ranges $[l, x_1-1], [x_1+1, x_2-1], \ldots, [x_{|output|-1}+1, x_{|output|}-1], [x_{|output|}+1, r]$. Therefore the total number of times that *range* is called is at most $2|output| + 1$. From the assumption of the oracle, Algorithm 7 runs in $O(|output|)$ time. Because the algorithm uses no data structures other than RMQ, the space complexity is $O(n)$ bits. □

B Omitted Pseudo codes and Figures

Algorithm 1 Obtaining the entry $A[x][y]$ of an two-dimensional array $A \in \mathit{IMS2}(n, m)$.

Require: Indices x, y
Ensure: $A[x][y]$
$x_b \leftarrow x/t,\ y_b \leftarrow y/t$ ▷ The block number
$x_r \leftarrow x\%t,\ y_r \leftarrow y\%t$ ▷ The position in the block
$ans \leftarrow E[x_b][y_b]$
if Block B_{x_b,y_b} is not flat **then**
    $ans \leftarrow ans + F_{x_b,y_b}[x_r][y_r]$
end if
return ans

Algorithm 2 Decide if $B_{i,j}$ is flat or not

Require: Block number i, j
Ensure: If block $B_{i,j}$ is flat or not
$x \leftarrow E[i][j], y \leftarrow i - j + u - 1$ ▷ $\Phi(i, j) = (x, y)$
if $K[x][y] =$ False **then** ▷ $\Psi(x, y) = \perp$
    return block $B_{i,j}$ is flat
end if
if $I_x[y] = i$ and $J_x[y] = j$ **then** ▷ $\Psi(x, y) = (i, j)$
    return block $B_{i,j}$ is not flat
else
    return block $B_{i,j}$ is flat
end if

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
if $n - \mathsf{rank}_0(\mathsf{select}_1(i, B_k), B_k) \leq j$ **then**
    return YES
else
    return NO
end if

Algorithm 4 Algorithm for Theorem 25

Require: a query range $[l, r]$
Ensure: a mode in range $[l, r]$ of string S
$f \leftarrow C[l][r]$ ▷ using Theorem 10
if $f = 1$ **then**
    $i \leftarrow l$
else
    $i \leftarrow \min\{x \mid C[l][x] = f\}$ ▷ using Theorem 23
end if
return $S[i]$

Algorithm 5 Algorithm for the range mode enumeration problem using the data structure for finding the leftmost index of range modes

Require: a query range $[l, r]$
Ensure: the set of all range modes *ans*

```
1: ans ← {}
2: (f, i) ← D([l, r])      ▷ a pair of the frequency f of modes in range [l, r] and the leftmost index i
3: x ← l
4: while do
5:     (freq, i) ← D([x, r])
6:     if f > freq then▷ if the frequency freq of modes in [x, r] is less than the frequency f of modes in [l, r]
7:         break
8:     end if
9:     ans ← ans ∪ {S[i]}
10:    if i = r then                          ▷ if the query range becomes empty
11:        break
12:    end if
13:    x ← i + 1                              ▷ update the query range
14: end while
15: return ans
```

Algorithm 6 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 indexli, 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** $i = l, \ldots, (b_l + 1)s - 1$ **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** $i = b_r s, \ldots, r$ **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)

Algorithm 7 Find all indices in range $[l, r]$ of sequence A with frequency at least t

Require: a query range $[l, r]$, a threshold t
Ensure: set of indices ans
1: $ans \leftarrow \{\}$
2: $range(l, r)$ ▷ the function in Algorithm 8
3: **return** ans

Algorithm 8 The recursive function *range* called in Algorithm 7

Require: a range $[x, y]$
1: **if** $x > y$ **then**
2: **return**
3: **end if**
4: $id \leftarrow RMQ(x, y)$
5: **if** $A[id] \geq t$ **then**
6: $ans \leftarrow ans \cup \{id\}$
7: $range(x, id - 1)$ ▷ the range to the left of id
8: $range(id + 1, y)$ ▷ the range to the right of id
9: **end if**

Algorithm 9 Algorithm for finding the index set

Require: a query range $[l, r]$
Ensure: the index set *ans*

$g \leftarrow f(l, r)$ ▷ using data structure C
$b \leftarrow \min\{t \mid f(l, t) \geq g\}$ ▷ using boundary bit-vector D
$ans \leftarrow \{\}$
$range(b, r)$
return *ans*

def function $range(x, y)$
if $x > y$ **then**
    return
end if
$id \leftarrow RMQ(x, y, H[g])$ ▷ the index of maximum value in range $[x, y]$ of sequence $H[g]$
if $A[S[id]] - g + 1 < 0$ **and** $B[S[id]][A[S[id]] - g + 1] \geq l$ **then**
    $ans \leftarrow ans \cup \{id\}$
    $range(x, id - 1)$ ▷ the range to the left of id
    $range(id + 1, y)$ ▷ the range to the right of id
end if
end def

Algorithm 10 Algorithm for finding the mode index set

Require: a query range $[l, r]$
Ensure: the mode index set *ans*

$g \leftarrow f(l, r)$
$b \leftarrow \min\{t \mid f(l, t) \geq g\}$
$x \leftarrow r$
$ans \leftarrow \{\}$
while $x \geq b$ **do**
    $ans \leftarrow ans \cup \{x\}$ ▷ add to the mode index set
    $x \leftarrow \mathsf{select}_1(\mathsf{rank}_1(x - 1, B[l]), B[l])$ ▷ update x
end while
return *ans*

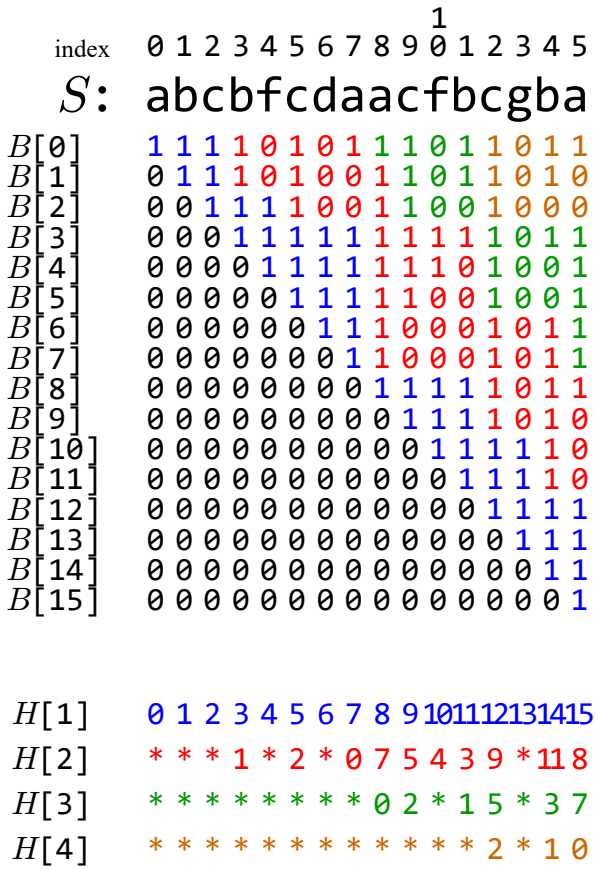

Figure 2 Bit-vectors $B[0], \ldots, B[15]$ and sequences $H[1], \ldots, H[4]$ for string S of length $n = 16$. The marks * stand for -1. Colors of numbers for B represent frequencies of modes of the corresponding ranges. Blue, red, green, and brown colors represent frequencies 1, 2, 3, and 4, respectively.